HEWLETT-PACKARD

HP-75

OWNER'S MANUAL

HEWLETT
PACKARD

HP-75
Owner's Manual

June 1986

00075-90001 Rev. D

Printed in U.S.A.

© Hewlett-Packard Company 1982

Preface

The HP-75D is the first of a new generation of Hewlett-Packard portable computers that provides the power and sophistication of a high-level language computer, combined with a truly portable computer *system*. The HP-75D has 16K (16,384) bytes of Random Access Memory (RAM) available to store user created programs, text, and variables. It also has 48K bytes of read-only memory (ROM) dedicated to its operating system.

How to Use This Manual

This manual was designed as both a *learning* and a *reference* tool. At first, you will use it to learn how your computer functions and to become familiar with all of its capabilities. Although your HP-75 was designed to be easy to use, because of its power and sophistication you'll need to invest some time in this learning process. After you learn to use your computer, the manual will serve as a comprehensive reference to every aspect of your computer's operation.

Part I of the manual, Welcome to the HP-75!, introduces you to the HP-75 and provides you with the information you need to start using your computer. Read part I first—it should only take a few hours to read the instructions and work through the examples. After completing part I, you should be able to turn the HP-75 on and off; enter and run a prerecorded BASIC program; use the keyboard to control the display; and edit and manipulate text and BASIC files in memory.

Part II, Using the HP-75, describes in detail all of the major features of the HP-75. Some of the sections in part II describe features that you may not need or want to use at this time. Feel free to skip over those sections and read only what you need. After finishing part II, you should be able to do keyboard calculations using operators and functions; use, set, and adjust the system clock; set and acknowledge appointments; perform card reader operations; use the built-in HP-IL interface; and redefine the keyboard.

Part III, Programming the HP-75, explains how to program the HP-75 in BASIC. It's written for those who have had some programming experience in BASIC or another high-level language. If you're new to programming, you may wish to study one of the many books available on beginning BASIC programming before starting part III. After completing part III, you should be able to write, edit, run, and correct BASIC programs using all of the system commands and BASIC statement available on the HP-75.

The appendices at the back of the manual provide valuable reference material. Appendix G, Glossary, defines many of the terms used in this manual—you may wish to refer to it early in your study of this manual. Appendices C, D, E, and H contain the principal reference information about the operation and programming of the computer. Appendix B contains information about caring for and obtaining service for your computer and appendix A contains a list of all of the standard accessories you received with your computer. Finally, appendix F contains listings of all of the programs in your Owner's Pac.

Two indexes are provided in this manual. A comprehensive subject index and, inside the back cover, a complete index to the HP-75 instruction set. The index to the instruction set will allow you to quickly locate the principal page reference for all of the commands, statements, functions, and operators in your computer.

The *HP-75 Reference Manual* that accompanies your HP-75 lists the instructions and operations of the computer in condensed form. It's compact enough to keep with your HP-75 wherever you take it. You can use the reference manual as a ready reference after you have completed part I of the manual and while studying parts II and III.

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

The HP-75 Portable Computing System

The HP-75 is designed so that it can be used as the controller of a portable computer *system.* The built-in HP-IL interface allows you to connect the HP-75 to a wide range of peripherals for increased computing power. You can directly connect a bar-code reading wand to the three-pin port on the back edge of the computer. It is designed to be used with an HP 92267A/B wand and an HP 82725A Bar Code Reader Module.

The three plug-in ports on the front edge of the HP-75 provide access for modules containing additional commands and BASIC statements as well as a variety of applications programs. Be sure to check the accessory brochure included with your computer to learn about Hewlett-Packard peripherals and software products that extend the capabilities of the HP-75.

Contents

HP-75 Keyboard 6

HP-75 Display Templates 6

Part I: Welcome to the HP-75!

Section 1: Getting Started 10

Introduction—The Three Operating Modes—TIME Mode—APPT Mode—EDIT Mode—The Modifier Keys—General Information—What's Ahead—Syntax Guidelines

Section 2: Keyboard and Display Control 34

Introduction—The Keyboard—The Editing Keys—Displaying Information—The HP-75 Character Set—Advanced User's Information

Section 3: File Editing 44

Introduction—Filenames—Files in Memory—EDIT Mode—Available Memory—Creating and Editing a File—Checking the System Catalog—Purging Files—Entering Lines in a Text File—Editing Operations—File Manipulations—Transforming Files—Command Summary

Part II: Using the HP-75

Section 4: Keyboard Calculations 68

Introduction—Arithmetic Operations—Multiple Calculations—The Last Result—Arithmetic Hierarchy—Parentheses—Numeric Precision—Number Formatting—Range of Numbers

Section 5: Numeric Functions and Expressions 78

Introduction—Variables—Naming Simple Numeric Variables—Assigning Values to Variables—Precision of Numeric Variables—Calculator and Program Variables—Numeric Functions—Numeric Expressions—Relational Operators—Logical Operators—Precedence of Operators—Recovering From Mathematical Errors

Section 6: TIME Mode Operations 92

Introduction—Clock Operation—Changing to Day\Month\Year and 24-Hour Formats—Adjusting the Clock—Clock Functions

Section 7: APPT Mode Operations 100

Introduction—Scheduling Appointments—The appt File—Deleting Individual Appointments—Editing Already Scheduled Appointments—Processing Due Appointments—Turning Off APPT Mode—Appointment Types—Command Appointments—New APPT Templates—The Extended Calendar—Copying Appointments To and From Mass Storage

Section 8: Card Reader Operations 114

Introduction—HP-75 Magnetic Cards—Using the Card Reader—Specifying Card Files—Cataloging a Card—From Memory To Card—From Card To Memory—Protecting Card Files—The Three P's of Protection

Section 9: HP-IL Operations 124

Introduction—Connecting the Hewlett-Packard Interface Loop—Assigning Device Codes—Listing Device Assignments—Declaring Display and Printer Devices—Displaying and Printing Information—Turning the Loop Off and On—Transmission Interruptions—Clearing Devices—Mass Storage Operations—Advanced User's Information

Section 10: Redefining the Keyboard 142

Introduction—Typing Aids—The keys File—Undoing Key Redefinitions—Immediate-Execute Keys—Fetching Key Definitions—TIME and APPT Modes—Multiple Key Operations—Multiple Keys Files—Copying Keys Files To and From Mass Storage—Advanced User's Information

Part III: Programming the HP-75

Section 11: Programming Fundamentals 156

Introduction—Writing and Editing Programs—The CHECK Program—Running Programs—Initializing Programs—Interrupting Programs—Examining Programs—Definitions—Multistatement Lines—Program Variables—Fundamental Statements—The PAYATTN Program—Using File Commands With Allocated Programs—The MANAGER Program—Turning the HP-75 Off—Errors

Section 12: Branches, Loops, and Subroutines 176

Introduction—Unconditional Branching—Conditional Branching—Looping—Subroutines—The Computed GOTO Statement—The Computed GOSUB Statement—Bypassing a Pending Subroutine Return—Program Timers

Section 13: Arrays, Strings, and User-Defined Functions 192

Introduction—Setting the Lower Bound—Declaring Arrays—Assigning Values to Arrays—String Expressions—String Functions—String Expression Comparisons—The PUT Statement—Simulating String Arrays—User-Defined Functions

Section 14: Storing and Retrieving Data 210

Introduction—Reading Data Within a Program—Rereading Data—The NAMELIST Program—Creating and Accessing Data Files—Storing Data In a File—Reading Data From a File—Closing Data Files—Serial and Random Access—Special Forms of PRINT # and READ #—Moving the Data Pointer—Statement Summary—Storing and Retrieving Arrays—Accessing Text Files—The FINDIT Program—Long Data Lines

Section 15: Program Calls 230

Introduction—Calling Programs—Comparing RUN and CALL—The FIRST and SECOND Programs—Passing Values Between Programs—Global and Local Declarations—Recursive Calls

Section 16: Display and Printer Formatting 238

Introduction—Using IMAGE—Delimiters—Blank Spaces—String Specifications—Numeric Specification—Compact Field Specifier—Replication—Reusing the Image Format String—Numeric Field Overflow—Formatting in DISP USING and PRINT USING Statements—Image Format Summary

Section 17: Debugging Operations 252

Introduction—Tracing Program Execution—Single-Step Execution—Checking a Halted Program—Processing Run-Time Errors—Error Numbers and Lines

Appendices

Appendix A: Accessories Included With the HP-75 264

Appendix B: Owner's Information 266

Appendix C: Keyboard Operations 284

Appendix D: Reference Tables 288

Appendix E: Error Conditions 298

Appendix F: Owner's Pac Program Listings 308

Appendix G: Glossary 320

Appendix H: Syntax Summary 330

Indexes

Subject Index 350

HP-75 Instruction Set Index ..... Inside back cover

HP-75 Keyboard

System keys are black; editing keys are shaded.

HP-75 Display Templates

Where the ▮ cursor symbol appears over another character, the cursor is shown as a shaded block behind the character.

TIME Mode

`SAT 02/05/1983 09:30:15 AM ▮`	Typical TIME display.
`Set Mo/Dy/Year Hr:Mn:Sc AM`	Set-time template. Used to set the clock.
`Date: MDY, ~Time: AM, Appt: YEAR`	STATS template. Used to specify date and time formats and to change the range of the appointment calendar.
`Adjust (N) ± Hr+Mn+Sc.t`	ADJST template. Used to adjust the clock setting.

APPT Mode

`Day Mo/Dy/Yr Hr:Mn AM #1N !Note`	APPT template. Used to schedule appointments.
`Rept=Mo+Dy+Hr+Mn ! DOW`	Repeat template. Used to set repeating appointments.
`Year? YYYY`	Year template. Used to schedule extended calendar appointments.

EDIT Mode

```
>
```

EDIT mode display showing the BASIC prompt and cursor.

```
:
```

EDIT mode display showing the text prompt and cursor.

```
  Name    Type Len  Time    Date
```

The first line of a CAT ALL listing.

```
ACCOUNTS    B 1242 10:20 02/21/83
```

Typical file catalog.

```
>DEF KEY 'a','a';
```

A default key definition.

```
>20 DIM A$[200],L$[96],T(6,20)
```

BASIC program statement.

Card Reader

```
Catalog card:  Align & [RTN]
```

For displaying the file catalog of a magnetic card.

```
Copy to card:  Align & [RTN]
```

For copying a file from memory to a card.

```
Verify card:  Align & [RTN]
```

For verifying the accuracy of information transferred to a card.

```
Copy from card:  Align & [RTN]
```

For copying information from a card to memory.

```
Protect card:  Align & [RTN]
```

For protecting a magnetic card from erasure.

```
Unprotect card:  Align & [RTN]
```

For removing a card's write-protection.

Hewlett-Packard Interface Loop

```
Device # 1=':  '
```

Displayed at the beginning of HP-IL device assignments.

Computer
Museum

1

Section 1

Getting Started

Contents

Introduction 10
You Can't Damage the HP-75 by Pressing Its Keys 10
Using the AC Adapter/Recharger 11
Turning the HP-75 ON ([ATTN]) 11
Turning the HP-75 Off ([SHIFT] [ATTN]) 13
Resetting the HP-75 ([SHIFT] [CTL] [CLR]) 13
The Two Most-Used Keys ([ATTN], [RTN]) 14
The Display Window 14
The Three Operating Modes 14
TIME Mode ([TIME]) 15
APPT Mode ([APPT]) 16
Scheduling an Appointment ([TAB], [BACK], [←], [→], [RTN]) 16
When an Appointment Arrives ([APPT], [ATTN]) 17
EDIT Mode ([EDIT]) 17
Keyboard Arithmetic ([+], SQR) 18
Typing in Edit Mode (' ' and " ") 19
Any Trouble ([SHIFT] [FET], [CLR], [ATTN], ERRN) 19
Writing and Running a BASIC Program ([RTN], [RUN], [ATTN]) 21
Copying a Prerecorded Program (COPY CARD) 21
Running a Prerecorded Program (EDIT, [RUN], [ATTN]) 23
Redefining a Key (DEF KEY) 25
Typing a Memo (EDIT, AUTO, PLIST) 25
Controlling HP-IL Devices 27
The Modifier Keys ([SHIFT], [CTL]) 27
The Shifted Keyboard ([SHIFT] [LOCK]) 27
The Numeric Keypad ([CTL] [LOCK]) 28
Special Display Characters ([CTL]) 28
General Information 28
Locking the HP-75 (LOCK) 28
Keeping the HP-75 On (STANDBY ON, STANDBY OFF, BYE) 29
The Beeper (BEEP, BEEP OFF, BEEP ON) 30
Keyword Abbreviations ([·]) 31
What's Ahead 31
Syntax Guidelines 32

Introduction

Congratulations on your purchase of the HP-75 Portable Computer! In order for you to start using the HP-75, there are several things you need to know right away.

You Can't Damage the HP-75 by Pressing Its Keys

You use the 65-key keyboard to interact with the HP-75. Many keystroke combinations are possible. By pressing a wrong key, you may cause an error message to appear or even erase the contents of memory, but you can't damage the HP-75 by pressing its keys.

Using the AC Adapter/Recharger

When you receive your HP-75 the rechargeable battery pack will already be installed in the computer. The battery pack may, however, be discharged initially. If it is you can operate the HP-75 immediately by using the ac adapter/recharger included with your unit. To correctly install the ac adapter/recharger:

Note: While the ac adapter/recharger is being plugged into the case of the computer or being unplugged, it's desirable to have the HP-75 turned off. If the HP-75 display is on, refer to Turning the HP-75 Off (page 13).

1. Insert the adapter plug into the adapter receptacle on the back edge of the HP-75.

2. Insert the power plug of the ac adapter/recharger into an ac power outlet.

These two steps may be reversed. Also, the ac adapter/recharger can be disconnected whenever you want to operate the unit from the battery pack.

Although the HP-75 may be operated from a power outlet with the battery pack *removed*, the contents of memory may be lost due to any outlet disconnection or line voltage variation.

It's normal for the ac adapter/recharger and the door of the battery compartment to be warm to the touch when the HP-75 operates from an ac outlet.

Turning the HP-75 On (ATTN)

What Happens At Power On. If the battery is charged or the ac adapter/recharger is plugged in, press the ATTN (*attention*) key to turn the computer on. You'll see one of two different displays:

```
Set Mo/Dy/Year Hr:Mn:Sc AM
```

The set-time template.

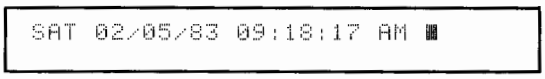
or

```
>
```

The edit mode cursor.

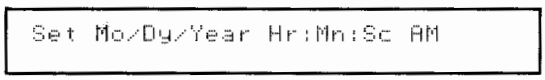
If you see the first display above, the HP-75 is signalling you to set the system clock. Skip down through this section to Setting the System Clock and continue reading from there.

If you see the second display above you need to examine the TIME display so you can check and if necessary, reset it. Press the [TIME] mode key on the upper left part of the keyboard. You will see a display that looks like this:



TIME display.

The date and time you see will, of course, be different from the one shown. If the date and time are the correct ones, you don't need to complete the steps in the rest of this section. If the date and time are *incorrect*, type the word "set" and press [RTN] (*return*) key. Now you will see the very first display shown above—the set-time template:



The set-time template.

Setting the System Clock. The set-time template contains all the information you need to set the system clock. The replace cursor, ▮, is a steadily blinking rectangle that moves across the display as you type. (This manual uses a shaded block behind the character to indicate the cursor symbol when the cursor appears over another character in the display.) Use the row of numbers on the keyboard to type the date and time information. In the set-time template above, the first digit you type will replace the M in Mo (month).

If you make a mistake, press the [BACK] (*backspace*) key or [CLR] (*clear*) key and continue typing. Pressing [CLR] allows you to start over. Supply this information in the TIME display:

The current month. Specify a number from 01 (for January) through 12 (for December). Be sure to use the [0] key when you type a zero (0) rather than the [O] key.*

The day of the month. Specify a number from 01 through 31.

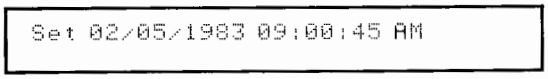
The year, a four-digit number.

The hour, a number from 01 through 12.

The minute, a number from 00 through 59.

The seconds, a number from 00 through 59. Set the seconds to a time that is approaching but still about 45 seconds away

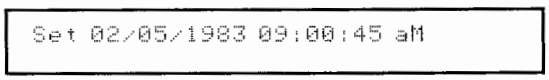
* For the month, day, hour, minute, and seconds fields, you may use a space instead of a zero. For example, 06, *space* 6, and 6 *space* in the month field all specify June.



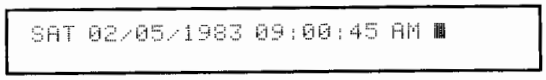
The display after the above information has been typed. In the examples, "today" begins on Saturday, February 5th, 1983, at 9:00 AM.

Type an a for AM (p for PM). Finally, at the specified time press the [RTN] (*return*) key. The [RTN] key sets the clock to the displayed time about 0.1 seconds after it's pressed.

> **Note:** Throughout this manual, side-by-side displays, such as the one that follows, show information to be entered (on the left) and the result after pressing [RTN] (on the right).



The completed template.



The day-of-week is automatically computed, and the TIME display is updated every second.

The HP-75 uses the system clock for many of its operations, which include scheduling appointments, dating files, and running programs.

Turning the HP-75 OFF ([SHIFT] [ATTN])

The [ATTN] key is the *on* button of the HP-75; pressing it turns on the display and activates the keyboard.

To turn the computer *off*, hold down the [SHIFT] key, then press the [ATTN] key and immediately release both—after a short delay (about one second) the display will turn off and the keyboard will become inactive.

The HP-75 uses power continuously to maintain system memory and operate the clock. Power consumption is higher when the computer is active (display on) and lower when it is inactive (display off). To conserve power, the HP-75 turns itself *off* after about 5 minutes of idleness. No matter how the computer is turned off, by you or by itself, the [ATTN] key will turn it back on.

Resetting the HP-75 ([SHIFT] [CTL] [CLR])

A *reset* causes the HP-75 to lose everything in memory and return to the ready state when the battery pack was first installed. There are three ways to reset the HP-75:

- Let the battery pack become completely discharged.
- Take out the battery pack while the unit is not connected to an ac source and while the display is on. The RAM circuits become discharged immediately.
- Reset the HP-75 from the keyboard using a three-key keystroke. You hold down the [SHIFT] key *and* [CTL] (*control*) key while pressing the [CLR] key for approximately 1 second.*

If you ever choose to reset the machine, you'll need to start all over by pressing [ATTN] and setting the system clock. Try [SHIFT] [CTL] [CLR] now if you'd like, but in the future it shouldn't be necessary.

To remove the battery pack without causing a machine reset, first turn the HP-75 off (press [SHIFT] [ATTN]) before sliding the battery door off. *Don't press* [ATTN] while the battery pack is out of the compartment. If the unit isn't connected to a power outlet, don't leave the battery pack out of the compartment for longer than 30 seconds.

Refer to appendix B for more battery information.

---

* In addition to [SHIFT] [CTL] [CLR], the following keystroke combinations also cause a system reset if held for one second or longer: [SHIFT] [CTL] followed by [I/R], [8], [I], [J], or [M].

The Two Most-Used Keys ([ATTN], [RTN])

The [ATTN] key has been placed in the upper-left corner for good reason—it's important. In effect, it interrupts almost all system operations and readies the HP-75 for your next keyboard entry.

While the [ATTN] key alerts the HP-75, the [RTN] key puts it to work. When you press [RTN], one or more of the following happen:

- The contents of the display are stored in memory, as happened when you set the clock.
- The command or calculation you've just typed is executed.
- The HP-75 informs you of an error by beeping and lighting an error indicator in the display window. You can recover from most errors by pressing the [ATTN] key.

Knowing when to press [RTN]—and when not to—means that you've learned to control the HP-75. You should be at this point when you've finished section 3, File Editing.

The Display Window

The display window shows 32 characters at a time. Each display *line* holds up to 96 characters including the cursor. When you type a message that completely fills the 32 character window, entering more characters causes the message to scroll left across the display.

The display window contains four *annunciators*, or status indicators, that tell you about special operating conditions.

BATT	ERROR	PRGM	APPT

Usually invisible, they appear when:

BATT: Battery voltage is low.

ERROR: The HP-75 doesn't understand one of your instructions or has encountered a mistake during a program.

PRGM: A program is currently running.

APPT: An appointment has come due.

The Three Operating Modes

Pressing any one of three keys—[TIME], [APPT] (*appointment*), or [EDIT]—causes the HP-75 to switch to the specified *mode* or operating state.

TIME mode is used for:

- Setting the system clock.
- Displaying the time.
- Specifying new time and date formats.
- Adjusting clock speed.

APPT mode is used for:

- Scheduling personal and business appointments.
- Checking calendar dates.

EDIT mode—the "workhorse" of the three—is used for:

- Doing keyboard calculations.
- Writing and running BASIC programs.
- Copying information to and from magnetic cards.
- Redefining individual keys and keystroke combinations to display messages and execute commands.
- Writing memos.
- Controlling Hewlett-Packard Interface Loop printers and other HP-IL peripherals.

Press each of the three keys now:

[TIME]:

```
SAT 02/05/1983 09:15:47 AM
```

Switches the HP-75 to TIME mode.

[APPT]:

```
Day Mo/Dy/Yr Hr:Mn AM #1N !Note
```

Switches the HP-75 to APPT mode.

[EDIT]:

```
>
```

Switches the HP-75 to EDIT mode.

The capabilities of all three modes—TIME, APPT, and EDIT—will be briefly introduced in this section.

TIME Mode ([TIME])

The HP-75 enters TIME mode when you press the [TIME] key.

You can set the clock to a different time by typing the word set in the TIME display and pressing the [RTN] key:

```
SAT 02/05/83 09:18:17 AM set
```

```
Set Mo/Dy/Year Hr:Mn:Sc AM
```

The set-time template appears.

Press [ATTN] to cancel this procedure, or type in the new information and press [RTN]. If you omit any information (like the month, day, or year), then it's supplied from current date and time values.

The SET command is one of five TIME mode commands. Section 6, TIME Mode Operations, explains how to use these commands to specify new display formats and adjust the clock. The examples in this manual usually show month/day/year and AM/PM formats.

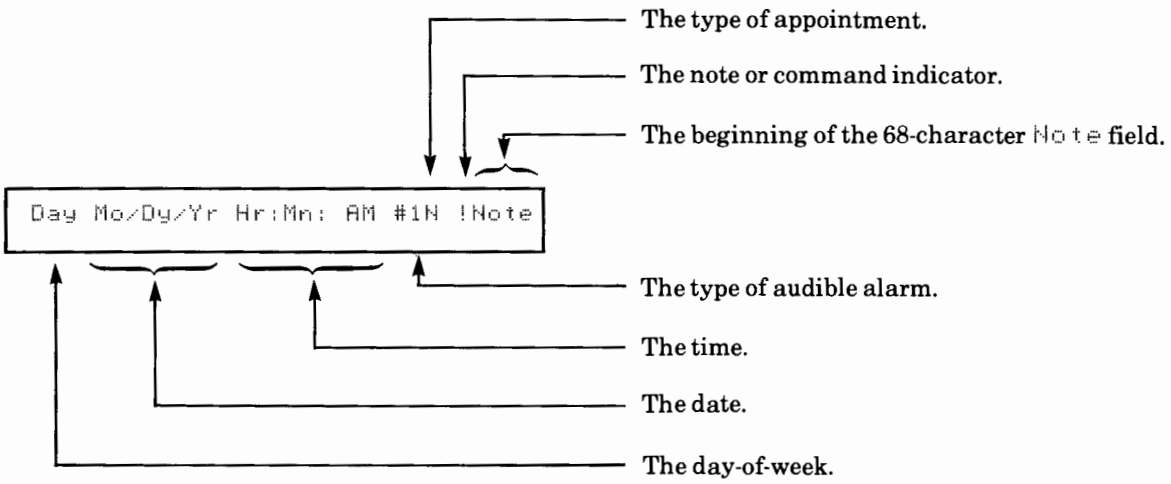
Note: As you're reading this manual, the display may go blank—remember that this means the HP-75 has turned itself off. Press [ATTN] to turn the HP-75 back on. The computer will always turn on in EDIT mode.

APPT Mode (APPT)

The HP-75 comes equipped with two calendars—a year calendar and an extended (10,000-year) calendar—that allow you to schedule alarms, messages, and programs. APPT Mode combines the accuracy of the HP-75 clock with the capability of the HP-75 microprocessor. It's possible to schedule more than 3000 different appointments on the HP-75C, and each appointment can keep repeating itself at intervals from 1 minute to 8 years.

Scheduling an Appointment (TAB , BACK , ← , → , RTN)

To schedule an appointment, switch the HP-75 to APPT mode—press APPT. The APPT template appears:



These *fields*, or display areas, are explained in detail in section 7, APPT Mode Operations. For now, here are the keys that help you schedule appointments:

- The TAB key causes the cursor to skip forward across the APPT template. Also, SHIFT TAB causes the cursor to skip backwards.
- The BACK, ← (*left-arrow*), and → (*right-arrow*) keys let you make changes in what you've typed.
- The CLR key clears the line and restores the APPT template.

For this example, press the TAB key six times to skip to the 1 in the alarm field. Then press 3 to set a two-tone alarm pattern:

```
Day Mo/Dy/Yr Hr:Mn AM #3N !Note
```

The APPT template after you've specified the type of alarm. Ten types are available (page 101).

Press TAB once more to move the cursor to the start of the Note field.

When the appointment comes due, your note will appear with it. We use a short note: Phone Jim: 555-1104.

```
:Mn AM #3N !Phone Jim: 555-1104
```

Your appointment message may contain up to 68 characters—more than two full display windows.

The ← and → keys let you review the completed appointment. When the appointment is typed the way you want it, press RTN, which stores the appointment in memory. The completed appointment is echoed:

```
SAT 02/05/83 09:25 AM #3N !Phone
```

The sample appointment is scheduled to go off at the beginning of the next minute.

You don't have to fill in the APPT template completely because the HP-75 calculates values for the fields you leave blank, based on current date and time values and on the values you do supply. Note that the cursor skips the Yr (year) field altogether; the HP-75 is set to the year calendar, so supplying year information is unnecessary if your appointment falls before the end of the next calendar year.

Press [CLR] to display the unfilled APPT template.

When an Appointment Arrives ([APPT], [ATTN])

Whether you're in TIME, APPT, or EDIT mode, when an appointment arrives, the alarm sounds and the APPT annunciator turns on. For this example, press the [TIME] key to check the time while waiting for the appointment to come due. When the seconds counter reaches 00, you'll see:

```
SAT 02/05/1983 09:25:00 AM
                        APPT
```

During TIME mode, an appointment comes due, sounding the alarm and turning on the APPT annunciator.

Pressing [APPT] brings the due appointment to the display:

```
SAT 02/05/83 09:25 AM #3N !Phone
                        APPT
```

The day-of-week, date, and time are underlined, signifying that this is a due appointment.

Press [ATTN] to *acknowledge* the appointment and to turn off the APPT annunciator. Use the [↑] (*up-arrow*) and [↓] (*down-arrow*) keys to review earlier and later appointments—they're stored according to their arrival times.

If an appointment comes due after the the HP-75 has turned off, then the HP-75 sounds the alarm, turns on in EDIT mode, and displays the note. To see this, turn the machine off—press [SHIFT] [ATTN].

```
!Phone Jim: 555-1104
```

When the HP-75 is turned off, it *processes* due appointments and past due appointments. In this case, the HP-75 displays the appointment note.

The note is held in the display for 5 seconds or until you press any key; afterwards, the HP-75 turns itself off. When you turn the machine back on (press [ATTN]), the APPT annunciator will appear if there are any appointments waiting to be acknowledged.

Besides displaying notes, appointments can be scheduled to run programs and to execute other programming statements. Refer to section 7 for more information.

EDIT Mode ([EDIT])

EDIT mode is the powerhouse of the HP-75. Practically all HP-75 operations are performed in EDIT mode. When the machine is turned on, or when the [EDIT] key is pressed, the HP-75 switches to EDIT mode.

```
>
```

The display indicates EDIT mode.

The BASIC prompt, a > symbol, appears at the left edge of the display line and indicates that the HP-75 is ready for your next command, program statement, or keyboard calculation. There is also a text prompt, a : (colon), that is used while you type memos and other correspondence.

To get the BASIC prompt or the text prompt in the display—if it isn't there already—you follow these steps:

1. Press [EDIT].
2. Type purge and press [RTN]. The PURGE command erases the computer's current workspace.

3. Choose one:

   - For the text prompt (:), type `edit text` and press [RTN]:

```
workfile    T     0 09:32 02/05/83
```

The display shows the catalog entry of a temporary text `workfile`.

   - For the BASIC prompt (>), type `edit basic` and press [RTN]:

```
workfile    B     0 09:32 02/05/83
```

The display shows the catalog entry of a temporary BASIC `workfile`.

   The time of the file's creation is expressed in 24-hour notation; for example, 1 p.m. is displayed as `13:00`. A `0` indicates the file is empty.

4. Press [ATTN]. The appropriate prompt will appear:

```
:
```

The text prompt.

```
>
```

The BASIC prompt.

The meaning of these steps will become apparent in section 3, File Editing.

Keyboard Arithmetic ([+], SQR)

The HP-75 serves as a powerful calculator in EDIT mode, equipped with 64 built-in functions and operators. You should have the BASIC prompt (>) in the display for keyboard calculations. (Refer to the preceding topic if the text prompt is there instead.) To add a series of numbers, type in the expression and press [RTN].

Example:

```
>1+2+3+4+5+6
```

An equals sign (=) is *not* used for keyboard arithmetic.

```
 21
```

Pressing the [RTN] key returns the answer.

The period (.) serves as a decimal point. Other arithmetic operators besides addition include subtraction (-), multiplication (*), division (/), and exponentiation (^). You can evaluate any combination of integers and decimals, using whatever operators you need. The HP-75 calculates to 12-digit precision. Section 4 details the use of the HP-75 for keyboard arithemetic.

To take the square root of a number, say 78, type the following and press [RTN]:

```
>sqr(78)
```

When typing the SQR function, the prompt reappears as soon as you press the [S] key.

```
 8.83176086633
```

The answer. Rounding occurs at the least significant (12th) digit.

Numbers, functions, and variables can be combined with arithmetic operators (such as +), relational operators (such as >), and logical operators (such as AND) to form complex expressions. Section 5 discusses HP-75 numeric functions and expressions.

Typing in EDIT Mode (' ' and " ")

When the BASIC prompt (>) is present, the HP-75 treats the lowercase letters in your entries the same as capital letters. This means you can use either lowercase or uppercase letters—whichever are easier—to type your commands and programs. The examples in this manual use mostly lowercase letters.

In EDIT mode, the HP-75 generally disregards the spacing of your entries so that you can use as many or as few spaces as you choose, except that *the first two* characters on a line must appear with no spaces between them, and consecutive digits of a number must have no spaces between them.

Example:

```
>sQ  r (64)
```

Arbitrary case and spacing.

```
8
```

When you press [RTN], the square root of 64 is computed.

You can preserve spaces and lowercase letters by enclosing your entries with quotation marks. It makes no difference whether you type single quotes (' ', produced by [SHIFT] [7]) or double quotes (" ", produced by [SHIFT] [2]), as long as they match. Within pairs of quotation marks, characters are interpreted exactly as you type them.*

Example:

```
>'The quick';" brown";' fox'
```

Semicolons separate the three quoted strings.

```
The quick brown fox
```

Pressing [RTN] causes the strings to be echoed as typed.

This manual generally uses single quote marks (' ') to delimit, or set off, strings of characters.

A typing convenience is the repeating keyboard: Holding down any character key or keystroke combination causes the key display character to be repeated after a short delay. For example, the [B] key:

```
>bbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
```

Holding down [B] produces three windowsful of b's. The line begins scrolling when the cursor reaches the right edge of the display.

System keys (like [TIME]) and editing keys (like [TAB]) also repeat their functions when held down.

The first display position is reserved for the prompt; the last, or 96th, position, is reserved for the cursor. This means that you may type 94 characters in each line. The HP-75 beeps when you type the 91st character—you can type three more characters before the cursor stops at the end of the line.

Any Trouble? ([SHIFT] [FET], [CLR], [ATTN], ERRN)

If the HP-75 beeps and turns on the **ERROR** annunciator while you're typing, the computer is signaling that it's tried unsuccessfully to interpret a command or an input or that it's attempted an improper operation. The word ERROR or WARNING and a brief message will appear in the display and, usually, what you've just typed will be recalled to the display. For example, press [CLR], type an incorrect expression, and press [RTN]:

---

* Exceptions are *filenames*, that is, quoted names that specify the files in memory. User filenames are converted to uppercase in the system catalog (page 45).

```
>2+6x3
```

The multiplication operator is *, not x.

```
ERROR: extra characters
                ERROR
```

The HP-75 response. The error message stays in the display for about 1 second.

Afterwards, the original expression is recalled:

```
2+6x3
      ERROR
```

The cursor returns to the position where the HP-75 first detected an error.

The message `extra characters` means that the HP-75 is unable to interpret the characters after `6`, the last meaningful character in the line.

There's no need to worry if an error occurs—no keyboard or program operation is capable of damaging the system. You've got several options:

View the Error Message Again. Hold down [SHIFT] while pressing the [FET] (*fetch*) key and *keep the* [FET] *key down.*

Example:

```
ERROR: extra characters
                ERROR
```

Pressing [SHIFT] [FET] displays the current error message.

Correct the Display. Use the line editing keys (in this case, the [←] key), to reposition the cursor and then type the necessary corrections:

```
>2+6*3
      ERROR
```

The corrected line. After typing the *, press [RTN].

```
20
```

The expression is interpreted correctly as 2 + (6 * 3). (Multiplication occurs before addition unless you change the order with parentheses.)

Clear the Display. Press the [CLR] or [ATTN] key and then retype the entry. Once the ERROR annunciator disappears, the error condition is cleared. Afterwards, pressing [SHIFT] [FET] displays a blank line.

View the Error Number. Each error has an identification number as well as a message. The `ERRN` function returns the identification number of the error which occurred most recently:

```
>errn
```

```
84
```

Pressing [RTN] returns 84, the number associated with the `extra characters` error.

Appendix E is a table of error numbers, messages, and conditions. Use the `ERRN` function to locate a specific entry in the table.

An error may cause either an `ERROR` or a `WARNING` message to appear. The difference is that an error condition will halt a running program, while a warning condition will cause a default (or predetermined) value to be supplied and allow program execution to continue. This manual refers to both errors and warnings as *errors* except where noted.

Writing and Running a BASIC Program ([RTN], [RUN], [ATTN])

The BASIC prompt in EDIT mode indicates the computer's readiness to accept BASIC program statements. Here's a one-line "random music" program for you to enter and run. Press [CLR] and type the following:

```
>1 beep rnd*999,rnd*.5 @goto 1
```

Two program statements are joined in one line. Spacing isn't important.

```
>
```

Pressing [RTN] stores the program line in memory.

Press the [↑] key to review the program:

```
>1 BEEP RND*999,RND*.5 @ GOTO 1
```

The HP-75 has converted the letters to uppercase and corrected the spacing for readability.

Now press [CLR] or [ATTN] to clear the display. To run the random music program, press the [RUN] key or type RUN followed by a [RTN]. To stop the program, press [ATTN].

Notice that the PRGM annunciator appears in the display window to indicate that a program is running.

Except for [ATTN], the keyboard is disabled during program execution to prevent unintentional disruption.

After you've stopped the program, give it a *filename* using the RENAME command. Type:

```
>rename to 'singsong'
```

Pressing [RTN] gives the current program a name, SINGSONG, converted internally to uppercase.

Copying a Prerecorded Program (COPY CARD)

One of the functions of the COPY command is to copy prerecorded programs from magnetic cards into memory. An example is the MONEY program, one of the three prerecorded programs included with your HP-75. Find and remove the two MONEY magnetic cards from the card holder shipped with your unit.

CAUTION

Handle magnetic cards by their edges to avoid contaminating the sensitive magnetic surface and the card reader head. Card cleanliness is important for proper card reader performance.

Never pass a magnetic card through a strong magnetic field. Such action will render the card totally unusable.

Protect your magnetic cards against scratches, creases, and dirt by replacing them in the card holder as soon as possible after use.

The COPY command initiates the card reader operation.

```
>copy card to 'money'
```

The command specifies the filename you want for the program. Press [RTN].

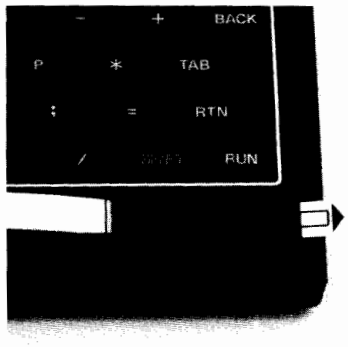
```
Copy from card:  Align & [RTN]
```

The response shows the HP-75 is ready for a card reader operation.

If you want, you can cancel the card reader operation now by pressing [ATTN]. Otherwise, the HP-75 waits for you to insert a card and to press the [RTN] key.

Each magnetic card may store information on two *tracks*. To copy the information on a full card, you'll need to copy the card in both directions. The order of the cards and the number of repetitions are immaterial.

Insert a card printed side up, either edge toward you, and gently slide it in the direction of the card reader arrow. Keep inserting the card until the rightmost alignment mark is under the entry slot and the alignment mark to its left remains exposed. About 1 cm (⅜-½ in.) of the end of the card should protrude from the exit slot.

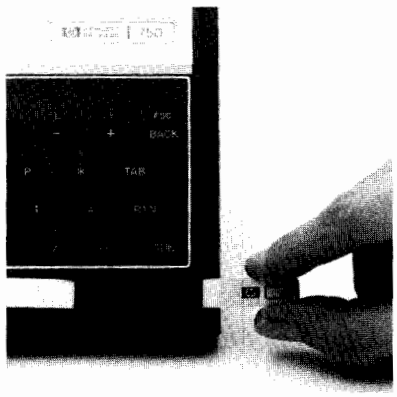


With the card aligned in the card reader, press [RTN]. The display will show:

```
Pull card ...
```

Ready for the first pass.

Grasp the right end of the card between your right thumb and index finger and pull smoothly and steadily.



A pull through the card reader should take about as long as it does to say the name "Hewlett-Packard." The speed of the card may range from 13 to 76 cm/second (5 to 30 in./second). It may be helpful to do a few trial pulls to search for the lower and upper speed limits. You may pull the same track through the card reader an unlimited number of times. The HP-75 display indicates immediately whether or not the pass has been successful.

Examples:

```
WARNING: pulled too fast
```

Indicates you've pulled the card too fast.

```
Track 2 done; insert track 1
```

Indicates that track 2 has been read and the card may be turned around for reading the other track.

```
WARNING: bad read/write
```

Indicates that the card was not read correctly. If this warning occurs repeatedly, you should clean the card (refer to appendix B).

After the message is displayed, you'll again see:

```
Copy from card:  Align & [RTN]
```

Indicates you may insert the same track or another track from the same program.

Press [RTN] before each pull. The HP-75 continues to prompt you as it guides you through the card reading process. After all four tracks of the program have been read, the cursor and prompt will reappear:

```
>
```

This display means the operation is finished. All four tracks of the MONEY program have been copied to memory.

Section 3, File Editing, shows the reverse process, which you use to copy information from memory to cards. Section 8, Card Reader Operations, details other features of the card reader, including password and private card options.

Running a Prerecorded Program (EDIT, [RUN], [ATTN])

To run the MONEY program, first find the program in memory using the EDIT command (type in edit, don't use the [EDIT] key):

```
>edit 'money'
```

Pressing [RTN] will display the catalog entry, or heading, of the program.

```
MONEY        B 2205 15:23 04/22/82
```

The size (in bytes) and date of the program may vary, depending on your version.

Note: If ERROR: workfile name? occurs when you enter an EDIT command, it means some "housecleaning" is necessary. Type purge [RTN] to erase the current workspace and then retype the EDIT command. This file-handling procedure is explained in section 3, page 63.

The MONEY program offers you two options:

Savings Option. The program calculates the *future amount* of money you'll have saved, given an initial deposit amount, the number of compounding periods, and the periodic interest rate. You may also specify a periodic deposit amount.

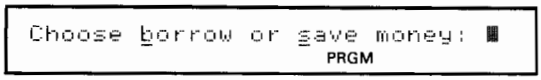
Borrowing Option. The program calculates the amount of periodic payment you'll need to pay out, given an initial loan amount, the number of compounding periods, and the periodic interest rate.

Example: If you deposit $30 per month (beginning today) in a current account of $2000, with 17% interest compounded monthly, how much will you have in 9 months? You'll use the arithmetic keys (digits, decimal point, and operators) when entering this information.

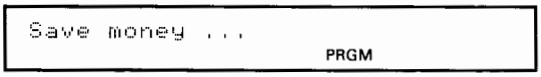
Press [RUN] or type run [RTN]. You'll see the **PRGM** annunciator and the following display:

```
~ M O N E Y ~
PRGM
```

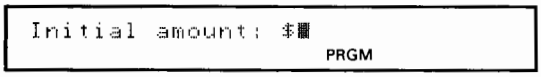
The program greeting is displayed momentarily, followed by your two options.



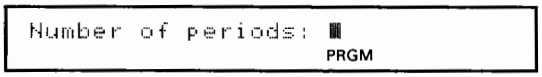
Type an s or S to choose the savings option. Then press [RTN].



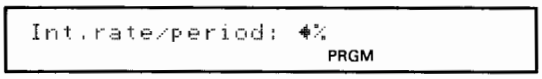
This message confirms your choice.



Type 2000 [RTN]. You may enter numbers with decimal points and one operator (such as [+] or [*]), but don't type dollar signs or commas.

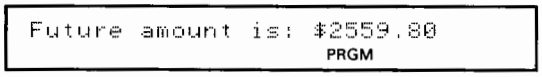


Type 9 [RTN] to specify nine compounding periods.



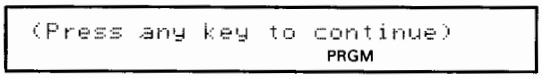
If the interest is 17% annually, then for each *period*, or month, it's 17/12%. Type 17/12 [RTN].

Type 30 [RTN]. (The assumption is that a deposit will be made at the *beginning* of each period.)



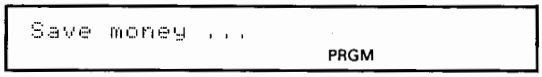
At the end of 9 months, you can expect $2559.80.

The **PRGM** annunciator remains on—the MONEY program continues executing, waiting for your response. After about 5 seconds, the HP-75 displays:



The HP-75 waits for a key response. If no key is pressed within a few seconds, the program will cycle back and forth between this display and the previous one.

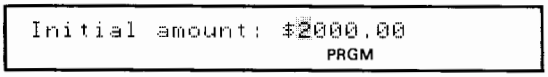
To stop the program, press [ATTN], To compute another savings amount, press any key but [ATTN], say [TAB]. The program will automatically stop after 2 minutes if no key is pressed. After the program terminates the display will be: END of 'Money' program.



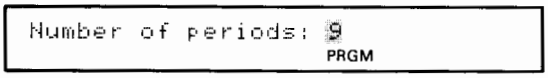
Indicates that execution has returned to the beginning of the savings option.

This time through, the HP-75 remembers and displays the figures you've entered from before.

Example:

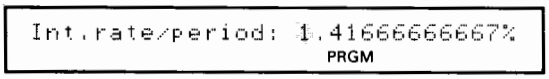


If you're satisfied with this amount, bypass the entry by typing nothing and pressing [RTN]. Otherwise, type right over the amount shown and then press [RTN].



You may leave this number alone or type another over it.

All dollar amounts are displayed with two decimal places. The interest rate is displayed with 1 to 12 digits. In this example:



Note that a *period* may be any unit of time. The example uses *monthly* compounding periods.

To figure the periodic payments you'd need to make on a loan, stop the program (press [ATTN]), restart it (press [RUN]), type a b [RTN] (for *borrow*), and you're in business.

Redefining a Key (DEF KEY)

The DEF KEY command enables you to redefine individual keys so that they function as typing and programming aids.

Example: Change the unshifted [Q] key so that it displays your name instead of lowercase q. Type the following and press [RTN]:

```
>def key 'q','Your Name';
```

The DEF KEY command. Be sure to end the command with a semicolon (;) before pressing [RTN].

```
>
```

The new definition of the [Q] key is stored in memory.

Now when you press [Q], you'll see your name. Holding down [Q] will cause your name to be repeated across the display line. Change the [Q] key back to its original definition by clearing the display and typing:

```
>purge keys
```

Pressing [RTN] clears that part of memory that holds your key redefinitions.

If you've redefined a key you need to type with, type it in uppercase instead. For example, if you've redefined the [P] key, type [SHIFT] [P] to display an uppercase P.

The DEF KEY command makes possible 190 key redefinitions. *Shifted keystrokes* (for example, [SHIFT] [Q]) as well as *control* keystrokes (for example, [CTL] [Q]) are user-definable. Refer to section 10, Redefining the Keyboard, for more information.

Typing a Memo (EDIT, AUTO, PLIST)

The HP-75 has text-editing capabilities that enable you to draft and revise correspondence in EDIT mode. For example, to write a short memo named RAYMEMO, first create a file by that name in memory. Type the following and press [RTN]:

```
>edit 'raymemo', text
```

The EDIT command creates file RAYMEMO and switches the HP-75 to text-editing.

```
RAYMEMO     T     0 10:16 02/05/83
```

The catalog entry appears, showing the name of the file, its type (text), its size (0 bytes), and the time and date of creation.

When you press [ATTN] now—or any other key—the text prompt (:) appears.

```
:
```

Pressing [ATTN] displays the text prompt (:) and cursor.

Now type a . (an abbreviation for auto) and press [RTN].

```
:a.
```

To start automatic line numbering.

```
:10
```

Numbering begins at line 10.

Then enter the memo by typing the lines and pressing [RTN] after each line.

```
:10 TO Ray
```

The first line of the memo. Use the [SHIFT] key to type capitals.

```
:20
```

The numbering increases in steps of 10.

```
:20 FROM Bill
```

The second line.

```
:30
```

```
:30 SUBJECT Computers
```

The third.

```
:40
```

```
:40 These machines are amazing.
```

Last line.

```
:50
```

Line 50 won't be used.

Press [ATTN] to stop the process. Then type `delay 2` [RTN] to set a comfortable reading rate. Finally, execute the `PLIST` (*print-list*) command:

```
:plist
```

Press [RTN] to review what you've typed.

The `PLIST` command causes file lines to be displayed without line numbers, one after another, with a 2-second delay between lines.

```
TO Ray
FROM Bill
SUBJECT Computers
These machines are amazing.
```

The last line listed. Press [CLR] to clear the line.

Your lines are displayed exactly as you've typed them, without line numbers.

To single-step through the text file, use the [↑] and [↓] keys. You can add as many new lines as you choose. Type a line number and then the text, and press [RTN]. Note that a line number should be separated from the first *digit* in a line by one or more spaces or characters.

Examples:

```
:50 30 days hath September.
```

This line would be stored as line 50, containing `30 days hath September.`

```
:5030 days hath September.
```

This line would be stored as line 5030, containing only `days hath September.`

To finish the text-editing example and return to the `MONEY` program, execute another `EDIT` command.

```
:edit 'money'
```

To return to a program already in memory.

```
MONEY        B 2205 15:23 04/22/82
```

The heading of the program is displayed.

Press [CLR] to clear the line:

```
>
```

The BASIC prompt and cursor reappear

You can return to a program or text file at any time using the EDIT command.

Example:

```
:edit 'raymemo'
```

Specify the name between quotes in uppercase or lowercase letters.

```
RAYMEMO     T    90 10:16 02/05/83
```

The catalog entry shows that the file takes 90 bytes of memory.

Press [CLR] and the text prompt will reappear. Finally, type `edit 'money'` [RTN] to return once more to the MONEY program.

The SINGSONG, MONEY, and RAYMEMO files remain in memory until you deliberately erase them, as with a PURGE command (page 50). You can create and edit as many files as memory can hold. The HP-75C can hold more than 150 memos the size of RAYMEMO. Section 3, File Editing, gives complete instructions for creating, revising, and manipulating files.

Controlling HP-IL Devices

From EDIT mode, you can control a variety of HP-IL (Hewlett-Packard Interface Loop) devices—such as the HP 82163 Video Interface, the HP 82162A Thermal Printer, and the HP 82161A Digital Cassette Drive. You may want to refer right now to the first part of section 9, HP-IL Operations, if you have a peripheral to connect to the HP-75.

The Modifier Keys ([SHIFT], [CTL])

Although the HP-75 has a simple and uncluttered keyboard, the computer can display 256 letters, digits, punctuation marks, Greek symbols, and other characters. The HP-75 uses many keystroke combinations to multiply the power of the keyboard. All keystroke combinations include the [SHIFT] key, the [CTL] key, or both.

Just as word prefixes modify the meaning of words they precede (as "un-" and "re-" change the meaning of "do"), so [SHIFT] and [CTL] modify the function of keys they precede. Like prefixes, the modifier keys [SHIFT] and [CTL] aren't used alone; they must be held down as you press another key. Refer to appendix C, Keyboard Operations, for a listing of possible keystroke combinations.

The Shifted Keyboard ([SHIFT] [LOCK])

As on a typewriter, keys [A] through [Z] display uppercase or lowercase letters, depending on whether they're pressed with or without the [SHIFT] key.

You can lock the keyboard in uppercase by holding down [SHIFT] while pressing the [LOCK] key. Afterwards, unshifted letter keys will display uppercase and *shifted* letter keys will display *lowercase* letters. ([SHIFT] [LOCK] has no effect on nonletter keys.) The keyboard will stay locked in uppercase while the HP-75 is turned off.

To lock the keyboard in lowercase again, press [LOCK] by itself. The PUT statement discussed in section 13 enables programs to lock the keyboard in uppercase and lowercase.

The Numeric Keypad (CTL LOCK)

To help you key in numbers, the HP-75 comes equipped with a numeric keypad. The illustration below shows the location of the keypad. You can also use the keypad overlay that is included with the HP-75.

To use the keypad, press the CTL key and hold it down while you press LOCK. The keypad, the digit keys, and the arithmetic operator keys are enabled; unused letter keys are disabled and cause the beep to sound when they're pressed. The numeric keypad has no effect on *shifted* keys. For example, the unshifted W key outputs a beep, while SHIFT W displays uppercase W.

The numeric pad remains in effect while the HP-75 is turned off. To restore the regular keyboard, press LOCK.

Special Display Characters (CTL)

If you hold down the CTL key while pressing alphabetic keys (like D) and a few other keys (like the space bar), they generate special display characters called *control characters*. Hold down CTL while pressing D and then Q: the ∝ and Ω characters appear. Section 2 discusses the entire 256-character set of the HP-75.

General Information

Locking the HP-75 (LOCK)

Using the LOCK command, you can "lock" the HP-75 against unwanted use by others who don't know your password. The password may be any combination of up to 90 letters, numbers, spaces, and symbols, although the computer checks the first 8 characters only. We recommend you choose an easy-to-remember password because the lock is *absolute*. If you forget the password, you have to start all over by resetting the machine.

When you type the LOCK command, use quotation marks to delimit the password.

Example:

```
>lock 'Ben'
```

Using a quoted name as the password. Double quotes work just as well.

```
>
```

The HP-75 secured with password Ben.

The HP-75 will ask for the current password each time it turns on. Check this by pressing [SHIFT] [ATTN] to turn the machine off and then pressing [ATTN] to turn the machine back on.

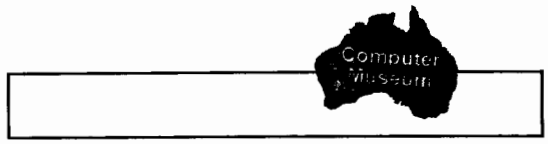
```
password? ■
```

Once on, the HP-75 prompts for your password.

Type it incorrectly and press [RTN] to see what happens:

```
password? ben■
```

Lacks an uppercase B.



So the machine turns itself off.

To regain control of the HP-75, you must type in the password *exactly* as it appeared in the LOCK command.

Example:

```
password? Ben■
```

Typed correctly. Quotes aren't used.

```
>■
```

The prompt and cursor appear.

If and only if the first eight (or fewer if the password is shorter) characters that you've entered agree with the first eight characters of the password, the HP-75 returns to EDIT mode, ready for your next entry. To remove an existing password, type `lock ''` [RTN] with *no space* between the quote marks. In effect, this locks the HP-75 with a null, or nonexistent, password.

The HP-75 may be locked and unlocked under program control by means of the LOCK command (page 174).

Keeping the HP-75 On (STANDBY ON, STANDBY OFF, BYE)

The STANDBY ON command lets you override the computer's normal 5-minute *timeout period.*

```
STANDBY ON
```

Type `standby on` [RTN] to cause the HP-75 to stay on indefinitely. To restore the 5-minute timeout period type `standby off` [RTN].

```
STANDBY OFF
```

The advantage of STANDBY ON is that it keeps the display on while you're not using the machine. For example, the HP-75 can serve as a desk-top clock. The advantage of STANDBY OFF is that it conserves battery power when the HP-75 is not connected to a power outlet. Fully charged nickel-cadmium batteries typically offer approximately 20 hours of STANDBY ON operation.

If you turn the HP-75 off—that is, if you press [SHIFT] [ATTN] or use the BYE command (type `bye` [RTN])—the previous STANDBY status is restored when you press [ATTN].

```
BYE
```

The BYE command can be used with program timers to turn the HP-75 off and on at regular time intervals under program control (page 187).

Note that the STANDBY OFF timeout period will lengthen to 10 minutes if you leave the computer with a result displayed in the window. For example, if you type 60*60*24 [RTN] and leave the 86400 in the display, then that result will stay in the display for 5 minutes; afterwards, the prompt and cursor will reappear; 5 minutes later, the HP-75 will turn itself off. If STANDBY ON is in effect, a result will remain in the display indefinitely.

The Beeper (BEEP, BEEP OFF, BEEP ON)

The BEEP statement produces an audible tone from the built-in speaker. There are three forms of the statement:

```
BEEP
BEEP frequency in hertz
BEEP frequency in hertz , duration in seconds
```

Typing beep [RTN] causes a 1400-hertz tone to sound for 0.1 second. Specifying the frequency causes a tone to sound for approximately 0.1 second.

Example:

```
>beep 261
```

Pressing [RTN] causes a middle C to sound for approximately 0.1 seconds.

You can specify any frequency, although tones from approximately 100 to 1400 hertz offer greatest fidelity.

The duration parameter sets the length of tone, accurate to tenths of a second.

Example:

```
>beep 220, 1.5
```

Pressing [RTN] causes the A below middle C to sound for 1.5 seconds.

Frequency values less than approximately 10 and nonpositive duration values cause the speaker to "click". Frequencies and durations can be specified by numeric expressions as well as by numbers (for example, BEEP 440*2^(1/12),L). The upper limit for duration is 2^{26} seconds. Values greater than 2^{26} default to this limit.

Pressing [ATTN] interrupts any beep.

Note: The speaker draws a significant amount of current, especially at low frequencies (below approximately 60 hertz).

Two commands let you disable and enable the beeper.

```
BEEP OFF
```

```
BEEP ON
```

To disable the beeper, type `beep off` [RTN]. Afterwards, the beeper will no longer signal the end-of-line, announce due appointments,* accompany error messages, or respond to `BEEP` statements. After a `BEEP OFF` command, the beeper remains nonfunctioning until you execute a `BEEP ON` command.

Keyword Abbreviations ([·])

Many of the words that the HP-75 recognizes (called *keywords*) can be abbreviated to save you typing time. For example, to sound a tone, you may type `be.` instead of the word `beep`. Here are the shortest allowable abbreviations of the keywords introduced so far:

Keyword	Shortest Distinct Abbreviation
AUTO	a.
BEEP	be.
COPY	co.
EDIT BASIC	e.ba.
EDIT TEXT	e.t.
PLIST	pl.
PURGE	pu.
RENAME	ren.
RUN	r.
STANDBY OFF	s.off
STANDBY ON	s.on

Note that all abbreviations end with periods and that you may include *more* characters in an abbreviation than the list above shows. For example, `pli.` is an acceptable form of `PLIST`. The following restrictions apply to abbreviations:

- You must type at *least* as many characters as appear in the shortest distinct abbreviation. For example, `p.` is not an acceptable form of `PLIST`.
- The period is not allowed to match the final character of the keyword. For example, `aut.` is not an acceptable form of `AUTO`.
- An abbreviated keyword may not contain embedded blanks.

An abbreviation can replace the keyword in any display line, although the examples in this manual spell out the complete forms of keywords for readability. Appendix D includes a complete list of HP-75 abbreviations.

What's Ahead

You've already learned many things about the HP-75 in this section. There's much more information ahead, but you don't have to read it all. We recommend you review this section and work through the examples of sections 2 and 3, but from then on, the choice is yours. You may choose to go directly to part III, Programming the HP-75, when you finish section 3.

You may want to check the appendices now to see what's there. For example, appendix G is a glossary that defines many of the terms used in this manual.

---

* There are 10 types of appointment alarms, 0 through 9. A type 6 alarm will always cause the alarm to sound, regardless of a `BEEP OFF` command.

Syntax Guidelines

Syntax is the way that instructions must be typed in order for the computer to understand their meaning. The following syntax guidelines are used throughout this manual.

DOT MATRIX TYPE	Words in dot matrix (like PLIST) may be entered in lowercase or uppercase letters. The examples show commands, statements, and functions entered in lowercase and converted internally to UPPERCASE.
italics type	Items in italics are the parameters you supply, such as the *frequency in hertz* for the BEEP statement.
' ', " "	Filenames and other character strings can be enclosed with single or double quotes and can be entered in lowercase or uppercase letters. The examples use single quotes. Quoted filenames are converted to uppercase internally.
[ ]	Square brackets enclose optional items.
...	An ellipsis indicates that the optional items within the brackets may be repeated.
stacked items	When two or more items are placed one above the other, one (and only one) of them may be used.
or	When two or more items are separated by "or", one or more instances of either or both items may be included.

The *HP-75 Reference Manual* lists the syntactical forms of all HP-75 instructions. Appendix H, Syntax Summary contains a detailed description of HP-75 instruction syntax and a complete set of syntax flow diagrams. Appendix H is a useful and concise reference to all of the commands and statements contained in your computer. You may wish to read this appendix to familiarize yourself with its contents before you go through the rest of the manual. As you study the manual, refer to appendix H whenever you have questions about HP-75 syntax. Appendix H will also be very useful as a ready reference when writing programs.

2

Section 2

Keyboard and Display Control

Contents

Introduction 34
The Keyboard 34
The Editing Keys 35
Moving Across the Display ([SHIFT] [←], [SHIFT] [→], [CTL] [←], [CTL] [→]) 35
Erasure Keys ([BACK], [SHIFT] [BACK], [DEL], [SHIFT] [DEL]) 35
The Insert/Replace Key ([I/R]) 36
Displaying Information 36
Display Echoing ([RTN], [→], [←]) 36
Recalling Your Last Entry ([CTL] [FET]) 38
The Delay Rate (DELAY) 39
Line Width (WIDTH, PWIDTH) 39
Setting the Right Margin (MARGIN) 40
The HP-75 Character Set 41
Characters and Decimal Codes (CHR$, NUM) 41
The Display Character Keystroke ([SHIFT] [I/R]) 42
Advanced User's Information 42
Control Characters ([CTL] [BACK], [CTL] [H], [CTL] [M], [CTL] [J]) 42
Carriage-Return/Line-Feed Keys 43

Introduction

This section shows how to use HP-75 editing keys and keystroke combinations to control the display line. The complete 256-character set is also discussed. You need to know how to add, delete, and insert characters in the display line to operate the computer effectively.

The Keyboard

There are 65 keys on the HP-75 keyboard, consisting of three groups:

- Typewriter keys. These are the 45 letter ([A]), digit ([9]), and symbol ([*]) keys, plus the space bar.
- System keys. These seven keys are [ATTN], [RTN], [TIME], [APPT], [EDIT], [RUN], and [FET]. You used them in section 1 and will continue to use them to control the HP-75 operating system.
- Editing keys. The 13 editing keys and a number of keystroke combinations offer you a large degree of control as you type information in the display. Examples from section 1 are [BACK], which backspaces the cursor, and [SHIFT] [LOCK], which locks the keyboard in uppercase.

The keyboard illustration on page 6 shows the location of the typewriter, system, and editing keys.

The Editing Keys

The editing keys generally work the same in all three modes, TIME, appt, and EDIT.* When used with the [SHIFT] and [CTL] keys, most editing keys perform a second and a third function. The editing keys, like the typewriter keys, repeat their functions when they're pressed and held.

Because of programming applications, this section shows the BASIC prompt (>) in the display, although the text prompt (:) can be there just as well. The HP-75 can store hundreds of program lines in memory, but the discussion focuses on a single display line, the current line, which is representative of all display lines in EDIT mode.

Moving Across the Display ([SHIFT][←], [SHIFT][→], [CTL][←], [CTL][→])

The left-arrow ([←]) and right-arrow ([→]) keys move the cursor across the display line without affecting anything else in the line. There are four ways to move the cursor to the left using the [←] key:

- Press [←] repeatedly.
- Press [←] and hold it down for continuous motion.
- Press [CTL][←], which moves the cursor 32 characters (one display window) to the left. If the cursor is less than 32 characters from the beginning of the line, [CTL][←] moves the cursor all the way to the left.
- Press [SHIFT][←], which moves the cursor all the way to the left.

The [→], [CTL][→], and [SHIFT][→] keystrokes work similarly, moving the cursor to the right.

Note that [TAB] and [SHIFT][TAB] are used in TIME and APPT modes to move the cursor forward and backward across the display. In EDIT mode, [TAB] has no effect and [SHIFT][TAB] has the same effect as [SHIFT][←].

Erasure Keys ([BACK], [SHIFT][BACK], [DEL], [SHIFT][DEL])

The [BACK] key backspaces the cursor, erasing characters one at a time until the cursor reaches the left edge of the display. [SHIFT][BACK] functions identically.

The [DEL] key lets you delete a character from the display without leaving a space in its place. If you hold down the [DEL] key, it continues its deletions. For example, type:

```
>This line will be deleted.
```

Now press [SHIFT][←] to left-shift the cursor to the beginning of the line.

```
>This line will be deleted.
```

When you press [DEL] the first time, the T will be deleted.

```
>his line will be deleted.
```

Now hold down [DEL] for continuous deletions.

```
>will be deleted.
```

The [DEL] key deletes one character at a time while left-shifting the characters to the right of the cursor.

* There are exceptions when appropriate. Example: In EDIT mode, the [CLR] key clears a display line; in APPT Mode, [CLR] displays an unfilled APPT template.

[SHIFT] [DEL] speeds up the deletions. The combination erases everything under and to the right of the cursor.

Example:

```
>will be deleted.
```

Press [→] to move the cursor to the right.

```
>will ▮
```

Pressing [SHIFT] [DEL] deletes to the end of the line.

The Insert/Replace Key ([I/R])

The HP-75 has two cursors to show you where the next character is going to appear as you type. The [I/R] (*Insert/Replace*) key lets you "toggle" between them:

- The replace cursor, the flashing ▮ symbol, is the normal cursor.
- The insert cursor, a flashing ◆ symbol, appears when you press the [I/R] key and disappears when you press [I/R] again.

The insert cursor enables you to add characters anywhere within an already typed line. For example, clear the display (press [CLR]) and type a command with a mistake:

```
>plst▮
```

The PLIST command is missing an i. Press [I/R] and use the [←] key to position the insert cursor.

```
>pl◆t
```

The ◆ points to where the next typed character will appear. Now type the i.

```
>pli◆t
```

The i inserted before the s.

Press [I/R] again to restore the ▮ cursor. Or press [RTN] instead to execute PLIST with the insert cursor still in the display.

The insert cursor may be used to edit the Note field in APPT mode. Pressing [I/R] has no effect in TIME mode.

You can use any editing key or keystroke to control the action of the insert cursor. For example, press [SHIFT] [DEL]:

```
>pli◆
```

Deletes all characters under and after the insert cursor.

Pressing [BACK] erases the character to the left of the insert cursor—the character that ◆ points to—while left-shifting the end of the line.

Displaying Information

Display Echoing ([RTN], [→], [←])

Numbers and characters can be "echoed" by the HP-75 as soon as you enter them. When you type a number and press [RTN] in EDIT mode, the HP-75 first tries to interpret that number as a line number (that is, as a number from 0 through 9999). If the number is greater than 9999, has a decimal point, or has a leading plus sign (+) or minus sign (−), then the HP-75 echoes the number.

Examples:

```
>123456.7
```

Entering a decimal.

```
123456.7
```

The number is echoed.

```
>- 3045
```

Entering a negative number.

```
-3045
```

```
>12
```

Entering an integer between 0 and 9999.

```
>
```

The entry is interpreted and entered as a BASIC line (although an empty line).

Expressions entered from the keyboard are evaluated and their results displayed.

Example:

```
>-6-12
```

A simple expression.

```
-18
```

The HP-75 will evaluate more than one expression at a time if you separate expressions with commas or semicolons. Commas cause results to be spread apart, while semicolons cause the results to be packed.

Examples:

```
>1.17,5.14
```

Separating the items with commas.

```
1.17                    5.14
```

```
>1.17;5.14;2+1
```

Separating the items with semicolons

```
1.17  5.14  3
```

Numbers are displayed on the HP-75 with one space for their sign (blank for positive, minus for negative) and one trailing blank. Note that if the results exceed one display window, only the last result or results will remain in the display.

Echoing also occurs for characters typed between quotation marks (' ' or " "). Pressing [RTN] causes the characters in the string to be displayed.

Example:

```
>'Weather forecast: sunny.'
```

Enclosing a character string with single quote marks.

```
Weather forecast: sunny.
```

Echoed without quote marks.

```
>'Did she say, "Come here"?'
```

Using two pairs of quotes. Note that the inner pair must be different from the outer pair.

```
Did she say, "Come here"?
```

The inner pair of quotes is preserved.

Numbers and quoted strings may be combined in mixed expressions.

Example:

```
>'Only';4;"workdays this week."
```

Two quoted strings and a number are separated by semicolons.

```
Only 4 workdays this week.
```

Note that the → and ← keys, alone and in combination with the modifiers SHIFT and CTL, may be used to scroll the characters that remain in the display. For example, press → three times:

```
y 4 workdays this week.
```

Pressing → and ← moves the displayed results back and forth.

Pressing any other key causes the prompt and cursor to reappear.

What display echoing is to keyboard results, DISP and PRINT statements are to program results. Refer to Displaying and Printing Information in section 11, page 166, for the use of DISP and PRINT statements to output program results.

Recalling Your Last Entry (CTL FET)

As you type, each character entered in the display line is also entered in a temporary *input buffer*, a 95-character location in memory that stores display information. You terminate entry in the input buffer by pressing the RTN key. In EDIT mode, holding down CTL while touching the FET key causes the current contents of the input buffer to reappear in the display. For example, if you press CTL FET now, you'll see:

```
Only';4;"workdays this week."
```

The last line you typed before pressing RTN.

The line is displayed left-justified, with the cursor positioned in the leftmost column. Although the prompt isn't displayed, the line is ready to be revised and reexecuted.

Using CTL FET means a savings in typing time, as you can recall long keyboard expressions without needing to retype them.

After you press RTN, new characters you type will replace old characters in the input buffer. Pressing CTL FET will display the contents of the buffer with those changes. For example, press CLR and type:

```
>'have
```

These new characters are entered in the input buffer. To verify, press CTL FET.

```
'have;4;"workdays this week."
```

The recalled line includes the last characters you've typed.

Note: Besides RTN, the ATTN, ↑, ↓, and RUN keys terminate character entries in the input buffer.

The following commands use the input buffer, causing your last entry to be lost: FETCH, FETCH KEY, LIST, LOCK, PLIST, and TRANSFORM.

The Delay Rate (DELAY)

The DELAY command regulates the display rate of error messages, card reader messages, HP-IL messages, and program output to the display and other display devices.

```
DELAY number of seconds
```

The *number of seconds* is any numeric expression that specifies the length of time between display lines, accurate to tenths of a second. When the battery pack is first installed, DELAY is set to 1 second; the MONEY program changes the DELAY to 0 seconds.

Parameters may range from 0 to 2^{26} seconds. Values outside this range default to the lowest or highest limits, respectively.

Example:

```
>delay 2
```

Pressing [RTN] sets the DELAY rate to 2 seconds.

Note that DELAY has no effect on single-line displays. For instance, echoed lines (page 36) remain displayed until a key is pressed or the HP-75 turns off.

You may override a delay condition by pressing any key. Each press causes the current line to be replaced immediately by the next line. For example, holding down [TAB] while listing the RAYMEMO file causes the lines to be displayed with no delay.

The current DELAY rate remains in effect until another DELAY command is executed.

Line Width (WIDTH, PWIDTH)

The WIDTH and PWIDTH commands enable you to specify the number of characters that will be displayed in a single display line or printed in a single print line.

```
WIDTH number of characters
```

```
PWIDTH number of characters
```

The WIDTH command sets the line width of display output. The PWIDTH command sets the line width of printer output. Initially, both WIDTH and PWIDTH settings are for 32 characters, the width of the display window. These commands don't affect the behavior of the display window during *input* operations.

You can include any numeric expression in a WIDTH or PWIDTH command to specify any line width of one or more columns. The appropriate setting depends on the output device. For example, a logical PWIDTH setting for the HP 82162A Thermal Printer is 24 or some multiple of 24, based on the 24-character print line of the printer.

The HP-75 sends as many characters to the display or printer device as the current WIDTH or PWIDTH setting specifies and then sends a carriage return/line feed to the device to begin a new line.

Example: Display the date and time in EDIT mode so that the information appears in the display window three characters at a time. Set the display line width by typing `width 3` [RTN]. Then use the `DATE$` and `TIME$` functions (page 98). Type:

```
>date$;time$
```

Pressing [RTN] causes the date and time to be displayed according to current `WIDTH` and `DELAY` settings.

The example shows that a setting less than 32 "breaks" displayed lines into smaller portions. A setting greater than 32 causes long lines to scroll across the display. A setting greater than 96 causes characters added after the 96th to replace themselves in the last character position.

`WIDTH` and `PWIDTH` parameters are rounded to integer values; values of `0` or `1` specify a width of 1 column. (Negative values and values greater than 255 result in widths of 255 characters, limited by the line length of the display or printer device.) `WIDTH INF` and `PWIDTH INF` (for "infinity") permit an unlimited number of characters to be sent to one line—the output never overflows to another line.

A `WIDTH` or `PWIDTH` setting remains in effect until you specify another setting. You should set the HP-75 back to its initial display width now by typing `width 32` [RTN]. `DELAY` and `WIDTH` commands are useful in programs to specify initial display conditions.

Setting the Right Margin (`MARGIN`)

After a system reset, the HP-75 is set to beep when you type the 91st character in the display line. The `MARGIN` command enables you to specify a new character position at which the beep will signal the end of line.

```
MARGIN number of characters
```

Examples:

```
>margin 26
```

Causes the beeper to sound when the 26th character is typed.

```
>margin 75
```

Causes the beeper to sound when the 75th character is typed.

Note that the beeper serves as a reminder of—not a restriction on—the line length. It actually responds to the position of the cursor—for the insert cursor, the beep doesn't correspond to the line length. Regardless of the `MARGIN` setting, you can still type 94 characters in a given line, that is, 96 column positions minus one position for the prompt and one position for the cursor.

The number of characters may be specified by any numeric expression (such as `90-15`); expressions are rounded to integer values.

The `MARGIN` setting remains in effect until another `MARGIN` command is executed.

Programming Note: The `MARGIN` command sets the number of characters that may be typed in response to an `INPUT` statement before the HP-75 beeps.

The HP-75 Character Set

The 256 HP-75 characters have numbers, or *decimal codes*, associated with them, ranging from 0 through 255. Appendix D includes a table of characters and associated decimal codes. Ninety-five characters (decimal codes 32 through 126) are standard printable characters as defined by the American Standard Code for Information Interchange (ASCII). Note, then, that characters 0 through 31 and characters 127 through 255 may be interpreted by an HP-IL device or another computer differently from the way the HP-75 displays them. You should refer to your owner's manual for the peripheral or computer to determine its character set.

Characters and Decimal Codes (CHR$, NUM)

Given a decimal code, the CHR$ function returns the corresponding display character.

Example:

```
>chr$(65)
```

```
A
```

Uppercase letters correspond to decimal codes 65 through 90.

CHR$ rounds numeric expressions to integer values and converts the integers to the proper range (modulo 256).

The NUM function works opposite to CHR$. Given a quoted character, NUM returns its decimal code.

Example:

```
>num('A')
```

```
65
```

The character must be enclosed with quotes.

The decimal code of A.

NUM may operate on any string expression, returning the decimal code of the *first* character in the string.

Example:

```
>num(' ABC')
```

```
32
```

Returns the decimal code of the first character in the string, a space.

Characters whose decimal codes are 128 through 255 appear underlined in the display. To display an underlined character, use the decimal code of the non-underlined character, the constant 128, and the CHR$ function. For example, to display A:

```
>chr$(65+128)
```

```
A
```

Using 65 as the value of A.

Returns the underlined character.

An alternate method includes the NUM function:

```
>chr$(num('A')+128)
```

```
A
```

NUM returns the decimal code of A.

The same result.

The Display Character Keystroke (SHIFT I/R)

One hundred ninety-four keys and keystroke combinations can produce display characters. Editing and system keys as well as typewriter keys have associated display characters. However, you can't display many of these characters directly by pressing the corresponding keys—many keys perform predefined operations when they're pressed. But SHIFT I/R, the *display character keystroke*, enables you to access these characters.

For example, the APPT key is associated with the ⊠ character. However, if you press APPT by itself, the HP-75 switches to APPT mode. To display the ⊠ character, you must first use SHIFT I/R. Pressed once, SHIFT I/R leaves the cursor in its present position and prepares the HP-75 to display the character of the very next key you press. Press SHIFT I/R once and then press the APPT key: The ⊠ character appears.

Pressing and releasing SHIFT I/R always causes the next key or keystroke combination to display its character. Try a keystroke combination: Press CTL 1 by itself. Nothing happens. Now press SHIFT I/R and then CTL 1—an underlined i is displayed.

If you're using the numeric keypad (CTL LOCK), then SHIFT I/R enables the keys to perform their normal display functions. For example, the keypad U outputs a 4. If you press SHIFT I/R and then the unshifted U, you display lowercase u.

SHIFT I/R also overrides key definitions that you've declared earlier. If the unshifted Q key is redefined to display your name, then SHIFT I/R followed by Q displays the regular q.

If you press SHIFT I/R twice in succession, what happens? The character associated with SHIFT I/R is itself displayed, ⊂.

It follows that all HP-75 keys and keystroke combinations have decimal code equivalents. For example, the APPT key corresponds to decimal code 130. SHIFT I/R and the NUM function enable you to determine the decimal code of a particular key or keystroke combination. For the decimal code of APPT, type:

```
>num('⊠')
```

You must press SHIFT I/R first to type the APPT display character.

```
 130
```

The APPT key corresponds to decimal code 130.

The character set table in appendix D includes the decimal codes of the keyboard and indicates which keystrokes must be preceded by SHIFT I/R to display characters.

Advanced User's Information

Control Characters (CTL BACK , CTL H , CTL M , CTL J)

There are 33 ASCII-defined *control characters*, used to control the interchange of information among communications systems. Control characters correspond to decimal codes 0 through 31, and 127. Control characters may be produced either by the CTL key or by the CHR$ function. For example, both CTL SPACE BAR and CHR$(0) generate the "null" control character, decimal code 0, displayed by the HP-75 as △.

A number of control characters have special meaning for the HP-75 display and for HP-IL devices. One in particular, the *escape* character, has a variety of control applications. The escape character is generated either by the CTL BACK keystroke—as indicated by the letters ESC over the BACK key—or by CHR$(27). The HP-75 doesn't display an escape character—instead, it shows > and turns off the cursor. The cursor

reappears when you type the next character. The responses of the display window and HP-IL devices to escape character codes are discussed further in section 9.

Here are three other special control characters:

- The *backspace* character (decimal code 8) causes the cursor to backspace. The backspace may be "displayed" by typing a few characters and then pressing [SHIFT] [I/R], followed by [CTL] [H], [BACK], or [SHIFT] [BACK]. (The three keystrokes are identical.) Note that the backspace character moves the cursor without causing an erasure.
- The *carriage-return* character (decimal code 13) causes the cursor to return to the left edge of the display. A carriage-return may be "displayed" by pressing [SHIFT] [I/R] and then [CTL] [M] or [RTN]. Note that the carriage-return character moves the cursor without causing the contents of the line to be executed. (Refer to the following topic, Carriage-Return/Line-Feed Keys.)
- The *line-feed* character (decimal code 10), causes the HP-75 to advance to a new display line. A line-feed may be "displayed" by pressing [CTL] [J].

The following LINEFEED program demonstrates the effect of the backspace, carriage-return, line-feed, and escape characters on the HP-75 display. Prepare for the program by typing `edit 'linefeed', basic` [RTN]:

```
LINEFEED    B     0 11:50 02/05/83
```

The heading of the program. Press [ATTN] and enter the program.

```
1 input a
2 disp'Beginning';chr$(a);'End'
```

The LINEFEED program consists of two lines. Press [RTN] after each.

Run the program four times. Each time the ? prompt appears, enter a new decimal code—8, 10, 13, or 27—and press [RTN].

Examples:

```
?8
                                PRGM
```

Specifies the backspace character.

```
BeginninEnd
```

The second string is displayed beginning from the location of the backspaced cursor.

```
?27
```

```
nd
```

The escape character plus capital E causes the HP-75 to clear the display completely.

The CHR$ function is often used in programs to send control character information to HP-IL devices.

Carriage-Return/Line-Feed Keys

Nine HP-75 keys cause both a carriage-return character and a line-feed character to be sent to the HP-75 display and other HP-IL display devices. These are the seven system keys ([ATTN], [RTN], [TIME], [APPT], [EDIT], [FET], [RUN]) and two editing keys ([↑], [↓]). (Note that pressing [CTL] [M] is the same as pressing [RTN].)

When you press one of these keys, a carriage-return/line-feed occurs before the normal function of the key is carried out. For example, before the [APPT] key causes the HP-75 to display the APPT template, it first returns the cursor to left edge of the display and causes the HP-75 to display a new line. In most cases, carriage-return/line-feeds occur so quickly that they're noticeable only on external display devices.

Section 3

3 File Editing

Contents

Introduction 44
Filenames 45
Files in Memory 45
EDIT Mode (EDIT) 46
Available Memory (MEM) 47
Creating and Editing a File (CAT, EDIT, FETCH) 47
Checking the System Catalog (CAT ALL, EDIT) 49
Purging Files (PURGE) 50
Entering Lines in a Text File (RTN) 50
- Automatic Line Numbering (AUTO) 51
- Syntax Errors 52

Editing Operations 53
- Stepping Through a File (↑, ↓) 53
- Fetching Lines from a File (FETCH) 53
- Revising Lines 54
- Inserting Lines 55
- Moving Lines 55
- Listing Lines (LIST, PLIST) 56
- Renumbering Lines (RENUMBER) 57
- Deleting Lines (DELETE) 59

File Manipulations 59
- Renaming Files (RENAME) 59
- Merging Files (MERGE) 60
- Copying Files to Cards (COPY) 60
- Duplicating Files (COPY) 61
- Locating and Creating Files (EDIT) 62
- The workfile (EDIT, NAME) 63

Transforming Files (TRANSFORM) 64
Command Summary 65

Introduction

The HP-75 has a versatile file handling system that enables you to store and access multiple programs, text memos, appointment calendars, keyboard redefinitions, and other blocks of information. A *file* is an area of memory that can be identified by name and manipulated as a unit. Files are collections of lines of information that have either been entered from the keyboard or else copied from a mass storage medium. Here are four examples of display lines from section 1 that are stored in files in memory.

```
SAT 02/05/83 09:25 AM #3N !Phone
```

Stored in an appointment file named appt.

```
>1■BEEP RND*999,RND*.5 @ GOTO 1
```

Stored in a BASIC program file named SINGSONG.

```
>def key 'q','Your Name';
```

Stored in a file that contains all key redefinitions named keys.

```
:10 TO Ray
```

Stored in a text file named RAYMEMO.

In each case, you entered the line in the file by pressing the [RTN] key. The [RTN] key has two important uses in this section:

- To enter new lines in files.
- To execute commands that control files, like RENAME.

Filenames

Each file in memory has a unique name, called its *filename*. For example, there can be only one file named MONEY. A filename may be composed of one to eight characters.* The first character of the filename must be a letter or period, and the remaining characters can be any combination of letters and digits. Examples of acceptable filenames are C, ACC1, .TEMP, REV3, .92, and LOWSCORE. You can use lowercase letters to enter filenames, but they are converted to uppercase by the HP-75. A blank character terminates the filename—subsequent characters are ignored. Use quotation marks, single or double, to delimit filenames when you key in commands. (There are two exceptions—the names of the special appt and keys files are stored as lowercase letters, and you must enter these filenames without quotation marks.)

A filename that begins with a period specifies a *volatile file*. If the HP-75 turns itself off or you execute a command that turns it off, all such volatile files will be purged when it is next turned on. Essentially, volatile files are temporary files that are intended for your current work session only.

Files in Memory

Files are stored in memory in the order of their creation, with the earliest file located at the "bottom." Here are the relative locations of the files created in sections 1 and 2. The diagram represents the HP-75's memory and includes names, types, sizes (in bytes), times, and dates of the files stored there.

Available memory.

Refer to appendix D for information regarding the memory required by files.

```
LINEFEED    B    53 11:50 02/05/83
```
The most recent file.

This BASIC file was created if you entered the program on page 43.

```
RAYMEMO     T    89 10:16 02/05/83
```
A text file.

```
MONEY       B 2103 00:00 01/04/82
```
Although the display shows the date of the prerecorded program, the file is located according to the time when you copied to memory.

← The keys file was located here until you erased it by typing purge keys [RTN].

```
SINGSONG    B    40 09:45 02/05/83
```
The first program you created.

```
appt        A    27 09:30 02/05/83
```
Your earliest file, created when you first scheduled an appointment.

* In its most general form, a filename can be any string expression. (The topic of string expressions is described later, in Section 13.)

The HP-75 can hold as many files as memory permits, of six types:

- BASIC or program files, indicated by the letter B. BASIC files are numbered lines of program instructions. A *private* BASIC file will be indicated by PB, meaning that the program may be run but not listed, changed, or copied (section 8, page 117).
- Text files, indicated by T. Text files are numbered lines of arbitrary sequences of characters.
- Appointment files, indicated by A (section 7, page 102).
- Keys files, also indicated by T. Keys files are special text files (section 10, page 144).
- *Language Extension Files*, or LEX files, indicated by L. LEX files are special program files available through prerecorded magnetic cards and tapes and through plug-in ROMs (appendix B, page 277).
- *Logical Interchange Files*, or LIF1 files, indicated by I. LIF1 files are specially formatted files for interchanging information between the HP-75 and other computers (appendix B, page 274).

This section uses *text* file examples for simplicity, but the process of composing and managing BASIC files is the same.

EDIT Mode (EDIT)

You create and update text and BASIC files in EDIT mode. Press EDIT key to enter EDIT mode:

```
>
```

The BASIC prompt and the cursor appear.

If the text prompt (:) appears instead, type `purge` RTN to erase the current text workspace, and then type `edit basic` RTN to prepare the HP-75 to edit a BASIC file. The BASIC prompt will appear when you begin typing or if you press ATTN.

The BASIC prompt indicates the readiness of the HP-75 to accept program statements. The text prompt indicates the computer's readiness to accept lines of text. Here are the different EDIT mode operations:

With the BASIC prompt (>) in the display:

Compose and revise BASIC files.

```
100 DISP A;B;C
```

With either prompt:
Evaluate expressions.

```
sqr (2)
```

Execute system commands.

```
delay 1
run 'money'
```

With the text prompt (:) in the display:

Compose and revise text files.

```
10 DearJohn,
```

You'll notice that you can execute system commands (like DELAY and RUN) regardless of the prompt in the display. The difference is that BASIC lines are translated into machine code as they are entered to speed program execution, while text lines are stored as entered. Therefore, text files can't be executed as programs, even if the text contains legal BASIC statements. (However, the TRANSFORM command, discussed in appendix B, on page 274, enables you to convert a file from one type to another.)

Available Memory (MEM)

The MEM function enables you to check the current size of available memory.

Example:

```
>mem
```

```
 4912
```

Typical result, indicating that the HP-75 can store an additional 4912 bytes of information.

The number of files in memory and the number of lines in each file are limited only by available memory. As an example, a 30-line memo (about a page of double-spaced typing) requires approximately 2500 bytes of memory, the equivalent of two magnetic cards of information.

The number of bytes returned by MEM may vary, depending on when the function is used.

Example:

```
>mem;mem;mem
```

```
4912  4905  4898
```

Each execution of the function temporarily requires seven bytes of memory.

MEM can be used to determine the memory required for any HP-75 operation. Execute MEM before and after a program or keyboard operation to determine its effect on available memory.

Note that the operating system requires approximately 2100 bytes of memory for system routines. After a system reset, memory available to the user is about 14,000 bytes. Installing the HP 82700A Memory Module adds 8K (8192) bytes of memory to the machine.

Creating and Editing a File (CAT, EDIT, FETCH)

Whenever the HP-75 is in EDIT mode, the computer accesses a BASIC or text file called the *current file*. While you are working on a file, (editing or running it, for example) it is the current file. Several HP-75 commands can change which file is the current file. At the moment, the HP-75 is "in" a BASIC file, as indicated by the > prompt. To learn more about the current file, execute the CAT (*catalog*) command:

```
>cat
```

```
LINEFEED   B   58 11:50 02/05/83
```

Displays the catalog entry of the current file.

A catalog entry contains five pieces of information:

- The name of the file you're currently editing (in this case, LINEFEED).
- The type of file (B for BASIC).
- The amount of memory the file occupies, in bytes. As a reference, each display character in a text file takes one byte of memory. A 0 indicates the file is empty.
- The time the file was created, expressed in 24-hour notation. For example, 13:00 represents 1:00 p.m.
- The date the file was created.

Press [→] to scroll the line to the left; press [←] to scroll the line to the right. Press [SHIFT] or [CTL] to modify the action of [→] and [←]. Any other key will restore the prompt and cursor:

```
>
```

The catalog entry disappears.

To create a new text or BASIC file, use the EDIT command. Four forms of the EDIT command follow:

EDIT '*filename*', BASIC	Creates a BASIC file by the specified name.
EDIT '*filename*', TEXT	Creates a text file by the specified name.
EDIT BASIC	Creates a temporary BASIC file named workfile.
EDIT TEXT	Creates a temporary text file named workfile.

The following pages show how an example text file is created, written, revised, and manipulated. Type the following and press [RTN]:

```
>edit 'example',text
```

Using the EDIT command to create a text file. Either single or double quotes may enclose the filename.

```
EXAMPLE     T     0 13:07 02/05/83
```

Displays the catalog entry of the file.

Press [ATTN] to cause the text prompt (:) to appear, indicating that you're editing a text file. Now type this line of characters, which begins with a line number, and press [RTN]:

```
:0This is line zero.
```

Typing a line for the EXAMPLE file.

```
:
```

Line 0 is entered in memory.

Although line 0 no longer appears in the display, it's still the current line of the file. The HP-75 uses an internal mechanism, the *file pointer*, to keep track of the current line of the current text or BASIC file. To fetch (or recall) the current line to the display, you press two keys in sequence [FET] and [RTN]. Press [FET] to display the word FETCH:

```
:FETCH
```

The [FET] key is a typing aid. You could instead type the word fetch.

Then press [RTN] to execute the FETCH command:

```
0This is line zero.
```

The current line appears just the way you typed it.

The current line—the line pointed to by the file pointer—is also called the *pending line. Whenever you wish to bring the pending line to the display, press* [FET] *and then* [RTN]. The pending line remains fixed unless you move the file pointer. For example, press [APPT] to leave EDIT mode temporarily:

```
Day Mo/Dy/Yr Hr:Mn AM 1N !Note
```

Pressing [APPT] switches the HP-75 to APPT mode. Now press [EDIT] again.

```
:
```

Back in EDIT mode.

Use the CAT command to confirm that the current file is EXAMPLE.

```
:cat
```

```
EXAMPLE     T    36 13:07 02/05/83
```

The file pointer is still located in the EXAMPLE file. The file has grown to 36 bytes.

FETCHing the current line confirms that it's line 0.

```
:FETCH
```

```
0This is line zero.
```

The pending line of the EXAMPLE file.

Even when the HP-75 turns off, the file pointer stays fixed in the current text or BASIC file at the pending line. To move the file pointer to another file (for example, to return to the SINGSONG file), you must execute another EDIT command. You'll be using the CAT, FETCH, and EDIT commands throughout this section.

Checking the System Catalog (CAT ALL, [EDIT])

Typing cat [RTN] displays the catalog entry of the current text or BASIC file. The CAT command has four other forms:

CAT ALL	Accesses the complete system catalog.
CAT '*filename*'	Displays the catalog entry of the specified file.
CAT APPT	Displays the catalog entry of the appt file.
CAT KEYS	Displays the catalog entry of the keys file.

Note that no quotes are used for the appt and keys files.

Executing CAT ALL lets you review the catalog of all files in memory. Press [CLR] to clear the line and type:

```
>catall
```

```
  Name    Type Len   Time    Date
```

The first line of the CAT ALL listing is displayed according to the current DELAY rate.

The first line labels the five fields of the catalog entries. After a delay equal to the current DELAY rate, the catalog entry of the most recently created file is displayed:

```
EXAMPLE     T    36 13:07 02/05/83
```

EXAMPLE, the current file, happens to be the most recently created file.

To display the catalog entries of the other files in memory, use [↓], the down-arrow key. The catalog entries appear according to when the files were created. Press [↓] repeatedly until you reach the oldest file, the appt file:

```
appt        A    27 09:30 02/05/83
```

The oldest file in memory appears last in a CAT ALL listing.

The up-arrow key ([↑]) lets you review the catalog entries of more recent files in memory. [SHIFT] [↓] displays the catalog entry of the oldest file; [SHIFT] [↑], the catalog entry of the most recent file.

The [EDIT] key assumes a new function during a CAT ALL listing. Pressing [EDIT] causes the file whose catalog entry is in the display to become the current file. (However, if an unnamed, nonempty workfile exists, you can't access another file in this way; refer to page 63.)

Example:

```
MONEY        B 2205 15:23 04/22/82
```

The catalog entry of the MONEY program during CAT ALL. Press [EDIT].

```
>
```

The file pointer is moved to the first line of the file, and the BASIC prompt and the cursor appear.

Note the file must be a BASIC or text file available for editing; otherwise, pressing [EDIT] causes warning 65—access restricted— to occur and the catalog entry of that file to reappear in the display.

To return to the EXAMPLE text file, either type edit 'example' [RTN] or execute another CAT ALL command and press [EDIT] with the EXAMPLE catalog entry in the display.

Purging Files (PURGE)

The PURGE command erases one file at a time from memory.

PURGE	Erases the current file.
PURGE '*filename*'	Purges the specified file.
PURGE APPT	Purges the appt file.
PURGE KEYS	Purges the keys file.

For example, if you typed purge [RTN] right now, you would purge the EXAMPLE file.

After a PURGE command, the memory formerly occupied by the file is reclaimed. The PURGE command can be considered "dangerous": Once executed, the command can't be undone. If you purge the current file, the HP-75 creates a temporary workfile of the same type and makes it the current file.

Entering Lines in a Text File ([RTN])

Each line of a text file starts with a line number, that is, an integer from 0 through 9999. Each line has 96 character positions, including one position for the prompt, one to four positions for the line number, and the final position for the cursor. This means that each line of a text file can consist of up to 90-93 characters. After typing a line, press [RTN]: The HP-75 stores the contents of the display line in the current file.

For example, type:

```
:100 To put a line of characters in your text file, just start typing.
```

The line begins to scroll across the display window at the 32nd character (the s in characters).

The display window at the end of your typing.

Use a combination of the [←], [→], and modifier keys to review what you've typed. Then press [RTN]:

```
:
```

Pressing [RTN] stores the complete line as part of the current text file.

To display the pending line—the line you've just entered—fetch it from memory:

```
:FETCH
```

```
:100To put a line of characters
```

The pending line. The cursor is conveniently positioned just after the line number.

To clear the display, press [CLR], [ATTN], or [EDIT]. This has no effect on what's been stored in the file.

Lines can be entered in any order, but they're stored in the order of their line numbers, from lowest to highest. You can enter lines in only one file at a time—the current file. *You must press [RTN] in order to enter a line in a file.*

When you press [RTN] while editing a *text* file, the HP-75 ignores all leading blanks and checks for a valid line number. If it finds a line number, the HP-75 stores the line *as is* in the current text file. For example, `10+3` is stored as line 10, the contents of which are `+3`. Uppercase and lowercase letters, spaces after the line number, quotation marks, control characters—in short, everything—is preserved the way you've typed it.

To do keyboard arithmetic while editing a text file, begin the line with any character except integers between 0 and 9999, for example, a decimal point (.), a plus sign (+), a minus sign (–), a left parenthesis (`(`),or a simple numeric variable name (refer to Assigning Values to Variables in section 5). For example, clear the display and type:

```
:+10+3
```

Adding two numbers with leading plus sign.

```
 13
```

The result. The text file is unaffected by the calculation.

Automatic Line Numbering (AUTO)

Executing the AUTO command causes the line numbers of a text or BASIC file to be provided automatically.

```
AUTO [beginning line number [, increment value]]
```

Clear the display and type:

```
:auto 1,1
```

```
:1
```

Starts the numbering at line 1. A space is added after the line number.

Now enter four more lines in the EXAMPLE file. Type:

```
:1 She walks in beauty,
```

Any combination of display characters may follow the line number.

```
:2
```

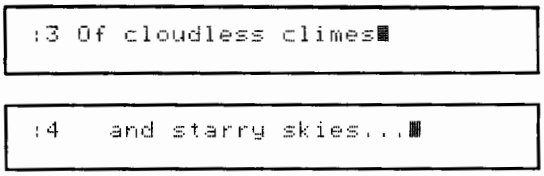
Line numbers increment in steps of 1, as specified in the AUTO command.

```
:2   like the night
```

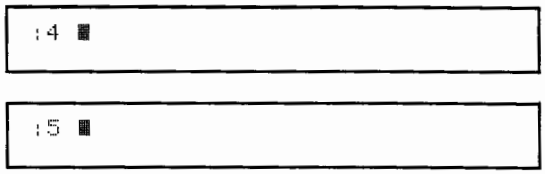
Indenting two additional spaces for the verse fragment.

```
:3
```

The file pointer advances to the new line as it's numbered.



The last line of the fragment.

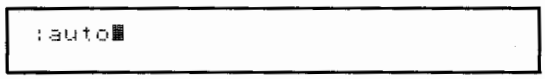


Press [ATTN] here to stop the numbering.

The file pointer is positioned at the last line containing text. That is, if you execute FETCH now, line 4 will be recalled to the display, although the last line number displayed was line 5.

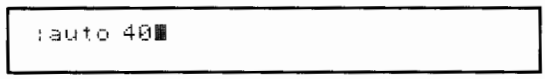
Executing AUTO with no parameters causes numbering to begin at the current line number *plus 10* and to increment in steps of 10.

Example:

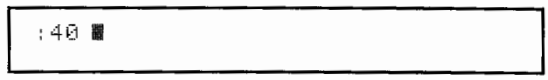


Starts the numbering at 14, which is the current line number (4) plus 10.

If you specify just the beginning line number, then the line numbers start at that number and increments in steps of 10. For example, press [ATTN] and then type:



Specifies the beginning line number.



The first line displayed is 40, the next will be 50, and so on. Press [RTN] once to verify.

Press [ATTN] to stop the process.

Note that auto-numbering won't cause existing lines in the file to be overwritten. For example, because line 4 exists in the EXAMPLE file, typing auto 4 causes line 4 to be recalled to the display rather than replaced. In other words, the AUTO command *fetches* lines from the current text or BASIC file.

Syntax Errors

While editing a text file, you can execute any system command (like FETCH and CAT) directly from the keyboard. You can also execute BASIC statements (like BEEP) and expressions (like SQR(2)). However, you should include line numbers for all lines containing straight text. If you don't, and then press [RTN], a *syntax*, or language-related, error may occur.

The HP-75 responds to syntax errors by:

- Beeping (assuming a BEEP ON setting).
- Turning on the **ERROR** annunciator.
- Displaying an error message according to the current DELAY rate.
- Recalling the incorrect line to the display.

For example, press [ATTN] and then type:

The author's name is missing a line number.

A syntax error results.

Error recovery is simple:

- To recall the error message, press [SHIFT] [FET]. In this case, `missing parameter` means that the display line was unsuccessfully evaluated as an expression.
- To display the error identification number, use the ERRN function (type `errn` [RTN]).
- To clear the error condition, the **ERROR** annunciator, and display line, press [CLR], [ATTN], or [EDIT].
- To correct the error, use the editing keys. In this instance, press [I/R], press [SHIFT] [←], insert a line number and a space, and then press [RTN]:

```
:34 --Lord Byron--
      ERROR
```

Inserting a line number and space with the insert cursor.

```
:
```

The corrected line is stored in the file as line 34.

Remember that [CTL] [FET] recalls the contents of the input buffer. This means that when you're adding lines to a file, pressing [CTL] [FET] displays the most recently entered line.

Example:

```
34 --Lord Byron--
```

[CTL] [FET] recalls the characters as they appeared when you pressed [RTN].

Editing Operations

Whenever the HP-75 is in EDIT mode, you can edit the current file—that is, you can add new lines to that file, examine lines, change lines, insert lines, move lines, list lines, renumber lines, and delete lines.

The editing keys and keystroke combinations covered in section 2 help you revise *individual* lines. These keys allow you to arrange the display line they way you want before entering it in the file (that is, before pressing [RTN]).

Stepping Through a File ([↑], [↓])

The [↑] and [↓] keys move the file pointer up and down through the current file. Pressing [↑] causes the line preceding the pending line to appear, ready for editing. Likewise, pressing [↓] enables you to view and revise the line following the pending line. Note that pressing [↑] or [↓] *doesn't* enter the displayed line in the file. If you've changed a line and want that change kept in the file, press [RTN] before pressing [↑] or [↓].

Holding down [↑] or [↓] causes the key to repeat its function. [SHIFT] [↑] moves the file pointer to the first line of the file, and [SHIFT] [↓] moves the file pointer to the last line of the file.

Fetching Lines from a File (FETCH)

One of the uses of the FETCH command is to display the pending line of the current file. If supplied with a quoted string, a line number, or both, FETCH can move the file pointer to any line in the file. Here are the four forms of the FETCH command:

FETCH	Fetches the pending line.
FETCH *line number*	Fetches the specified line.
FETCH '*search string*'	Fetches the next line after the pending line that contains the *search string*.
FETCH '*search string*', *line number*	Beginning at the specified line fetches the first line that contains the *search string*.

The [FET] key is a typing aid—press [FET] to place the word FETCH in the display. Adding a quoted string, for example, causes a search to occur from *the line after the pending line* to the end of the file.

Example: Find the line in the EXAMPLE file that contains the characters cloud. Begin by moving the file pointer to the first line of the file (press [SHIFT] [↑]). Then clear the display (press [CLR]) and type:

```
:FETCH 'cloud'
```

The command with a quoted phrase. The desired line must appear *after* the pending line.

```
:3 Of cloudless climes
```

First line after the pending line to contain the phrase. The cursor is positioned at the c.

The characters in the FETCH string must be typed with the *same spacing and case* as those in the desired line. If the line is found, the cursor will be positioned at the beginning character of the specified string.

You need supply only a short phrase in the FETCH command (like 'loud' or even 'd'). Either single quotes or double quotes can set off the phrase, so long as they're paired, ' ' or " ". If the search string doesn't appear anywhere in the file after the pending line, then the FETCH command leaves the file pointer at the current line.

FETCH *line number* fetches the specified line:

```
:FETCH 100
```

The command with a line number.

```
100To put a line of characters
```

Line 100 is fetched, ready for editing.

If there is no such line, then the HP-75 displays the specified line number, ready for you to type the new line.

Example:

```
:FETCH 102
```

Fetching a nonexistent line.

```
:102
```

The specified line number is displayed.

Note that the new line is not entered in the file until you type at least one character and press [RTN]. Otherwise, the new line disappears as soon as the file pointer moves to a different line.

By specifying both a search string *and* a line number, you cause the HP-75 to start at the specified line and search for the first line with the specified string.

Example:

```
:FETCH 'beauty',0
```

Separate the two parameters with a comma.

```
:1 She walks in beauty,
```

Beginning at line 0, the HP-75 searches for the first line containing the string.

Revising Lines

To revise an existing line in a text file:

1. Recall the line to the display using FETCH, [↑], or [↓].
2. Change the display line to the way you want it.
3. Press [RTN]. The edited line permanently replaces the original in the file.

For example, press [SHIFT] [↑]:

```
:0This is line zero.
```

Pressing [SHIFT] [↑] moves the file pointer to the first line of the file.

Now add a space between the line number and the first character (T): Press [I/R], then the space bar:

```
:0 This is line zero.
```

Editing the current line. Pressing [RTN] stores the new version in the file.

If you replace the line *number* with a different line number and then press [RTN], the new line is entered in the file. However, the original line—with the old line number—stays in the file also. In effect, the old line is duplicated.

Inserting Lines

To insert a new line in a text file:

1. Type the new line with appropriate line number.
2. Press [RTN] to put the new line in the file.

Remember that line numbers must be integers. It's not possible to insert a new line between two consecutively numbered lines, say, between lines 2 and 3. The RENUMBER command (page 57) allows you to change the intervals between line numbers.

Moving Lines

To move a line to a different place in the current file:

1. Move the file pointer to the desired line using FETCH, [↑], or [↓].
2. Change its line number.
3. Press [RTN] to put the new line in memory.
4. Delete the original line by entering a blank line with the same number.

For example, to move Byron's name to the head of the verse, follow these steps:

```
:Fetch 'B',0
```

To find the desired line, beginning the search from the top of the file.

```
:34 --Lord Byron--
```

Line 34 is now the pending line.

```
: 0--Lord Byron--
```

To change the line number, use [SHIFT] [←], a space, and a zero.

```
:
```

Stores this line in the file, replacing the original line 0.

Finally, delete the old line (34). Type its line number and press [RTN]:

```
:34
```

The old line number.

```
:
```

Deletes line 34.

This procedure enables you to relocate lines anywhere in the current EDIT file. You can also use the DELETE command (page 59) to remove unwanted lines.

When you insert a line or when you delete a line by typing the line number and pressing [RTN], the file pointer is set to that line. For example, when you typed 34 [RTN], line 34 became the pending line.

Listing Lines (LIST, PLIST)

The FETCH command and the [↑] and [↓] keys recall single lines from the current file and make them available for editing. The LIST and PLIST commands also display file lines. In addition, LIST and PLIST can access any text or BASIC file in memory and can display any or all of its lines. However, lines are listed for you to examine but not to edit.

```
LIST [beginning line number [,final line number]]
LIST 'filename' [,beginning line number [,final line number]]
```

```
PLIST [beginning line number [,final line number]]
PLIST 'filename' [,beginning line number [,final line number]]
```

To list all the lines of the current file, execute LIST alone. To stop a listing, press [ATTN]. Type list [RTN]:

```
0 --Lord Byron--
1 She walks in beauty,
2   like the night
3 Of cloudless climes
4   and starry skies...
40
100 To put a line of characters
in your text file, just start ty
ping.
```

Lines are listed in the order of their line numbers at the current DELAY rate.

Line 40 consists of a line number and one space. Line 100 appears 32 characters at a time as specified by the WIDTH setting.

Notes on LIST:

- During a listing, the file pointer remains fixed at the current line.
- A WIDTH setting greater than 32 causes long lines to scroll across the display as they're listed.
- You may override the current DELAY by pressing any key except [ATTN], [SHIFT], or [CTL]. The listing will immediately continue at the next line.

After a listing of EXAMPLE, the display shows:

```
ping.
```

Notice that neither the prompt nor the cursor appears in the display.

Listed lines are not available for editing, although the [→] and [←] keys are active. When you press another key, the listed line disappears and the prompt and cursor reappear. The file pointer remains positioned at the pending line from before the listing. To check this, press [FET] and then [RTN]:

```
:FETCH
```

To fetch the pending line.

```
34
```

Line 34 is the pending line from before. It's empty because you've deleted it.

LIST *line number* displays the one line specified in the command. For example, clear the display and type:

```
:list 4
```

```
4   and starry skies...
```

The specified line is displayed without prompt and cursor.

If a nonexistent line is specified, the LIST command is ignored.

LIST *beginning line number , final line number* lists the lines bracketed by and including the specified line numbers.

It's also possible to list lines from another text or BASIC file. The following examples all cause one or more lines from the RAYMEMO text file of section 1 to be listed:

```
:list 'raymemo'
```

Lists the entire file.

```
:list 'raymemo', 20
```

Lists line 20 of the file.

```
:list 'raymemo',10,30
```

Lists lines 10 through 30 inclusive.

The PLIST (*print-list*) command works similarly to the LIST command. There are four differences between PLIST and LIST:

- PLIST causes the lines of the specified file to be output to current PRINTER IS devices. If no printer devices are assigned, then the file is listed on the HP-75 display and DISPLAY IS devices (page 128).
- The DELAY setting doesn't affect the print-listing rate of external printers. Print-listings on external printers occur with no delays between lines.
- You use the PWIDTH command rather than WIDTH to determine the number of characters printed in each line.
- PLIST lists the lines of a text file *without their line numbers.* (BASIC files are print-listed with their line numbers intact.)

PLIST is useful because it enables your text files to be printed free of line numbers. All spaces after the line numbers are preserved during a PLIST operation.

Renumbering Lines (RENUMBER)

The RENUMBER command is used to renumber all or part of the lines of the current text or BASIC file.

```
RENUMBER [beginning line number [ , increment value [ , from old line number
  [ , through old line number]]]]
```

RENUMBER alone causes the entire file to be renumbered so that lines begin at 10 and increment by 10. If just the *beginning line number* is specified, renumbering begins at that line and increments by 10. Notice that zero, one or two parameters cause the entire file to be renumbered.

Examples: Press [RTN] after each RENUMBER command and then verify the renumbering by typing list [RTN].

```
:renumber
```

Renumbers the current file (EXAMPLE) so that the first line number is 10, the second is 20, and so on.

```
:renumber 100
```

Renumbers the entire file, beginning with a new line 100 and incrementing by 10.

```
:renumber 200,5
```

Renumbers the entire file, beginning with a new line 200 and incrementing by 5.

Using the last two RENUMBER parameters allows you to renumber selected *portions* of the current file.

Examples:

```
:renumber 800,2,220
```

Renumbers lines 220 to the end of the file (since no final parameter is specified). These last lines begin at 800 and increment by 2.

```
:renumber 10,1,1,210
```

Renumbers lines 1 through 210, beginning at line 10 and incrementing in steps of 1.

If you've tried the last three examples, then a listing of the file will show:

```
10 --Lord Byron--
11 She walks in beauty,
12   like the night
215 Of cloudless climes
800   and starry skies...
802
804 To put a line of characters
in your text file, just start ty
ping.
```

The renumbered beginning portion.

The renumbered ending portion.

The file pointer stays fixed at the same line during a renumbering, although the line number of the pending line may be changed.

RENUMBER will compress or expand segments of the file and open holes in the file, but it will not change the *order* of the lines. Warning 90—bad line number—and a default renumbering will occur anytime either of two conditions occurs:

- A RENUMBER command attempts to overlap or reorder the lines.
- A RENUMBER command forces a line number past 9999.

In these cases, the system will renumber the specified portion of the file, beginning at the first available line number and incrementing in steps of 1.

In conjunction with the MERGE command (page 60), RENUMBER allows you to combine two separate files into one larger file.

Deleting Lines (DELETE)

The DELETE command allows you to delete blocks of unwanted lines without affecting other lines.

```
DELETE [beginning line number [, ending line number]]
```

DELETE by itself deletes the pending line, although the file pointer stays set to that line number.

DELETE *line number* deletes the specified line.

Example:

```
:delete 802
```

Pressing [RTN] deletes line 802.

DELETE *beginning line number, ending line number* deletes the portion of the file bounded by and including the two line numbers.

Example:

```
:delete 12,800
```

Pressing [RTN] deletes lines 12 through 800.

Executing DELETE 0,9999 has the same effect as purging the entire file, except that DELETE doesn't destroy the file name. The file pointer is not affected by DELETE commands. It stays fixed at the pending line, even if that line is deleted.

Remember that the [DEL] key deletes characters, not lines.

File Manipulations

The RENAME, MERGE, COPY, EDIT, and NAME commands give you additional control over files in memory.

Renaming Files (RENAME)

The RENAME command changes the name of a file.

```
RENAME ['old filename'] TO 'new filename'
```

With only the *new filename* specified, the HP-75 renames the current file. Type the following and press [RTN]:

```
:rename to 'lb'
```

Renames the current file (EXAMPLE) to LB.

Executing CAT shows the new filename:

```
:cat
```

```
LB          T  126 13:07 02/05/83
```

To rename another file, specify its current name in the RENAME command.

Example:

```
:rename 'raymemo' to 'hank'
```

Pressing RTN causes RAYMEMO to be renamed to HANK.

Verify the name change by typing `cat 'hank'` RTN. Refer also to the discussion of the NAME command (page 64).

Merging Files (MERGE)

You may sometimes want to combine two text or two BASIC files into one larger file. The MERGE command makes file merging possible.

```
MERGE 'filename' [,beginning line number [,ending line number]]
```

MERGE causes the specified lines of the specified file to be merged into the *current* file. The two files must be of the same type (both being text or BASIC), and both must reside in memory. The specified file remains intact.

MERGE alone causes *all* lines of the named file to be merged into the current file. If one line number is specified, then only that line is merged. If two line numbers are specified, then all lines from the *beginning line number* through the *ending line number* inclusive are merged.

Example:

```
:merge 'hank',20,40
```

Pressing RTN causes lines 20 through 40 of the text file HANK to be added to the current file (LB).

Afterwards, list the current file to verify that merging has occurred. A check of the system catalog (CAT ALL) will show that the merged file (HANK) still exists in memory. The file pointer doesn't move as a result of performing a MERGE operation.

> **Important:** No renumbering occurs during a MERGE operation. If the current file contains any line numbers identical to those of the merged portion, then those lines of the current file are overwritten. You may want to ensure that the line numbers of the two files don't overlap. Use the RENUMBER command if necessary to change the range of the line numbers of the current file.

Copying Files to Cards (COPY)

The COPY command, used in section 1 to copy a prerecorded program to memory, also allows you to copy a file in memory to one or more magnetic cards.

```
COPY ['filename'] TO CARD
```

Without a filename, the COPY command defaults to the current file. By specifying a filename, you can copy another file in memory. To copy the current file, LB, type:

```
:copy to card
```

```
1 track(s) needed
```

This message is displayed according to the current DELAY setting.

After displaying this message, the HP-75 prompts:

```
Copy to card:  Align & [RTN]
```

Select a blank card from the Owner's Pac, align the card with the edge of the entry slot between the card's two alignment marks, and press [RTN].

```
Pull card ...
```

As you've done before, pull the card through the card reader smoothly but steadily.

Afterwards, you should see:

```
Verify card:  Align & [RTN]
```

The card reader prompts for a second pass of the same track, this time to verify the accuracy of information recorded on the track.

Align the card again and press [RTN]:

```
Pull card ...
```

The HP-75 allows approximately 5 seconds for you to pull a card when prompted.

If the file has been copied correctly, the prompt and cursor will appear after the second pass:

```
:
```

Indicates the file has been copied to the card.

If `WARNING: verify failed` is displayed after the second pass, it means that the information on the track didn't verify. If this occurs, the HP-75 will prompt you to reinsert the track for another cycle:

```
Copy to card:  Align & [RTN]
```

You'll need to pass the track through the card reader two more times, once to copy and once to verify.

Card cleanliness, always important, is *essential* during a copy to card operation. A dirty or damaged card may cause the HP-75 to repeat warning messages and card reader prompts indefinitely. Refer to appendix B for card cleaning information.

To copy a card file back into memory, follow the procedure you used in section 1 to copy the prerecorded `MONEY` program. In so doing, you must supply a new filename for the incoming file—each file in memory must have a unique name.

Duplicating Files (`COPY`)

The `COPY` command has another use besides transferring files to and from magnetic cards: It allows you to duplicate files in memory.

```
COPY ['original filename'] TO 'new filename'
```

When you copy one file to another, you create a new file and copy to it all the lines of the original. Without the *original filename* specified, `COPY` duplicates the current file.

Examples:

```
:copy to 'new'
```

Pressing [RTN] causes the contents of the current file (`LB`) to be copied to the `NEW` file.

```
:copy 'singsong' to 'tunes'
```

Pressing RTN causes the contents of SINGSONG to be duplicated in a new file named TUNES.

Now check the catalog entry of NEW:

```
:cat 'new'
```

```
NEW          T  191 15:04 02/05/83
```

Shows that NEW is the same type and size as the current file.

Notice that the time listed in the NEW catalog entry is more recent than the time of the LB file. The reason is that the NEW file is created when you execute the COPY command. Use the COPY command when you want to save a copy of a file before editing it or when you want to update the time and date of a file.

Locating and Creating Files (EDIT)

The EDIT command allows you to move the file pointer from one file to another as well as to create new files.

```
EDIT ['filename' [,BASIC]
                 [,TEXT]]

     BASIC
     TEXT
     KEYS
```

If a file exists in memory, then naming the file in an EDIT command causes the file pointer to be moved to the first line of that file. For example, move the file pointer to the first (and only) line of the SINGSONG file:

```
:edit 'singsong'
```

```
SINGSONG     B    47 09:45 02/05/83
```

The catalog entry of the SINGSONG file is displayed.

Press FET followed by RTN to verify the position of the file pointer. Because SINGSONG is a BASIC file (as indicated by the B in the file catalog), the BASIC prompt (>) appears instead of the text prompt. The file is ready to be edited. You can add lines, list lines, change lines, renumber lines, delete lines, and so on.

Use the EDIT command to create new BASIC and text files. For example, this time specify a nonexistent file:

```
>edit 'barbara'
```

```
BARBARA      B     0 15:40 02/05/83
```

BARBARA, a BASIC file, has just been created. The file pointer is positioned to line 0.

If neither BASIC nor TEXT is specified in the EDIT command, then the created file will be of the same *type* as the current file. For example, the BARBARA file is created as a BASIC file, the same type as SINGSONG.

You can create either BASIC or text files by specifying BASIC or TEXT in the EDIT command. Example:

```
>edit 'Eric',text
```

```
ERIC          T     0 15:42 02/05/83
```

Use a comma to separate the two parameters.

The specified text file is created.

Note that it's not possible to use the same filename for two different files, even if one is BASIC and one is text.

The system catalog lists the files in the order of their creation. For example, ERIC, the most recent file, will appear first in a CAT ALL listing. However, to conserve memory, a current file that is empty is purged from memory when you begin editing another file—and it won't appear in the catalog.

To edit the keys file (EDIT KEYS), refer to section 10, page 144.

The workfile (EDIT, NAME)

The workfile is a temporary text or BASIC file that the HP-75 creates on five occasions:

- When you first install batteries and set the clock.
- When you reset the HP-75.
- When you PURGE the current file.
- When you create a new text or BASIC file without specifying its filename in the EDIT command.
- When you execute the NAME command (page 64).

The purpose of the workfile is to provide you with a "scratchpad" file in memory. You can edit and manipulate a workfile in the same ways as any other text or BASIC file. To create a workfile, execute an EDIT command without specifying a filename:

```
:edit text
```

```
workfile      T     0 15:46 02/05/83
```

The EDIT command with no filename.

Creates workfile, a text file.

Executing edit basic creates a BASIC workfile. Executing edit alone creates a workfile of the same type (text or BASIC) as the current file.

Example:

```
:edit
```

```
workfile      T     0 15:52 02/05/83
```

EDIT alone.

Creates a new workfile, the same type as before (text).

A workfile can exist only as the *current* file; thus, there can be only one workfile in memory at a time. Before leaving a nonempty workfile—that is, a workfile containing at least one line—you must either purge the workfile or give it a name. If error 69—workfile name?—occurs when you execute

an EDIT command, it means that you've attempted to move the file pointer out of a nonempty workfile. For example, add a line to the workfile:

```
:3 The file is no longer empty.
```

```
:
```

Line 3 is entered in the text workfile.

Then try to edit another file without naming the current file.

```
:edit
```

Pressing RTN causes error 69—workfile name?—to be displayed. You're not allowed to leave a nonempty workfile.

The NAME command does two things:

- Renames the current file.
- Creates another workfile of the same type, text or BASIC.

```
NAME 'filename'
```

Executing NAME has the same effect as executing RENAME TO '*filename*' and EDIT separately. For example, use NAME to name the current file and create another workfile:

```
:name 'finish'
```

Specifying a name for the current workfile.

```
workfile    T     0 15:59 02/05/83
```

Renames the old workfile to FINISH and creates a new workfile of the same type—in this case, text.

NAME is provided as an editing convenience for naming a workfile before creating another workfile. However, you can execute NAME from any file in memory. Although you can NAME an *empty* file—that is, a file whose catalog entry shows a length of zero bytes—that file will disappear from memory as soon as the file pointer is moved to the new workfile.

Transforming Files (TRANSFORM)

The TRANSFORM command enables you to transform text files into BASIC files, BASIC files into text files, and both types of files into interchange, or LIF1, files. Refer to Special Files, appendix B, page 274, for more information.

Command Summary

Below is a summary of the 14 file commands introduced in this section.

Command	Parameters*	Operates On†	Must Be Current File?	May Move File Pointer From Current Line?
AUTO	Ln,Incr	B,T	yes	yes
CAT	Fn‡	B,T,A,L,I	no	no
COPY	Fn,Fn‡	B,T,A,L,I	no	no
DELETE	Ln,Ln	B,T	yes	no
EDIT	Fn	B,T	no	yes
FETCH	Str,Ln	B,T	yes	yes
FETCH	Ln	B,T	yes	yes
LIST	Fn,Ln,Ln	B,T	no	no
MERGE	Fn,Ln,Ln	B,T	yes	no
NAME	Fn	B,T	yes	yes
PLIST	Fn,Ln,Ln	B,T	no	no
PURGE	Fn§	B,T,A,L,I	no	no¶
RENAME	Fn,Fn§	B,T,A,L,I	no	no
RENUMBER	Ln,Incr,Ln,Ln	B,T	yes	no
TRANSFORM	Fn	B,T,I	no	no

* Ln = line number, 0-9999; Fn = filename, in single or double quotation marks; Incr = increment value, 1-9999; Str = quoted string.

† B = BASIC file; T = text file; A = appointment file; L = LEX file; I = interchange file.

‡ May specify a file residing on a magnetic card or tape.

§ May specify a file residing on a magnetic tape.

¶ If the current file is purged, the file pointer is positioned to line 0 of a new `workfile`.

Private BASIC files, indicated by `PB` in the system catalog, may be copied to memory, run, and purged only (page 117).

A command that takes line numbers or line increments must be supplied with *unsigned integers*. For example, `LIST 700,750` is a valid command; `DELETE A,A+50` is not.

A command that takes quoted strings may be supplied with any suitable string variable or expression (refer to section 5, page 78).

Examples: If string variable `B$` is assigned `'DISP'`, then `FETCH B$` causes the HP-75 to search the current BASIC or text file for the next occurrence of `DISP`. If `A$` has the value `'money'`, then `PURGE A$` purges the `MONEY` file.

String variables may not be used to specify *keywords*. For example, the words `APPT`, `KEYS`, `TEXT`, and `BASIC` must be typed explicitly in commands.

Note: Filenames are usually terminated by the end quotation mark. However, the first *blank* after the filename may be used to terminate the filename. For example, `edit 'scores and averages'` is the equivalent of `EDIT 'scores'` and names the file `SCORES`.

All of the above commands are programmable. Refer to section 11, page 156, for programming information.

Section 4

4 Keyboard Calculations

Contents

Introduction 68
Arithmetic Operations 69
Addition (+) 69
Subtraction (−) 69
Multiplication (*) 70
Division (/) 70
Exponentiation (^) 70
Integer Division (DIV and \) 70
Multiple Calculations 71
The Last Result (RES) 71
Arithmetic Hierarchy 72
Parentheses ( ) 72
Numeric Precision 73
Number Formatting 73
Floating Point Format 73
Exponential Notation (E) 74
Range of Numbers (INF, EPS) 76

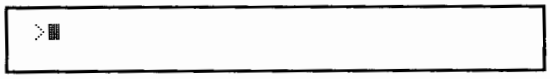
Introduction

The HP-75 can evaluate arithmetic expressions (such as 64.32 + 128.16) in either of two ways:

- By calculating the answers directly as you type.
- By computing the answers as part of BASIC programs.

This section explains the first—the calculator way of doing arithmetic. The HP-75 calculates to 12-digit precision over the range of numbers from -10^{499} to $+10^{499}$. *Numeric* (or algebraic) variables and functions are discussed in section 5.

All keyboard calculations are performed in EDIT mode. Press [EDIT]:



The BASIC prompt should be in the display for calculating.

If the text prompt (:) appears in the display instead of the BASIC prompt, it means that a display line beginning with a number from 0 through 9999 will be interpreted as a line of text and will be stored rather than executed.*

All keyboard calculations consist of two steps:

1. Type the desired numbers and operators in the display (without an equals sign). Up to 94 characters may be used in a single expression, and spacing is not important.

* Remember, however, that you can do arithmetic while editing a text file. Prefix the number in the display with a plus sign, a minus sign, a decimal point, or a left parenthesis.

2. Press [RTN]. The replace cursor or the insert cursor may be anywhere on the display line when this occurs.

Results are displayed with a leading blank or minus sign, a decimal point (when necessary), and a trailing blank.

Note that keyboard calculations have no effect on the contents of any file.

The HP-75 numeric keypad, illustrated on page 28, is enabled by pressing [CTL] [LOCK]. The numeric keypad is a convenience to speed up keyboard calculations. To restore the normal keyboard, press [LOCK].

Arithmetic Operations

Addition (+)

To add two or more numbers, use the [+] key.

Example:

```
>2819+3017
```

```
 5836
```

The result is displayed without prompt and cursor.

Don't type commas because they're interpreted as number separators, not digit separators.

Subtraction (−)

To subtract one number from another, use the [−] key.

Example:

```
>.051379-1.11750
```

```
-1.066121
```

The first display position is reserved for the negative sign.

If three or more numbers are combined with + and − operators, the HP-75 evaluates the expression from left to right.

Example:

```
>24-12-3
```

```
 9
```

Evaluated as (24 − 12) − 3.

The minus sign also works as a *unary* operator—you can use it on single numbers.

Example:

```
>--3
```

```
 3
```

Returns the opposite of −3.

Multiplication (*)

To multiply two or more numbers use the [*] key.

Example:

```
>30.4*12*365
```

```
133152
```

The decimal point is suppressed because the result is an integer.

Division (/)

To divide two numbers, use the [/] key.

Example:

```
>4/1.25
```

```
3.2
```

The result. The HP-75 suppresses trailing zeros.

When three or more numbers are combined with * and / operators, the HP-75 processes them from left to right.

Example:

```
>24/12*7
```

Spacing is unimportant.

```
14
```

Evaluated as (24/12)*7.

Exponentiation (^)

The exponentiation operator (^) raises a number to a given power. For example, to raise 5 to the third power (5^3):

```
>5^3
```

The ^ symbol is produced by [SHIFT] [*].

```
125
```

Exponents may be negative as well as positive, decimal as well as integer.

Raising zero to the power of zero causes warning 6—`0^0`—to occur and a default value of 1 to be supplied. Raising zero to a negative power causes warning 5—`0^neg`—to occur and a default value of $9.99999999999 \times 10^{499}$ (largest possible number) to be supplied. Raising a negative number to a non-integer power causes error 9—`neg^non-integer`—to occur and no default to be supplied.

Integer Division (DIV and \)

The DIV operator divides two numbers but returns just the integer portion of the quotient—with no rounding.

Example:

```
>200 div 3
```

The DIV operator.

```
66
```

The whole number of times 3 divides into 200.

That is, A DIV B = IP(A/B), where IP is the *integer part* of A/B.

You may enter the names of DIV and other operators and functions in lowercase or uppercase letters. Also, you may type a backslash (\) instead of DIV as the integer division symbol.

Example:

```
>200\3
```

The backslash is displayed by pressing CTL /.

```
 66
```

Identical to 200 DIV 3.

Refer also to the MOD function, page 83.

Multiple Calculations

More than one expression can be evaluated at a time. Use commas or semicolons to separate expressions.

Examples:

```
>2^7,-.04*100
```

```
 128                    -4
```

The comma causes the second result to be displayed beginning at position 22.

```
>2^7;-.04*100
```

```
 128 -4
```

The semicolon causes the second result to be displayed immediately after the trailing blank of the first result.

The Last Result (RES)

A fixed location in memory stores the result of your most recent calculation until it's replaced by another computed value. Any time you want to recall the last result, you can use the RES (*result*) function, which accesses this memory location.

To return the result of the calculation above, type:

```
>res
```

The RES function.

```
-4
```

The most recent result.

To add 3 to the value of the result, type:

```
>res+3
```

The RES function can be used in place of a number.

```
-1
```

This result (−1) is now stored in the RES memory location.

Note: During program execution, RES returns the last calculation to be displayed or printed by the HP-75, not necessarily the last computed value.

Remember that the CTL FET keystroke recalls the contents of the input buffer to the display. Press CTL FET to recall an entire calculator expression. You can then add to or edit the expression and press RTN to reevaluate it.

Arithmetic Hierarchy

When an expression contains two or more arithmetic operations, they are performed in the following order:

Operator	Function	Precedence
^	Exponentiation.	Performed first.
*, /, and DIV or \	Multiplication, division, and integer division.	↓
+, −, and unary minus	Addition, subtraction, and negation.	Performed last.

Two or more operations at the same level are evaluated from left to right.

Examples:

```
>1+3*2
```

Multiplication is performed before addition.

```
 7
```

Evaluated as 1 + (3 * 2).

```
>-8+20/4^3
```

The HP-75 performs exponentiation first, then division, then addition.

```
-7.6875
```

Evaluated as −8 + (20/(4^3)).

```
>24/12/2
```

The division occurs from left to right.

```
 1
```

Evaluated as (24/12)/2.

The unary minus is applied as soon as another operator of equal or less precedence appears in the expression.

Example:

```
>-2*3^2+18
```

The unary minus applies only to 2*3^2 because the + operator is at the same level of precedence.

```
 0
```

The expression reduces to −18 + 18.

Parentheses ( )

Parentheses serve two purposes:

- To clarify the order of execution where it may appear ambiguous. For example, you may want to type (24/12)/2 to show explicitly the order of division.
- To alter the normal order of evaluation. For example, entering 24/(12/2) changes the order in which the two divisions occur.

Parentheses take highest precedence in the arithmetic hierarchy. When one pair of parentheses contains another pair, the innermost quantity is evaluated first.

Example:

```
>1.8+(5*(4-2))^3
```

Includes two pairs of nested parentheses.

```
 1001.8
```

Causes 4 − 2 to be evaluated first, then its product with 5.

The HP-75 automatically removes unnecessary parentheses from BASIC program statements.

Parentheses by themselves won't cause multiplication. For example, 3(9 − 5) must be written as `3*(9-5)`. The left and right brackets (`[` and `]`)are not used at all in keyboard calculations.

Numeric Precision

When you enter numeric data into the HP-75, either from the keyboard or from a `DATA` statement under program control (the `DATA` statement is discussed in section 14), the computer will read 13 significant digits of the input, round to 12 significant digits, and store the results.

The following examples will help clarify input data precision:

Type in:

```
>12.3456789012345
```

Value stored after [RTN]:

```
 12.3456789012
```

The 13th digit is less than 5, so the 12th digit is rounded down. Note also that the 14th digit is ignored in the rounding.

```
>12.34567890125
```

```
 12.3456789013
```

The 13th digit is greater than 5, so the 12th digit is rounded up.

HP-75 calculations are performed internally with 15 significant decimal digits, but results are stored and displayed rounded to 12 significant digits. When rounding, the 12th digit is rounded up if and only if the 13th digit is equal to or greater than 5.

Number Formatting

The HP-75 uses the conventions established by the American National Standards Institute to express numbers.

Floating Point Format

Numbers that can be entirely represented with 12 or fewer digits are expressed in *floating point* format, so called because the decimal point can "float", or appear, anywhere within the number. Leftover zeros to the right of the decimal point are suppressed.

Example:

```
>32.10000*2
```

```
 64.2
```

The trailing zeros are suppressed.

Leading zeros are also suppressed.

Example:

```
>040350.3 + 10
```

```
 40360.3
```

The leftmost zero is suppressed.

All significant digits of a number are displayed, up to 12 digits.

Example:

```
>.123456789012345
```

A number with 15 digits.

```
 .123456789012
```

Rounded to 12 digits.

Numbers whose absolute values are greater than or equal to 1 but less than 1 trillion (10^{12}) are output showing up to 12 significant digits and no exponent.

Example:

```
>-999988887777.6666
```

Entering a 16-digit number.

```
-999988887778
```

The 12 most significant digits are displayed.

Numbers between −1 and 1 are output showing all significant digits and no exponent if they can be represented precisely in 12 or fewer digits.

Example:

```
>0.00112233
```

A decimal with 8 significant digits.

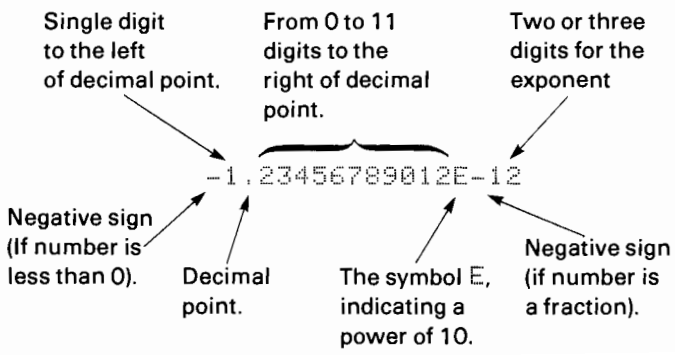
```
 .00112233
```

All significant digits are output.

All other numbers are expressed in exponential notation.

Exponential Notation (E)

Exponential, or scientific, notation is a short-hand system to express numbers too large or too small to fit the display normally—that is, numbers that can't be expressed adequately with 12 digits. The number −.000000000001234567890 12 expressed in exponential notation is:



Exponential representations have two parts—the *mantissa*, which consists of significant digits, and the *exponent*, which consists of a power of 10.

Examples:

```
>1000000000000000000000000
```

A 1 followed by 24 zeros.

```
1.E24
```

Expressed as a power of 10 with one significant digit.

```
>.000000000000547
```

A small positive number.

```
5.47E-13
```

Expressed as a negative power of 10 with three significant digits.

You may also *enter* numbers in exponential notation. However, the HP-75 will display a number in exponential notation only when a 12-digit figure won't do.

Examples:

```
>28.19e2
```

Positive mantissa and exponent.

```
2819
```

```
>2819e-3
```

Positive mantissa and negative exponent.

```
2.819
```

```
>-28.19e16
```

A number that exceeds 12 digits.

```
-2.819E17
```

Exponents must be integers from −499 to 499. Typing a + sign for nonnegative mantissas and exponents is optional.

Finally, you can enter expressions using mixed forms of floating point and exponential notation.

Examples:

```
>1e12-1000
```

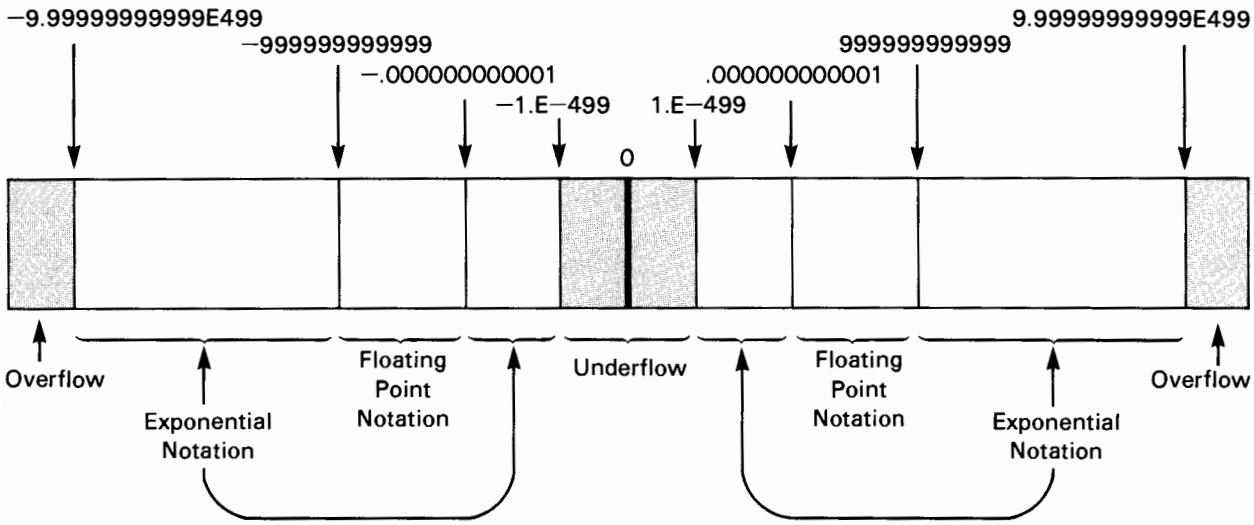
```
999999999000
```

```
>1e12+1000
```

```
1.000000001E12
```

Range of Numbers (INF, EPS)

The following diagram showing the range of values that can be entered and stored:



The largest number on the HP-75 is $9.99999999999 \times 10^{499}$. To access it quickly, use the INF (*infinity*) function:

```
>inf
```

The INF function.

```
 9.99999999999E499
```

Returns the largest machine number.

The largest negative number is the negative of INF:

```
>-inf
```

```
-9.99999999999E499
```

The smallest positive number on the computer is 1×10^{-499}. To display it, use the EPS (*epsilon*) function:

```
>eps
```

```
 1.E-499
```

Returns the closest positive number to 0.

Finally, the smallest negative number is the negative of EPS:

```
>-eps
```

```
-1.E-499
```

Using a value greater than INF or less than −INF causes warning 2—num too large—and a default value of plus or minus machine infinity to be supplied. Using a nonzero value between EPS and −EPS causes warning 1—num too small—and a default value of 0 to be supplied. (Refer also to the DEFAULT OFF and DEFAULT ON commands, page 89.)

Section 5

Numeric Functions and Expressions

5 Contents

Introduction 78
Variables 78
Naming Simple Numeric Variables 79
Assigning Values to Variables ([=], LET) 79
Precision of Numeric Variables (REAL, SHORT, INTEGER) 80
Calculator and Program Variables (CLEAR VARS) 81
Numeric Functions 82
- Number-Alteration Functions (ABS, IP, FP, INT, FLOOR, CEIL) 82
- General Functions (SQR, MOD, SGN, MAX, MIN, RMD, PI, INF, EPS, RND) 83
- Logarithmic Functions (LOG, EXP, LOG10) 84
- Trigonometric Commands (OPTION ANGLE RADIANS, OPTION ANGLE DEGREES) 85
- Trigonometric Functions (SIN, ASIN, COS, ACOS, TAN, ATN, ANGLE, CSC, SEC, COT, RAD, DEG) 85

Numeric Expressions 87
Relational Operators(=, < >, #, >, >=, <, <=) 87
Logical Operators (AND, OR, EXOR, NOT) 88
Precedence of Operators 89
Recovering from Mathematical Errors (DEFAULT ON, DEFAULT OFF) 89

Introduction

This section introduces the *numeric*, or mathematical, capabilities of the HP-75 that are beyond simple arithmetic. All numeric computations are performed in EDIT mode, either from the keyboard or during program execution. The following examples show the BASIC prompt in the display, although you may evaluate numeric expressions while editing a text file.

Variables

Frequently, you may want to represent a number or a string of characters by a *name* rather than by a specific value. The HP-75 makes such representations possible through its use of variables. A variable is simply a name for a location in memory that stores numeric or string information. Variables are a fundamental part of computer programming.

You can use the HP-75 to assign values to variables and to compute, modify, compare, and output variable values.

HP-75 variables come in three forms:

- Simple numeric variables, such as X and X1. These represent single numbers—integers (e.g., 2937) and decimals (4.07198)—and are covered in this section.
- Numeric array variables, such as A() and A1(,). These store quantities of like numbers (e.g., a series of temperature readings) and are convenient for handling lists and tables of numeric data in a program. Array variables are discussed further in section 13.

- String variables, such as S$ and S1$. These store character information (e.g., 'Hours worked:') of any length, from zero characters up to a maximum limited only by available memory. String variables may be used to specify filenames, HP-IL device codes, key definitions, etc., as well as used to input, process, and output textual information. String variables are discussed further in section 11.

For simplicity, the variables in the examples are keyboard, or calculator, variables; program variables are manipulated similarly.

Naming Simple Numeric Variables

On the HP-75 you can use the following for the names of simple numeric variables:

- Any letter from A through Z. (The examples use lowercase letters; they are interpreted as if they were capitals.)
- Any letter immediately followed by a digit from 0 through 9.

For instance, acceptable names are a1, C, R2, D2, z, J5, C3, and p0. In all 286 simple numeric variables can be named.

Assigning Values to Variables ([=], LET)

To assign a value to a variable, use the [=] and [RTN] keys.

Examples:

```
>a=15
```

Pressing [RTN] stores the value 15 in a variable (that is, a location in memory) named A.

```
>x3=2*25
```

The variable X3 is assigned the value 50 to the *right* of the equals sign.

The LET statement also assigns values to variables.

Example:

```
>let b6=x3
```

Pressing [RTN] assigns to B6 the current value of variable X3.

Note that the LET keyword is optional—a=15 is equivalent to let a=15.

After variables have values assigned to them, the variables can be used in place of numbers in keyboard and program calculations.

Examples:

```
>a^2
```

Using the exponentiation operator (^).

```
 225
```

The result of 15 squared.

```
>x3/b6
```

Combining the values of two variables with the division operator (/).

```
 1
```

The result of 50 divided by 50.

Variables may be quickly reassigned values. For instance, to change the value of A to 16, you can type a=a+1 [RTN], a=16 [RTN], or any comparable assignment. To display the value of a calculator variable, type the variable name and press [RTN].

Example:

```
>x3
```

To display a variable value.

```
 50
```

The current value of X3.

You can display the values of multiple variables in the same line by separating the variable names with semicolons or commas.

Example:

```
>a;x3,b6
```

Use semicolons for compact spacing, commas for 22-column spacing.

```
 15  50                  50
```

Three variable values are displayed.

Note that you can make multiple variable assignments in the same line. Use *commas* to separate the variable names.

Example:

```
>a,b,c=0
```

Pressing [RTN] causes variables A, B, and C to assume the value 0.

To check this assignment, recall the individual variable values:

```
>a;b;c
```

```
 0  0  0
```

The variables contain identical values.

Precision of Numeric Variables (REAL, SHORT, INTEGER)

Besides declaring the name and value of a numeric variable, you can declare its precision—that is, the number of digits used by the HP-75 to represent its value. Three types of precision are offered: REAL, SHORT, and INTEGER.

- REAL variable values are stored with the full precision of the HP-75. They cover the entire range of values, $-9.99999999999 \times 10^{499}$ through $-1.00000000000 \times 10^{-499}$, 0, and $1.00000000000 \times 10^{-499}$ through $9.99999999999 \times 10^{499}$. REAL numbers are represented internally by twelve digits and a three-digit exponent.
- SHORT variable values cover a narrower range, -9.9999×10^{99} through -1.0000×10^{-99}, 0, and 1.0000×10^{-99} through 9.9999×10^{99}. Accordingly, SHORT numbers are represented internally by five digits and a two-digit exponent.
- INTEGER variable values lie between ±99999. INTEGER numbers are stored with five digits and no exponent.

To specify the precision of a numeric variable, use REAL, SHORT, or INTEGER in a variable declaration.

Examples:

```
>real p
```

Declares P to be a REAL precision variable.

```
>short x,y,z
```

Declares X, Y, and Z to be SHORT precision variables.

```
>integer i,j,k
```

Declares I, J, and K to be INTEGER precision variables.

To check these declarations, assign the value of π to one variable of each precision:

```
>p,x,i=pi
```

Assigns the value of π to three variables according to their degree of precision.

Then recall the variables' individual values:

```
>p;x;i
```

```
 3.14159265359  3.1416  3
```

Values are displayed according to REAL, SHORT, and INTEGER precision.

Notice that rounding occurs when the number of digits exceeds the precision of the variable.

All numeric variable values are assumed full precision (REAL) unless you specify otherwise. For example, variables A, X3, B6, B, and C appearing earlier have had their values stored with REAL precision even though you didn't indicate their precision at the time you assigned them values.

The HP-75 doesn't allow you to change the precision of a variable after you've once declared its precision or assigned a value to the variable. For example, typing `short p` [RTN] now will cause error 35—`DIM exist var`—to occur, indicating an attempt to declare the precision of an existing variable.

Although REAL variables offer the greatest accuracy and the largest range in calculations, you conserve HP-75 memory if you designate SHORT or INTEGER variables.

Refer to System Memory Requirements, appendix D, for information regarding REAL, SHORT, and INTEGER variables.

Calculator and Program Variables (CLEAR VARS)

The values of calculator variables—that is, the values you assign from the keyboard—are preserved in memory until you change or clear them. This means that you can store calculator variable values for continual use. For example, you can first assign Avogadro's number to variable A from the keyboard, run a program, recall the value of A, and use the variable in other keyboard calculations.

Note that program variables and calculator variables are kept distinct. Program variables are assigned values only during program execution. A program uses variables strictly *local* to the program itself and cannot access calculator variable values. If program execution is interrupted for any reason, say, by pressing the [ATTN] key or executing a STOP statement, then program variable values are directly accessible from the keyboard. For example, if there is a variable A in the program as well as a calculator variable A, then you can check the value of *program* variable A after a program interruption and not the value of calculator variable A. However, you *can* reference other calculator variables at this point if they have names different from program variables. After the program finishes execution, you can recall all calculator variable values from before the program. Refer to Program Variables, section 11, for more information.

To clear all calculator and program variables and to free the memory required for them, use the CLEAR VARS command.

```
CLEAR VARS
```

You may wish to use the CLEAR VARS command occasionally (for example, right now) so that the variables won't consume an unnecessary amount of memory. After a CLEAR VARS command, the HP-75 reclaims the memory formerly taken by the variables. (It also "deallocates" a program that has been "initialized"—refer to page 158.) Using a limited number of *names* for calculator variables will minimize the amount of memory taken by the variables.

Numeric Functions

Numeric *functions* are built-in routines that take numeric or string information and return single numbers. The HP-75 is equipped with more than 40 predefined numeric functions; all can be executed from programs as well as from the keyboard. Some, like NUM and MEM, have been discussed previously. A complete list may be found in appendix H.

The information acted on by a function is called the *argument* of the function; an HP-75 function may operate on zero, or more arguments. An argument may itself be a variable, another function, or an entire expression, so long as it reduces to a constant at the time it's evaluated.

To execute any HP-75 function from the keyboard:

1. Type the function name using lowercase or uppercase letters.
2. Type the argument, if the function requires one, enclosed within parentheses. If the function requires multiple arguments, separate them with commas.
3. Press [RTN] to compute the result.

The following sections group the HP-75 numeric functions according to their use.

Number-Alteration Functions (ABS, IP, FP, INT, FLOOR, CEIL)

The table below lists the function name, argument, and meaning of six functions that alter numbers. Each function is listed with the value returned from a simple argument, *X*, which may be a constant (like 112.75), a variable (like A), another function (like SQR(A)), or an expression (like 112.75*SQR(A)).

Function and Argument	Meaning	Example
ABS(X)	Absolute value of X.	abs(-235) 235
IP(X)	Integer part of X—that portion of X to the left of the decimal point.	ip(10/3) 3
FP(X)	Fractional part of X—that portion of the number to the right of the decimal point (including the decimal point).	fp(10/3) .33333333333
INT(X)	The greatest integer less than or equal to X.	int(-7.23) -8
FLOOR(X)	Greatest integer less than or equal to X. (Same as INT(X).)	floor(7.23) 7
CEIL(X)	Smallest integer greater than or equal to X.	ceil(7.23) 8

Notice the difference between the IP, FLOOR (or INT), and CEIL functions. Given a positive argument, IP and FLOOR return identical values; given a negative argument, IP and CEIL return identical values.

General Functions (SQR, MOD, SGN, MAX, MIN, RMD, PI, INF, EPS, RND)

Of the ten general functions, four take two arguments (separated by commas), and four require no argument at all. Except for RND, the zero-argument functions are *constant* functions because they return the same value each time they're executed.

Function and Argument(s)	Meaning	Example
SQR(X)	Positive square root of X.	sqr(16.1) 4.01248052955
MOD(X,Y)	The integer remainder of X/Y, that is, X-Y*INT(X/Y).	mod(10,3) 1
SGN(X)	Sign of X—returns 1 if the argument is positive, 0 if it is 0, and −1 if it is negative.	sgn(-5) -1
MAX(X,Y)	Maximum of two values—returns the larger value.	max(4.5,4.67) 4.67
MIN(X,Y)	Minimum of two values—returns the lesser value.	min(-3,-2.999999) -3
RMD(X,Y)	Remainder of X divided by Y—that is, X-Y*IP(X/Y)	rmd(10,3) 1
PI no argument	Twelve-digit approximation of π.	pi 3.14159265359
INF no argument	Machine infinity—the largest HP-75 number.	inf 9.99999999999E499
EPS no argument	Epsilon—smallest positive HP-75 number.	eps 1.E-499
RND no argument	Random number—generates the next number R in a sequence of pseudo-random numbers such that $0 \leqslant R < 1$.	rnd .011750051479

RMD(X,Y) and MOD(X,Y) are the same when both X and Y are positive. However, the results may differ if X and Y are of opposite signs. For example, RMD(-37,7) equals −2, while MOD(-37,7) equals 5.

The RND function returns a new pseudorandom number every time it's evaluated; the value is greater than or equal to 0 but less than 1.

The *starting* number of a random number sequence determines the sequence of values that RND will return. At each evaluation, the RND function combines the last random number with a predefined multiplier, creating a new random number. The starting number may be set at any time by the RANDOMIZE statement, executed either from the keyboard or in a program.

```
RANDOMIZE [numeric expression]
```

To set the starting number for the random number generator:

- Execute RANDOMIZE alone, which causes the HP-75 to generate the starting number arbitrarily, based on the current system clock reading.
- Specify any constant or expression within the range of the HP-75 in a RANDOMIZE statement, which causes the HP-75 to start the sequence based on the value of that expression.

For example, to use 423 as the "seed" of a random number sequence, type:

```
>randomize 423
```

```
>
```

A new sequence of random numbers will be now returned by the RND function.

Check the starting number of this sequence:

```
>rnd
```

```
 .629385058782
```

The first number of the RND sequence, based on a seed of 423.

Use the same seed to produce the same sequence of random numbers in your programs. (A seed of zero will cause a constant sequence of zeros.)

In the absence of a RANDOMIZE statement, the HP-75 supplies a default starting number (.529199358633) for the RND function each time a program is run.*

For example, the SINGSONG program of section 1 without a RANDOMIZE statement always produces the same sequence of tones. The default starting number is also supplied when RND is entered from the keyboard after a program has been run that doesn't use the random number generator.

To generate random integers, $i_1, i_2, \ldots i_j \ldots$ such that $A \leqslant i_j \leqslant B$, and where A and B represent any two real numbers (with $|A|, |B| < 10^{13}$) use the following formula:

$$i_j = \text{IP}((B + 1 - A) * \text{RND} + A).$$

For instance, to generate a random number from 1 through 100, type:

```
>ip(100*rnd+1)
```

Finding the next random number, 1 through 100.

```
 75
```

The result, based on the second random number in the sequence from above.

Good statistical properties can be expected from the random number generator if a statistically significant sample size is considered.†

Logarithmic Functions (LOG, EXP, LOG10)

The HP-75 computes both natural and common logarithms as well as their inverses.

Function and Argument	Meaning	Example
LOG(X)	Ln X (or $\log_e$ X)—the natural logarithm of a positive X to the base *e*.	log(26) 3.25809653802
EXP(X)	e^X—the natural antilogarithm.	exp(1) 2.71828182846
LOG10(X)	$\log_{10}$ X—the common logarithm of a positive X to the base 10.	log10(1000) 3

* If one program calls another program, the sequence of random numbers begun in the first program is continued by the second program, assuming the second program doesn't execute its own RANDOMIZE statement (page 234).

† The HP-75 random number generator has passed the Spectral Test. Donald E. Knuth, *The Art of Computer Programming* (Massachusetts, 1969), vol. 2, section 3.4.

The common antilog (10^X) may be found using the exponentiation operator: `10^X`.

Trigonometric Commands (OPTION ANGLE RADIANS, OPTION ANGLE DEGREES)

Use either of two commands to set the HP-75 to the desired trigonometric mode for executing trigonometric functions.

```
OPTION ANGLE RADIANS
```

There are 2π radians in a circle. This is the setting of the HP-75 after a system reset occurs.

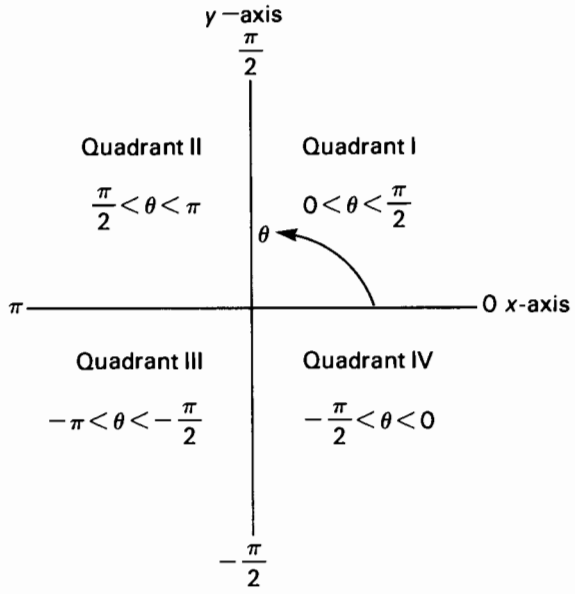
```
OPTION ANGLE DEGREES
```

There are 360 degrees in a circle.

The current trigonometric setting remains in effect until you change it with another `OPTION ANGLE` command.

Trignometric Functions (SIN, ASIN, COS, ACOS, TAN, ATN, ANGLE, CSC, SEC, COT, RAD, DEG)

The HP-75 provides 12 predefined trigonometric functions. It's important to keep in mind the range of values that the *inverse* functions (arcsine, arccosine, and arctangent) return, which may lie in Quadrants I through IV. Assuming *radians* mode, the HP-75 represents angles as follows:

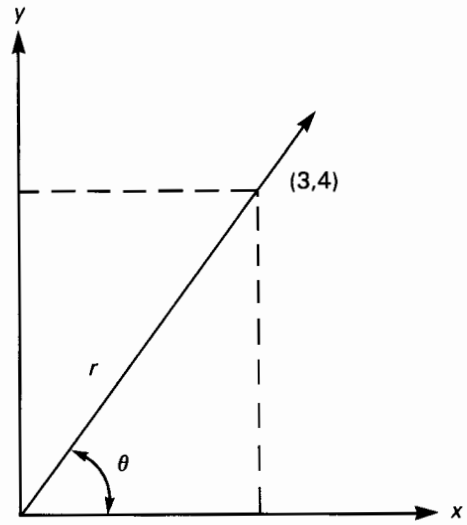


Function and Argument	Meaning	Example (assuming an OPTION ANGLE RADIANS setting)
SIN(X)	Sine of X.	sin(pi/2) 1
ASIN(X)	Arcsine of X, where $-1 \leq X \leq 1$. In Quadrant I or IV.	asin(1) 1.57079632679
COS(X)	Cosine of X.	cos(0) 1
ACOS(X)	Arcosine of X, where $-1 \leq X \leq 1$. In Quadrant I or II.	acos(1) 0
TAN(X)	Tangent of X.	tan(pi/4) 1
ATN(X)	Arctangent of X. In Quadrant I or IV.	atn(1) .785398163397
ANGLE(X,Y)	Arctangent of Y/X, in proper quadrant, that is, the angle θ between (X,Y) and the x-axis.	angle(sqr(3),1) .523598775598
CSC(X)	Cosecant of X.	csc(pi/6) 2
SEC(X)	Secant of X.	sec(pi/3) 2.00000000001
COT(X)	Cotangent of X.	cot(pi/6) 1.73205080757
RAD(X)	Degrees to radians conversion.	rad(45) .785398163397
DEG(X)	Radians to degrees conversion.	deg(pi) 180

The ATN and ANGLE functions are useful in polar/rectangular coordinate conversions. Given the x and y coordinates of a point in Quadrants I or IV, both ATN and ANGLE return the measure of the angle formed by the x-axis and by the line running from the origin through the point.

Example: Convert rectangular coordinates (3,4) to polar form with the angle expressed in degrees.

Quadrant I



Solution:

Use the Pythogorean theorem to solve for r:

$$r = \sqrt{x^2 + y^2}$$

Use either ATN or ANGLE to solve for θ.

θ = ATN(4/3) or θ = ANGLE(3,4)

These steps provide the solution:

```
>option angle degrees
```

Pressing [RTN] sets the HP-75 to degrees mode.

```
>sqr(3^2+4^2)
```

```
 5
```

Solving for r.

```
>angle(3,4)
```

```
 53.1301023542
```

Solving for θ.

The measure of θ in decimal degrees.

Note the difference between the ANGLE and ATN functions. ANGLE takes two arguments to find the proper quadrant of their arctangent. ATN returns the principal value of the arctangent—that is, the value in Quadrant I or IV—of a single argument. For example, ANGLE(-3,-2) returns −146.309932474 degrees (in Quadrant III), whereas ATN(-2/-3) returns 33.690067526 degrees (in Quadrant I).

Numeric Expressions

Numeric expressions appear in every mathematical operation. The simplest numeric expressions are just constants, like 5. Simple numeric variables (like X3) and functions (like sqr(2)) also constitute primitive expressions. However, numeric expressions can consist of any series of constants, variables, and functions combined by operators and parentheses. Section 4 discussed arithmetic operators, such as + and ^, that combine expressions. The following topics cover two other types of operators—relational and logical operators—that *compare* expressions.

Relational Operators (=, <>, #, >, >=, <, <=)

The seven HP-75 relational operators compare the values of two expressions and return a 1 if the comparison is true, a 0 if the comparison is false. That is, the relational operators operate on numeric values to return Boolean values.

Relational Operator	Comparison	Example
=	Equal to?	`1=1` ` 1`
<> and #	Not equal to?	`3<>4` ` 1`
>	Greater than?	`1>-1` ` 1`
>=	Greater than or equal to?	`3.14>=pi` ` 0`
<	Less than?	`3.14<pi` ` 1`
<=	Less than or equal to?	`-eps<=0` ` 1`

Note that the equals sign (=) is used in both variable assignment statements and in relational expressions. Whenever an entry can be interpreted either way, the HP-75 assumes the entry is a variable assignment.

Examples:

```
>a=3
```

Does this set A equal to 3 or instead compare the two values?

```
>
```

Interpreted as a variable assignment.

```
>(a=3)
```

Using parentheses to specify a numeric expression.

```
 1
```

Indicates the relationship is true.

```
>3=a
```

Typing the reverse order forces evaluation as a comparison.

```
 1
```

Shows that A currently has the value of 3.

```
>b=a=3
```

Pressing RTN causes B to be assigned the value of (a=3)—that is, the value 1.

Because an equals sign can denote both a variable assignment and a relational operation, its use should be clearly expressed. Relational operators are frequently used to control the order of program execution (section 12, page 177) and may be used to compare string values (section 13, page 203) as well as to compare numeric values.

The relational operators use subtraction to evaluate the relationship. Numeric values that would produce an error condition for subtraction will produce the same error for any of the relational operators.

Logical Operators (AND, OR, EXOR, NOT)

The four logical operators operate on Boolean values to return Boolean values. The logical operators interpret all nonzero numeric operands as 1, or true, and all operands equal to zero as 0, or false. AND, OR, and EXOR return a value of 1 if the relationship between operands is true and a value of 0 if the relationship is false. NOT, a unary operator, returns the opposite value (0 or 1) of a single operand.

Logical Operator	Basis of Evaluation	Example
AND	Both expressions true (that is, nonzero)?	3 and 4 1
OR	Either expression true?	3 or 0 1
EXOR	One or the other expression true—but not both? This is the equivalent of (A AND NOT B) OR (B AND NOT A).	3 exor 0 1
NOT	Is the expression false (that is, zero)?	not 1 0

Relational and logical operators may be used to compare numeric constants (3 OR 0), variables (A AND B), functions (SIN(A) AND COS(A)), and larger expressions. For example, if A = 0 and B = 20, then:

```
>not a and (b<50-b)
```

Entering a numeric expression with logical, relational, and arithmetic operators.

```
 1
```

The expression evaluates "true."

Note that the pairs of parentheses dictate the order in which expressions are evaluated.

Precedence of Operators

Parentheses take highest precedence when the HP-75 computes the value of an expression. Other operations follow according to their placement in the following table.

Precedence	Operation
Highest	( ) Parentheses.
↓	Functions.
↓	^ Exponentiation.
↓	NOT.
↓	*, /, DIV or \.
↓	+, −, unary minus.
↓	=, < >, or #, >, >=, <, <= Relational operators.
↓	AND.
Lowest	OR, EXOR.

When an expression contains two or more operations at the same level, the HP-75 performs them from left to right.

Recovering from Mathematical Errors (DEFAULT ON, DEFAULT OFF)

If you type a mathematical expression incorrectly, a syntax error may occur. Assuming an expression is correctly typed, a mathematical error may occur because of an improper argument or an undefined value. Left to itself, such an error would halt the execution of a running program. However, the HP-75 provides default values for out-of-range results for the functions listed below, thus overriding the error condition and preventing the error from halting execution. The computer will alert you to the error by beeping and displaying a WARNING message. Then it will supply a default value for the expression and continue program execution (unless ON ERROR has been declared—page 258).

Assuming an OPTION ANGLE DEGREES setting, the warning conditions and default values are:

Warning Number	Warning Condition	Default Value
1	Underflow; that is, a nonzero result between −EPS and +EPS.	0
2	Variable precision overflow: • For INTEGER variables. • For SHORT variables. • For REAL variables.	 ±99999 ±9.9999E99 ±9.99999999999E499
3	COT or CSC equal to infinity, caused by an argument equal to a multiple of 180°.	9.99999999999E499
4	SEC or TAN equal to infinity, caused by an argument equal to an odd multiple of 90°.	9.99999999999E499
5	Zero raised to a negative power.	9.99999999999E499
6	Zero raised to a power of zero.	1
7	Using an unassigned numeric variable value.	0
8	Division by zero.	9.99999999999E499

For instance, set the computer to OPTION ANGLE DEGREES and try taking the secant of 90°:

```
>sec(90)
```

To produce an out-of-range result.

```
 9.9999999999E499
```

The beep sounds and a WARNING message appears, but the **ERROR** annunciator stays off and the HP-75 supplies a default value—in this case, machine infinity.

After a system reset occurs, the HP-75 is set to supply the above default values so that out-of-range mathematical errors won't stop program execution (although a WARNING message appears). To catch such errors, execute the DEFAULT OFF command.

```
DEFAULT OFF
```

Typing default off [RTN] cancels the use of default values for mathematical errors and sets the system to normal error processing. For instance, if DEFAULT OFF is the current setting, then evaluating sec(90) causes error 4—TAN or SEC inf—to be displayed, the **ERROR** annunciator to light, and no default value to be supplied. This error would now halt a running program.

To set the system to default error processing, type:

```
>default on
```

Pressing [RTN] restores the way the HP-75 handles errors initially.

Warning 7—no value—occurs when you try to use an *undefined* variable—that is, a variable that hasn't had a value assigned to it.

Example:

```
>b9
```

To display the value of an unassigned variable.

```
WARNING: no value
```

This message is displayed, and 0 is left in the display.

With DEFAULT ON, the HP-75 supplies zero as the default value of an undefined numeric variable and the null string (consisting of zero characters) as the default value of an undefined string variable. However, the variable itself is *not* assigned a value and remains undefined until you explicitly assign it a value.

Section 6

TIME Mode Operations

Contents

6

Introduction 92
Clock Operation (SET) 93
Changing to Day\Month\Year and 24-Hour Formats (STATS) 93
Adjusting the Clock (ADJST) 95
  Calibrating the Clock (EXACT) 95
  Resetting the Clock Rate (RESET) 97
  Normal and Absolute Adjustments (N, A) 97
Clock Functions (DATE, TIME, DATE$, TIME$) 98

Introduction

Pressing the [TIME] key causes the HP-75 to switch to TIME mode and display the day-of-week, date, and time:

```
MON 02/14/1983 01:03:15 PM
```

A typical TIME display.

This section shows how to specify three different TIME display formats:

```
MON 14\02\1983 01:04:48 PM
```

A TIME display showing alternate day\month\year format.

```
MON 02/14/1983 13:04:48 **
```

A TIME display showing 24-hour format. The time advances from 00:00:00 (midnight) through 23:59:59.

```
MON 14\02\1983 13:04:48 **
```

A TIME display showing combined day\month\year and 24-hour formats.

The cursor in the TIME display is positioned at the first character of the five-character *command field*:

```
MON 02/14/1983 01:04:48 PM
```

The TIME display command field.

There are five commands you can enter in the command field to control the setting, display formats, and accuracy of the system clock.

SET	Sets the clock.
STATS	Specifies a month/day/year or a day\month\year format, an AM/PM or a 24-hour format, and the appointment calendar mode.

ADJST	Adjusts the clock setting.
EXACT	Establishes exact timing marks to calibrate the clock.
RESET	Clears the timing marks and speed adjustment.

These commands can be entered in the command field in lowercase or uppercase letters and are executed as soon as [RTN] is pressed. If an out-of-range value is entered in any TIME mode template, then error 89—bad parameter—will occur. If an illegal character is entered in the command field, then error 78—syntax—will occur. In either case the incorrect line will reappear for correction.

Clock Operation (SET)

The clock reading is displayed until:

- You change operating modes by pressing [EDIT] or [APPT].
- You execute a SET, STATS, or ADJST command in TIME mode, as discussed in this section.
- The HP-75 turns off—it will turn back on in EDIT mode.

The HP-75 clock operates continuously, even while the HP-75 is off. Only a system reset will cause the clock to suspend operation (refer to Resetting the HP-75 in section 1 for conditions that cause a reset).

Should a system reset occur, when the machine is again turned on (with [ATTN]), the set-time template appears and the clock starts up from the following values:

01/01/0000	January 1st, 0000 A.D. for the date.
12:00:01 AM	One second after midnight for the time.

The TIME display will show these values if a reset is immediately followed by [ATTN] pressed twice, or [ATTN] followed by any other carriage-return/line-feed key (page 43).

Typing set [RTN] in the TIME command field also displays the set-time template. Press [ATTN] to cancel a set-time template—the TIME display will reappear.

Use the [TAB] key to skip across display fields and [SHIFT] [TAB] to skip back across display fields as you fill in a set-time template. In fact, any editing key may be used to fill in any TIME mode template except for the [I/R] key, which is disabled in TIME mode. For example, the [CLR] key clears any TIME template.

The hour and AM/PM fields are not totally separate. If you change the AM/PM field, you *must* enter a number in the hour field also—the current hour is associated with the current AM/PM value. You may enter an integer from 13 through 23 in the hour field if you also type ** (two asterisks) in the AM/PM field, instead of AM or PM.

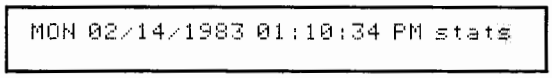
The clock is set to the time in the template as soon as the HP-75 senses a [RTN]. There is about a 0.1 second delay in the process. Unfilled and blank set-time fields will default to the present date and time values.

Changing to Day\Month\Year and 24-Hour Formats (STATS)

The STATS (*status*) command enables you to choose between:

- Month/day/year and day\month\year formats.
- AM/PM and ** (24-hour) formats.
- The normal year calendar and the extended calendar for scheduling appointments.

The following procedure changes the HP-75 to day\month\year and 24-hour format. Type stats in the TIME command field and press [RTN]:



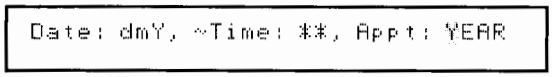
The STATS template appears.

The ~ (tilde) symbol before the word Time signifies that the time is *approximate*—that is, you haven't yet calibrated the clock (page 95).

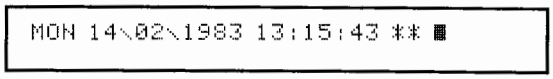
At this point, you may:

- Change to a different date format by typing DM (in uppercase or lowercase letters) so that MDY becomes DMY.
- Change to 24-hour notation by typing ** in the AM display field.
- Type EXTD (*extended search*) over the word YEAR. Doing so will change the way the HP-75 works in APPT mode. Appointment searches are discussed in section 7, APPT Mode Operations.

Type the following in the STATS template:

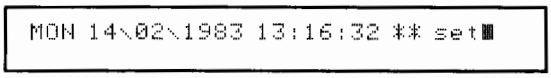


The modified STATS template. Notice the cursor skips to the correct field.

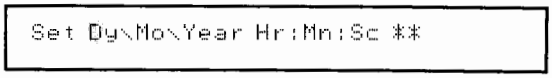


The day\month\year format is indicated by back slashes (\). The double asterisk (**) indicates 24-hour notation.

Afterwards, executing the SET command causes the following display:

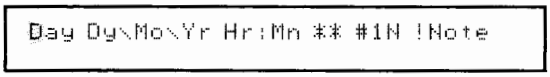


The SET command in the TIME display.



The set-time template uses the newly specified formats.

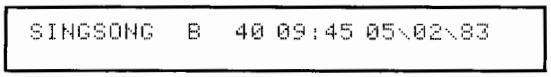
Press [ATTN] to return the TIME display. Pressing [APPT] shows that the APPT template now has a new look:



The new formats carry over to APPT mode displays.

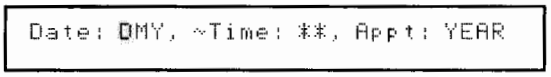
New appointments are entered and old appointments are displayed according to these new formats.

Finally, press [EDIT] to enter EDIT mode and type catall [RTN]. Then use the [↓] key to review your files in memory. You'll notice that their catalogs appear with the new day\month\year format. Example:



The file catalogs show the new day\month\year format. (Catalogs always use 24-hour notation.)

New TIME display formats remain in effect until they're changed by another STATS procedure or by a machine reset. To change the new formats to their original forms, reenter TIME mode (press [TIME]) and retype stats [RTN]:



The new STATS template.

Type md in the DMY field; then type either am or pm over the double asterisk (**)—the effect is the same. Then press [RTN]:

```
Date: mdY, ~Time: am, Appt: YEAR
```

Altering the STATS template.

```
MON 02/14/1983 01:16:54 PM
```

The original month/day/year and AM/PM formats are restored.

Adjusting the Clock (ADJST)

The ADJST command lets you make *relative* adjustments to the clock setting. You can adjust the HP-75 clock setting ahead or back as many as 99 hours, 99 minutes, and 99.9 seconds.

Example: While flying from San Francisco to New York, you wish to set the clock ahead 3 hours. Type adjst in the TIME command field and press [RTN]:

```
Adjust (N)  + Hr+Mn+Sc.t
```

The ADJST template appears.

This procedure is *normal* adjustment, so the N field won't be changed. To set the clock to an earlier time—that is, to subtract time from the setting—you need to press the [-] key to enter a minus (–).

In this example, you wish to set the clock ahead 3 hours—that is, to add 3 hours to the setting—so you leave the plus sign (+) alone and press [TAB] to move the cursor on to the Hr (hours) field. Type in the amount of adjustment. The digit in each position may range from 0 through 9. Unfilled Hr, Mn, Sc and t (*tenths*-of-second) fields default to zero. (The two inner + signs can't be changed.)

In this example, press [TAB] once, type 03—or *space* 3 or 3 *space*—and press [RTN]:

```
Adjust (N)  + 03+Mn+Sc.t
```

```
MON 02/14/83 04:17:19 PM
```

The adjusted TIME display.

Calibrating the Clock (EXACT)

The EXACT command is used to calibrate the clock. The speed of the clock may be adjusted within a range of ±10%.

Note: Clock accuracy is affected by the physical environment of the computer. Refer to appendix B, Owner's Information, for operating specifications.

The following sample procedure corrects the clock speed for an excess of 5 seconds a week. As the first step, use the SET or ADJST procedure to synchronize the TIME display with an accurate source. Afterwards, type exact [RTN] in the TIME command field:

```
MON 02/14/1983 01:22:47 exact
```

Specifying that the current TIME display is exact.

```
MON 02/14/1983 01:22:48 PM
```

The setting remains unchanged, but an exact mark has been established.

Check the STATS template now:

```
MON 02/14/1983 01:22:55 PM stats
```

```
Date: MDY, *Time: AM, Appt: YEAR
```

The asterisk before Time signifies that EXACT has been executed at least once.

That's all you do for the next week. Later, you compare the TIME display against the calibrated source. Noticing the clock has gained 5 seconds, you execute another ADJST command, type a minus, tab over to the seconds field, and type *space* 5:

```
Adjust (N)  - Hr+Mn+ 5.t
```

The ADJST template shows -5 seconds correction. 5 *space* or 05 may be typed instead.

```
MON 02/21/1983 06:22:14 PM
```

A relative adjustment of -5 seconds is made.

You may want to repeat the ADJST procedure, using the tenths-of-a-second field to get the TIME display and the calibrated source to agree. Alternately, you may fix the TIME display using the SET command and set-time template.

When the TIME display is synchronized, execute a second EXACT command:

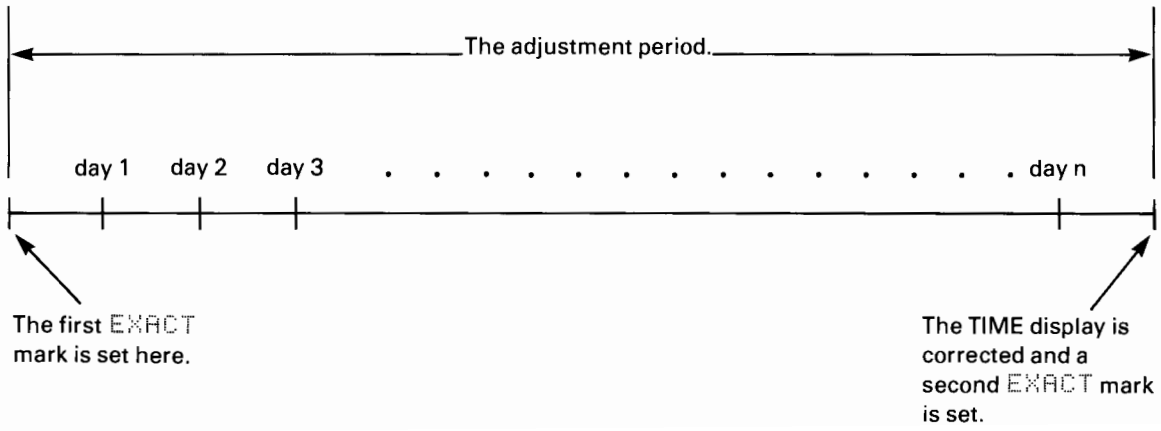
```
MON 02/21/1983 06:25:13 PM exact
```

Entering a second EXACT command.

```
MON 02/21/1983 06:25:14 PM
```

Marks the end of the adjustment period. The clock speed is now corrected for 5 seconds/week.

Clock accuracy is adjusted accordingly:



Here's a summary:

1. Fix the TIME display initially using ADJST or SET.
2. Type exact in the TIME command field and press [RTN].
3. After a period of days, weeks, or months, correct the TIME display, again using ADJST or SET. You may fix the TIME display more than once during the adjustment period.
4. Execute another EXACT command. Doing so creates an adjustment factor that will regulate the clock rate from this point on.

The HP-75 use only the two *most recent* EXACT commands to compute the adjustment factor. However, executing two EXACT commands in a row with no intervening SET or ADJST correction will cause no change in the adjustment factor.

If the amount of the correction is greater than 30 minutes, EXACT uses the time difference by which the correction exceeds the next lower multiple of 30 minutes—a time difference between 0 and 30 minutes.

A correction over a short adjustment period (less than a day) is likely to result in an erroneous clock rate. In other words, the longer the adjustment period between EXACT commands, the better the calibration. If the adjustment computation would be improper, the HP-75 will display warning 70—time adjust bad—when the second EXACT command is executed, leave the adjustment factor at its current value, and accept the EXACT command as the beginning of a new adjustment period.

Resetting the Clock Rate (RESET)

The current adjustment to the clock rate remains in effect until:

- You execute another EXACT command.
- You execute the RESET command by typing reset [RTN] in the TIME command field.

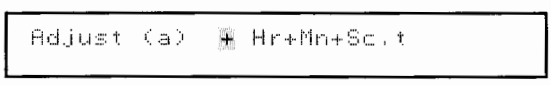
When a RESET command is executed, the EXACT marks and the adjustment factor are cleared. The STATS template will display ~Time again instead of *Time.

Normal and Absolute Adjustments (N, A)

The N (normal) option in the ADJST template causes a certain amount of the adjustment to be used for error correction. The amount is computed so that time zone, daylight savings, and other adjustments that consist of whole hours or half hours will not affect clock speed. However, "leftover" amounts of time—up to 15 minutes per adjustment—will be considered as error correction if the ADJST template is used as part of a clock calibration procedure.

The A (absolute) option in the ADJST template means that all of the adjustment will be considered a time zone change and none will be used for error correction.

To make an absolute adjustment, type adjst [RTN] in the TIME command field. Then press [←] or [BACK] and type an a or A in the ADJST template:



Changing the ADJST template for an absolute adjustment.

Absolute adjustments are seldom necessary; use the A template to make small corrections without affecting clock speed.

Clock Functions (DATE, TIME, DATE$, TIME$)

The HP-75 clock reading can be accessed in EDIT mode by means of four programmable functions—DATE, TIME, DATE$, and TIME$.

The DATE function returns an integer representing the current date in a *yyddd* format.

Example:

```
>date
```

```
83038
```

The numeric representation of February 7, 1983.

The leftmost two digits indicate the year number of the current century. The rightmost three digits indicate the day number of the current year; for example, January 1st is indicated by 001.

The TIME function returns the number of seconds that have elapsed since the most recent midnight, a value ranging from 0 (at midnight) through 86399.999 (at 11:59 PM and 59.999 seconds). TIME values are rounded to milliseconds.

Example:

```
>time
```

```
49272.555
```

The numeric representation of 01:41:12 PM and .555 seconds.

The TIME function is useful for checking computing times. The difference between the value of TIME before and after a program segment is the number of seconds taken by that segment. Refer to the stopwatch program (page 220) for an example.

The DATE$ function returns an 8-character string in a year/month/day (*yy/mm/dd*) format.

Example:

```
>date$
```

```
83/02/14
```

The string representation of February 14, 1983.

The TIME$ function returns an 8-character string in an hours:minutes:seconds (*hh:mm:ss*) format, where the values range from 00:00:00 through 23:59:59.

Example:

```
>time$
```

```
13:41:52
```

The string representation of 01:41:52 PM.

Section 7

APPT Mode Operations

Contents

Introduction 100
Scheduling Appointments ([APPT]) 100
The appt File ([↑], [↓], CAT APPT, PURGE APPT) 102
Deleting Individual Appointments ([SHIFT] [DEL]) 104
Editing Already Scheduled Appointments ([RTN], [SHIFT] [DEL]) 104
Processing Due Appointments ([RUN]) 105
Turning Off APPT Mode (ALARM OFF, ALARM ON) 106
Appointment Types (N, R, A, [SHIFT] [APPT]) 106
Command Appointments ([←], >) 108
New APPT Templates (STATS) 109
The Extended Calendar (EXTD) 110
Using The Extended Calendar (**) 110
Date/Day Searches 111
Copying Appointments To and From Mass Storage (COPY) 112

7

Introduction

This section shows you how to:

- Schedule appoints to display messages and execute commands.
- Examine and edit the appointments stored in the appt file.
- Acknowledge and process due appointments.
- Schedule a variety of one-time and repeating appointments on two calendars—the year calendar and the extended calendar.
- Turn APPT mode off and on.
- Find day/date matches on the extended calendar.
- Use mass storage to expand the appointment storage capacity of your computer.

Scheduling Appointments ([APPT])

To schedule appointments on the HP-75, enter APPT mode: Press [APPT]. If no appointment is due—that is, if the **APPT** annunciator isn't turned on—the APPT template appears:

```
Day Mo/Dy/Yr Hr:Mn AM #1N !Note
```

The APPT template is used to schedule appointments.

Fill in as much of the APPT template as you need. The HP-75 supplies default values for unused days-of-week, date, and time fields:

APPT Template	Fields	Default Values
day-of-week	Day	The day (SUN—SAT), computed from date information.
date	Mo	The current month or the next possible month before the end of the following year.
	Dy	The current day or the next possible day.
	Yr	The current year if possible; otherwise, the following year. Appointments may be scheduled for anytime within the next 365 (or 366) days. The Yr field is used only when scheduling appointments outside of this range. (Refer also to The Extended Calendar, page 110.)
time	Hr	The current hour or the next hour if a same-day appointment; otherwise, 12 midnight.
	Mn	The following minute if a same-hour appointment; otherwise, 00.
	AM	AM, if any time outside the current day.
alarm type	#1	A type 1 alarm, a short beep.
appointment type	N	A *normal*, or type N appointment that comes due only once.
message or command indicator	!	The start of a message field. (May be changed to > to schedule a command or BASIC statement.)
message or command field	Note	An unfilled Note field disappears from the completed appointment.

In general, unfilled date and time fields default to the earliest *future* values possible.

The hour and AM/PM fields are not totally separate. If you make an entry in the AM/PM field, you must enter a number in the hour field also—the current hour is associated with the current AM/PM value.

Use the alarm field (#1) to set the type of audible alarm you want to accompany the appointment. You have ten types to chose from, 0 through 9.

Digit	Alarm Type
0	The beep is suppressed.
1	A short chirp.
2	A long, low tone.
3	A two-tone pattern, repeated three times.
4	A series of high, insistent tones.
5	A long, low tone followed by a long, high tone.
6	A series of eight siren sounds.
7	A type 2 alarm repeated every 15 seconds.
8	A type 4 alarm repeated every 15 seconds.
9	A type 6 alarm every 15 seconds.

Alarm types 7, 8, and 9 cause the HP-75 to keep beeping at 15-second intervals until you press a key. Because they may discharge the battery pack if they go unattended, you may choose to schedule these three types of alarms only when operating the HP-75 from a power source. Note that a low battery condition will eventually cause the HP-75 to terminate all operations, including appointment functions, and shut itself off. Refer to Low-Battery Safeguards, appendix B.

Executing a BEEP OFF command disables all appointment alarms *except* for alarm types 6 and 9.

The N field is used to specify one of three types of appointments:

Letter	Appointment Type
N	A *normal* or one-time appointment whose APPT annunciator will stay on until you press the [ATTN] key in APPT mode.
R	A self-scheduling, *repeating* appointment that updates itself for the future as soon as it sounds the alarm.
A	A self-scheduling appointment that waits for you to *acknowledge* it (press [ATTN]) before it reschedules.

Refer to Appointment Types, page 106, for more information.

The message/command indicator (!) marks the start of the Note field, which holds up to 68 characters of text or any combination of EDIT mode commands and BASIC statements. Refer also to Command Appointments, page 108.

The [TAB] and [SHIFT] [TAB] keys enable the cursor to skip forward and backward across display fields. Other editing keys (for example, [←] and [SHIFT] [→]) also speed up your entries.

When done typing and reviewing the appointment, press [RTN]. The completed appointment is displayed with the missing values filled in—the way it's stored in memory. To display the unfilled APPT template again, press [CLR]:

```
Day Mo/Dy/Yr Hr:Mn AM #1N !Note
```

Ready to schedule another appointment.

The [↓] key returns the most recently displayed appointment to the display.

One feature of the day-of-week (Day) field is that it allows you to specify weekly and monthly information:

Day-of-week specified:	Appointment is scheduled for:
SUN, MON, ..., SAT	A day in the coming week, if possible; otherwise, the next possible day/date match.
SU+, MO+, ..., SA+	The next day after the current or specified date.
SU1, SU2, SU3, ..., SA3, SA4, SA5	The day that falls in the 1st through 5th week of the current month or next possible month.
SU-, MO-, ..., SA-	The first day occuring before the specified date. If scheduled before today's date, the appointment will "go off" as soon as you press [RTN].

Inappropriate information entered in any APPT mode field may cause an error condition. For example, specifying the wrong day-of-week for a given date will cause error 72—day/date mismatch—to occur when you press [RTN]. The incorrect template will reappear, ready for correction. Pressing [SHIFT] [FET] will display the error message again. Clear the error by pressing [ATTN], [APPT], or [CLR] (or [EDIT] or [TIME]), or use the editing keys to correct the display. Typing errn [RTN] in EDIT mode will return the identification number of the most recent error.

The HP-75 will not accept two identical appointments. At least one character in one of their fields must differ. If two different appointments are scheduled for the same time, the first one to be scheduled will come due first, followed immediately by the alarm for the second appointment.

The appt File ([↑], [↓], CAT APPT, PURGE APPT)

Your appointments are stored in memory as part of the HP-75 appt file. When you press [APPT], you switch the file pointer to the appt file, positioned at the due appointment. If there is at least one due appointment, the one that came due first is displayed (due and past due appointments appear with their day-of-week, date,

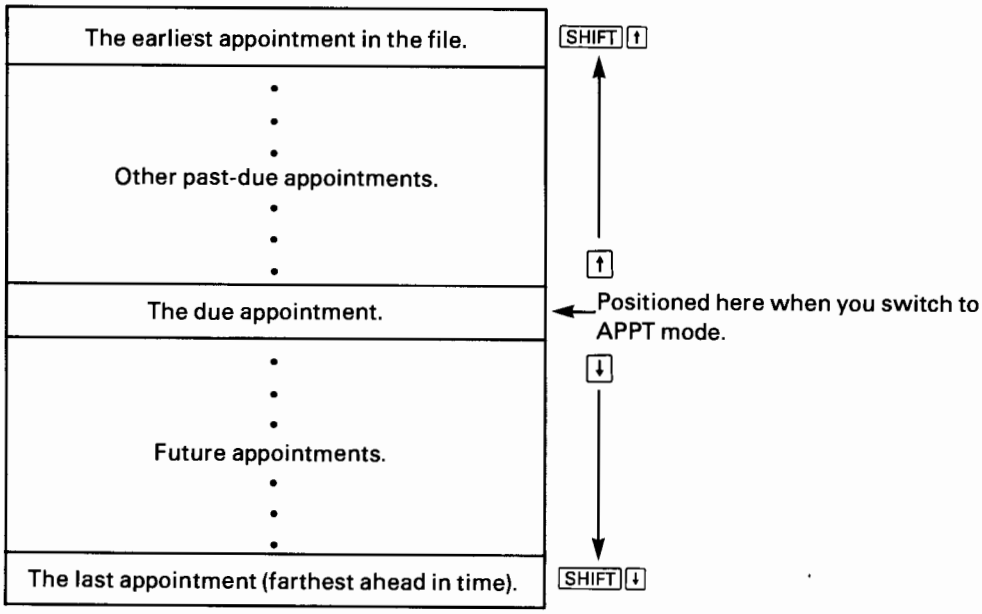
and time fields underlined). If there are no appointments currently due, then the APPT template is displayed.

The [↑] key and [↓] key let you locate individual entries in the appt file. Use [↑] to display earlier appointments and [↓] to display later appointments. When the APPT template is displayed and no appointments are currently due (the **APPT** annunciator is off):

- Pressing [↑] will display past appointments, most recent first.
- Pressing [↓] will display future appointments, nearest due first. If there are no future appointments, then the most recent past appointment will be displayed.

Pressing [CLR] causes the APPT template to reappear.

To display the first appointment in the file, press [SHIFT] [↑]. To display the last appointment in the file, press [SHIFT] [↓].



To display the catalog entry of the appt file, press [EDIT]. Then execute the CAT APPT command:

```
>catappt
```

An EDIT mode command. CAT ALL may be used instead. Spacing and case are unimportant.

Press [RTN]:

```
appt          A    21 09:40 02/05/83
```

A typical appt file catalog entry.

Filename in lowercase to show that the HP-75 creates and maintains the appt file.

The time and date of the file—that is, the system clock reading when you entered the first appointment after a system reset or after destroying a former appt file.

The size in bytes.

The type of file. There can be several appointment files in memory (type A), but only one can be named appt.

Although you can't LIST, EDIT, or RUN an appointment file, you can COPY and PURGE the file.

Example:

```
>purge appt
```

Pressing [RTN] purges the appt file and removes its name from the system catalog.

The appt file may also be renamed with the RENAME command. The appointments in a renamed appointment file will have no effect on the HP-75 unless the file is again renamed to appt. In the meantime, the HP-75 will create a new appt file the next time you schedule an appointment.

If you copy an appointment file to the appt file, all of the appointments from both files are merged into the appt file.

The name of the active appointment file, appt, appears unquoted in all EDIT mode file commands.

Deleting Individual Appointments ([SHIFT] [DEL])

All appointments remain in the appt file until you purge the file or delete the appointments one by one. To delete an individual appointment:

1. Press [APPT] to switch to APPT mode and the appt file.
2. Use [↑] and [↓] to locate the unwanted appointment.
3. When the appointment appears in the display, delete it—press [SHIFT] [DEL].

Afterwards, the following appointment in the appt file becomes the displayed appointment. If you continue to depress [SHIFT] [DEL], the displayed appointment and succeeding appointments are deleted one by one as they are brought to the display.

If all appointments are deleted in this way, the appt file will disappear from the system catalog.

Editing Already Scheduled Appointments ([RTN], [SHIFT] [DEL])

You can *edit*, or change, an already scheduled appointment by displaying it with [↑], or [↓] and then typing new information over it. Pressing [RTN] afterwards stores the new appointment in the appt file *without* removing the original.

[SHIFT] [DEL] by itself deletes a displayed appointment. However, if you type over an appointment in the display and *then* press [SHIFT] [DEL], you *replace* the former appointment with the new appointment.

Example:

```
@AT 02/05/83 09:25 AM #3N !Phone
```

An old appointment to be replaced by a new appointment, say, for Thursday at the same time (assuming today is Tuesday, February 8th).

```
thu 02/  /83 09:25 AM #3N !Phone
```

Type thu in the day-of-week field, tab over the month field, and type two blanks in the day field. The underlining is removed as you type.

Then press [SHIFT] [DEL]:

```
THU 02/10/83 09:25 AM #3N !Phone
```

The new appointment replaces the original.

If an error occurs when you press [RTN] or [SHIFT] [DEL], it probably means that the information in the template specifies a day-of-week, date, or time that's impossible for the HP-75 to schedule. Both past and future appointments may be edited using [RTN] and [SHIFT] [DEL].

You may also *acknowledge* a due appointment by pressing [SHIFT] [DEL] with the due appointment in the display. Doing so will simultaneously acknowledge and delete the appointment.

Processing Due Appointments ([RUN])

Appointments may arrive while the HP-75 is on or off. You saw in section 1 that a due appointment in either case causes the APPT annunciator to turn on and the specified alarm to sound.

If the HP-75 is on, then the due appointment won't interfere with the computer's current operation, whether it's running a program or displaying the time. The APPT annunciator stays on as long as the display stays on to indicate that the due appointment is waiting.

Whether the HP-75 is on or off, if more than one appointment arrives before the others are acknowledged, then all due appointments wait for you to acknowledge them one by one with [ATTN] or [SHIFT] [DEL].

If a due appointment includes a message or command, you can display that message or execute that command by pressing [RUN] in APPT mode. Pressing [RUN] at *any* time in APPT mode causes any due appointment to be processed.

Example:

TUE 02/08/83 3:00 PM #5N !Call APPT	A due appointment with a message. Press [RUN] to process it.
!Call home. APPT	The message is displayed in EDIT mode for approximately 5 seconds or until you press a key.
TUE 02/08/83 3:00 PM #5N !Call APPT	If you haven't pressed a key after 5 seconds, the due appointment returns to the display and waits for you to acknowledge it.

Due appointments can be processed only *once*. (Pressing [RUN] at any other time in APPT mode works the same as pressing the [APPT] key.)

When an appointment comes due while the HP-75 is off, then the HP-75 processes the appointment immediately! The APPT annunciator turns on, the alarm sounds, the HP-75 turns on in EDIT mode, and the appointment message is displayed or its command executed. After about 5 seconds the HP-75 turns itself off.

The HP-75 doesn't allow due appointments to interrupt keyboard operations. If a due appointment with a message or command remains unprocessed at the time the HP-75 turns off, then the HP-75 processes the appointment as it would while off. For example, if you press [SHIFT] [ATTN] or execute BYE to turn off the HP-75, then any due appointments, whether acknowledged or unacknowledged, will be processed immediately.

Similarly, due appointments won't interrupt program execution, even when program execution has been suspended—however, the alarm does sound. If the BYE command is executed from a *program*, the HP-75 shuts off without causing past due appointments to be processed. While off in this condition, the HP-75 will not respond to new due appointments because the program hasn't completed. When the computer is turned on again with the [ATTN] key, the program finishes executing. Past due appointments are processed when you next turn off the HP-75.

Turning Off APPT Mode (ALARM OFF, ALARM ON)

The ALARM OFF command prevents future appointments from coming due. To turn off APPT mode, press [EDIT] and type:

```
>alarm off
```

Pressing [RTN] causes the HP-75 to ignore future appointments.

As long as the ALARM OFF setting is in effect, newly arrived appointments cause neither the **APPT** annunciator to turn on nor the alarm to sound. Neither will the HP-75 display their messages or execute their commands. However, the HP-75 *will* process any past-due appointments and let you schedule new appointments.

> **Important:** Set the HP-75 to ALARM OFF if you intend to remove the battery pack while the HP-75 isn't connected to a power outlet. Otherwise, a due appointment will cause a machine reset if the HP-75 attempts to turn on without a source of power.

To return the HP-75 to the way it normally handles appointments, execute the ALARM ON command in EDIT mode. Doing so will cause all past due appointments since the last ALARM OFF command to turn on the **APPT** annunciator and sound their alarms, ready to be acknowledged and processed.

Appointment Types (N, R, A, [SHIFT] [APPT])

Using the N field of the APPT template, you can set three types of appointments, N, R, and A.

```
Day Mo/Dy/Yr Hr:Mn AM #1N !Note
```

The field that specifies the type of appointment.

Here's a comparison of their functions:

Type of Appointment:	N	R	A
Abbreviation for:	Normal.	Reschedules immediately.	Waits for acknowledgement before rescheduling.
Turns on the APPT annunciator?	Yes	Briefly	Yes
Sounds the beep?	Yes	Yes	Yes
Needs acknowledgement (the [ATTN] key)?	Yes	No	Yes
Updated for the future?	No	Yes	Yes, when acknowledged.

A 0 in the alarm field suppresses the alarm. A 7, 8, or 9 causes the alarm to sound repeatedly until you press a key.

To set a self-scheduling appointment, type an r or a in the appointment type (N) field.

Example:

```
Sun Mo/Dy/Yr  7:55 pM #4r !Nova
```

Setting a self-scheduling appointment by specifying an R.

```
Rept=Mo+Dy+Hr+Mn | DOW
```

The Rep t template appears.

If you want to cancel the procedure, press [ATTN] or [APPT]. (Error 76—bad rep field—will occur when you cancel a repeat procedure.) Pressing [CLR] clears the Rep t template.

What you put in the Rep t template sets the date and time when the appointment will come due again.

Examples:

Rep t=Mo+07+Hr+Mn I DOW	Sets the appointment to reschedule itself every week. Unfilled Rep t fields default to zero.
Rep t=Mo+Dy+02+Mn I DOW	Sets the appointment to reschedule itself every two hours.
Rep t=Mo+Dy+Hr+15 I DOW	Sets the appointment to reschedule itself every fifteen minutes.

Values in the fields may range from 00 through 99. However, the time fields won't assume negative values. When you're done with the Rep t template, press [RTN].

Example:

Rep t=Mo+ 7+Hr+Mn I DOW	SUN 02/13/83 07:55 PM #4R !Nova
Specifies a weekly alarm.	The completed repeating appointment.

To check the repeat interval of an R or A appointment, press [SHIFT] [APPT] and keep the [APPT] key held down.

Example:

Yr=1983 I Rep t=Mn+07+Hr+Mn I DOW	[SHIFT] [APPT] temporarily displays the year and repeat interval of the displayed R or A appointment.

When a type R appointment comes due, the alarm sounds and then the appointment is updated in the appt file as specified by its repeat interval. To update a type A appointment, you acknowledge the appointment—that is, press [ATTN] with the due appointment in the display. A type A appointment is rescheduled based on the time and date of the appointment—not of the acknowledgement. Both R and A appointments will continue rescheduling indefinitely—delete them with [SHIFT] [DEL].

When you examine the appt file, you'll find that the HP-75 stores just the current or updated form of a repeating appointment.

The DOW field of the Rep t template enables you to set repeat intervals for day-of-week occurrences.

Entering SUN, MON, ..., SAT causes the appointment to reschedule itself for future occurrences of the specified day-of-week.

Example:

Rep t=Mo+Dy+Hr+Mn I mon	The repeating appointment will be rescheduled for each Monday.

Entering SU+, MO+, ..., SA+ in the day-of-week field causes the same repeat intervals as SUN, MON, ..., SAT.

Entering SU−, MO−, ..., SA− causes the appointment to reschedule for the first day-of-week *prior* to the interval specified by the month and day repeat fields.

Example:

```
Rept= 1+Dy+Hr+Mn ! we
```

Sets the appointment to reschedule for the nearest Wednesday that occurs before the one-month repeat interval.

Entering SU1, SU2, SU3, ..., SA3, SA4, SA5, causes the appointment to reschedule for the specified day-of-week and specified occurrence (first through fifth) of the current month (if possible) and of succeeding months.

Example:

```
Rept=Mo+Dy+Hr+Mn ! th1
```

Sets the appointment to repeat every first Thursday of succeeding months.

A combination of Rept template fields may be used to specify a variety of repeat intervals. Example:

```
Rept= 1+ 7+Hr+Mn ! fr2
```

Sets the appointment to repeat the second Friday of every other month. (The extra 7 days causes the rescheduling to go to the next month.)

In all cases, pressing [RTN] places the completed appointment in the appt file. Pressing [SHIFT] [APPT] temporarily displays the year and repeat interval of the current appointment.

Command Appointments ([←], [>])

You can schedule appointments to execute EDIT mode commands, BASIC statements, and calculator expressions.* To do so, change the Note field into a command field by replacing the ! mark with a BASIC prompt (>).

```
Day Mo/Dy/Yr Hr:Mn AM #1N >Note
```

Press [←] to backspace over the !. Then press [SHIFT] [.] to type the BASIC prompt.

Then type a command, statement, or calculator expression in the newly created command field and press [RTN] to schedule the appointment.

Example:

```
ay Mo/Dy/Yr Hr:Mn AM #1N >plist
```

Entering the PLIST command in the command field.

```
TUE 02/08/83 03:15 #1N >plist
```

Default values are supplied, and the command appointment is scheduled.

Individual appointments can hold multiple commands and statements. Use the @ symbol to concatenate the commands and statement.

Example:

```
Hr:Mn AM #1N >pwidth 24 @ plist
```

This appointment includes both a PWIDTH and a PLIST command.

* A BASIC statement must be executable from the keyboard to make sense in a command appointment.

The command field is stored in the appt field just the way you've typed it—your spacing, lowercase letters, and abbreviations are not altered by the HP-75.

The HP-75 processes due appointments with command fields the same way it processes due appointments with message fields. A due appointment won't interfere with the operation of the machine while the HP-75 is turned on. Instead, a command field of a due appointment will be executed at one of five times:

- When you press [RUN] in APPT mode.
- When you press [SHIFT] [ATTN] in any mode.
- When you execute a bye command from the keyboard.
- When the HP-75 turns off after 5 minutes of idleness, assuming a STANDBY OFF setting.
- When an appointment comes due while the HP-75 is already turned off.

In all but the first case above, the HP-75 will shut itself off after the appointment is processed. Consequently, a STANDBY ON command entered in the command field will not keep the machine turned on, but it will set STANDBY ON.

All three appointment types—N, R, and A—may contain command fields. For example, assume there's a short program in memory named REVEILLE that plays a reveille tune. Here's the way to schedule this program to run every morning at 7:00:

```
Day Mo/Dy/Yr 07:Mn AM #0r >Note
```

Setting a repeating appointment (type R). The zero will suppress the beep alarm.

Then type the RUN command followed by the appropriate filename:

```
Yr 07:Mn AM #0r >run 'reveille'
```

Scheduling the program REVEILLE to run at 7:00 AM.

Notice that the filename takes quote marks. Pressing [RTN] causes the Rep t template to appear. It's filled in, and then [RTN] is pressed a second time:

```
Rep t=Mo+01+Hr+Mn | DOW
```

Setting the program for a daily occurrence.

```
WED 02/09/83 07:00 AM #0R >run '
```

The completed appointment is displayed, scheduled for the next occurrence of 7:00 AM.

New APPT Templates (STATS)

The STATS command—a TIME mode command— controls the format of both TIME and APPT displays. Typing stats [RTN] in the TIME command field produces the STATS template:

```
TUE 02/08/1983 04:11:23 PM stats
```

TIME template with the stats command.

```
Date: MDY, ~Time: AM, Appt: YEAR
```

The STATS template appears.

To specify a day\month\year format, type dm. To specify a 24-hour time format, type **:

```
Date: dmY, ~Time: **, Appt: YEAR
```

Specifying new date and time formats. The cursor advances to the proper fields.

```
TUE 08\02\1983 16:12:52 **
```

The modified TIME display.

The STATS template controls the way your appointments are entered and displayed in APPT mode. Press [APPT]:

```
Day Dy\Mo\Yr Hr:Mn ** #1N !Note
```

For new appointments, fill in the date and time fields according to the new format.

If using 24-hour notation, you have a choice in entering the time of a new appointment. You can:

- Enter the time as a number between 00:00 and 23:59. For example, 5:30 PM is entered as 17:30. (With an AM/PM format, 24-hour notation may be used by typing ** in the AM field.)
- Enter the time in normal AM/PM form. For example, specify 5:30 PM by typing 05, 30, and pm. The HP-75 converts the time to 24-hour notation.

Existing appointments in the appt file will be displayed according to the current STATS setting.

> **Note:** When you SET the clock in TIME mode, your appointments maintain their original times and dates. If a new clock setting causes any appointments to come due, they'll do just that.

The Extended Calendar (EXTD)

After a system reset occurs, the HP-75 is set to a year calendar; appointments can be scheduled for anytime within the next 365 (or 366) days. When you enter EXTD in the YEAR field of the STATS template, you change the APPT mode calendar to an *extended* calendar so that appointments can be scheduled from the year 0000 through the year 9999. Press [TIME], type stats [RTN], type extd in the YEAR field, and press [RTN]:

```
Date: mdy, ~Time: am, Appt: extd
```

The STATS template, specifying date and time formats as well as the extended appointment calendar.

```
TUE 02/08/1983 04:20:56 PM
```

The revised TIME display.

Then press [APPT]: The APPT template looks the same, but the Yr field now assumes special importance.

Using The Extended Calendar (**)

In filling out the APPT template, you'll notice that the cursor no longer skips the year (Yr) field. You can use any two digits to specify any year of the current century.

Example:

```
Day 01/17/84 Hr:Mn AM #1N !Note
```

Filling in Yr information.

```
TUE 01/07/84 12:00 AM #1N
```

Sets an appointment for 1984.

Default values for the time fields are 12:00 AM (midnight).

Skipping the Yr field means that the appointment is supplied with current year values, if possible. If the specified date doesn't exist in the current year, then the HP-75 searches ahead in time for the desired match.

To check the year of any appointment in the display, press [SHIFT] [APPT]:

```
Yr=1984
```

[SHIFT] [APPT] displays the year of the current appointment and its repeat interval, if type R or A.

The EXTD option enables you to schedule appointments for times *past*. To find the day-of-week of your birth, supply the date information.

Example:

```
Day 04/03/50 Hr:Mn AM #1N !Note
```

Specifying a date in 1950.

```
MON 04/03/50 12:00 AM #1N
                          APPT
```

The day-of-week (Monday) is computed. The appointment comes due immediately because 1950 is decidedly past.

To schedule an appointment outside of the current *century*, type a double-asterisk (**) in the Yr field.

Example:

```
Day 05/14/** Hr:Mn AM #1N !Note
```

A double-asterisk (**) typed in the Yr field gives you access to a hundred centuries.

```
Year? YYYY
```

The HP-75 responds with the Year? template, asking: What year is the appointment for?

Type in four digits (0000 through 9999) and press [RTN].

Example:

```
Year? 2001
```

Specifying an appointment for 2001.

```
MON 05/14/01 12:00 AM #1N
```

The day-of-week and default values are supplied.

To check the year of the appointment in the display, press [SHIFT] [APPT]:

```
Yr=2001
```

The example appointment will arrive during 2001.

Date/Day Searches

To find any date or day-of-week on the extended calendar, supply the desired values and let the HP-75 compute the rest.

Examples:

```
Day 12/25/83 Hr:Mn AM #1N !Note
```

Finding the day-of-week of Christmas, 1983.

```
SUN 12/25/83 12:00 AM #1N
```

Christmas, 1983, arrives on a Sunday.

```
Day 1 /7 /** Hr:Mn AM #1N !Note
```

Determining the day when Galileo first noticed three "starlets" near the planet Jupiter.

```
Year? YYYY
```

The HP-75 prompts for the year.

```
Year? 1610
```

The year was 1610.

```
THU 01/07/** 12:00 AM #1N
                          APPT
```

Galileo discovered the moons of Jupiter on a Thursday.

```
tue 11/Dy/84 Hr:Mn AM #1N !Note
```

Finding the day of the first Tuesday in November, 1984.

```
TUE 11/06/84 12:00 AM #1N
```

The first Tuesday falls on November 6th.

```
sun  2/29/Yr Hr:Mn AM #1N !Note
```

Finding the next year in which February 29th falls on a Sunday.

```
SUN 02/29/04 12:00 AM #1N
```

The next such Leap Year is 2004. (Press [SHIFT] [APPT] to confirm.)

The HP-75 uses the Gregorian calendar—adopted in some European countries in 1582 and in the U.K. and American colonies in 1752—for all APPT and TIME mode operations. Consequently, dates before 1582 won't agree with the calendar in use at the time.

To return to the normal YEAR calendar, execute `stats` [RTN] in TIME mode, type the word `year` over the characters EXTD, and press [RTN]. When you check your `appt` file, you'll find that appointments you've scheduled on the extended calendar stay in the `appt` file with their dates intact.

Copying Appointments To and From Mass Storage (COPY)

The `appt` file can be copied to and from a mass storage medium. For example to copy the `appt` file to a magnetic card or cards, press [EDIT] and execute:

```
>copy appt to card
```

```
Copy to card:  Align & [RTN]
```

Begins the COPY operation.

The HP-75 guides you through the process. When the prompt and cursor reappear, the operation is completed.

To copy appointments from magnetic cards to memory, type:

```
>copy card to appt
```

To copy prerecorded appointments to the `appt` file in memory.

```
Copy from card:  Align & [RTN]
```

This operation will create an `appt` file if one doesn't exist.

The HP-75 *merges* those appointment on cards with those already in the current `appt` file so that all appointments are sorted chronologically. If a pair of duplicate appointments exist—identical in all respects—the HP-75 will ignore the card appointment while merging the two files. (Refer to Copying Files To and From Mass Storage, section 9, page 135, for procedures to use with other mass storage devices.)

Card Reader Operations

8

Contents

Introduction 114
HP-75 Magnetic Cards 114
Using the Card Reader 115
Specifying Card Files 116
Cataloging a Card (CAT CARD) 117
From Memory to Card (COPY TO CARD) 118
From Card to Memory (COPY CARD) 119
Protecting Card Files (PROTECT, UNPROTECT) 121
The Three P's of Protection 122

Introduction

The HP-75 card reader enables you to transfer text files, BASIC files, appointment files, key files, LEX files, and interchange files between memory and magnetic cards. A file may move in either of two directions:

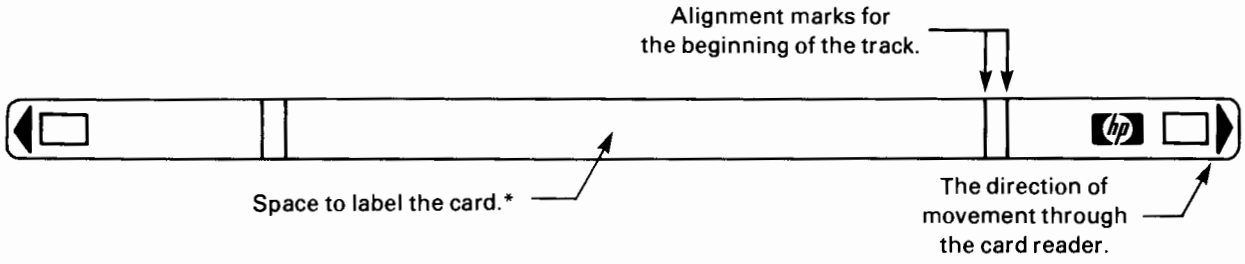
memory ⟶ ***card*** ***card*** ⟶ ***memory***

Regardless of direction, a file transfer results in duplicate files—one in memory and one on a magnetic card or cards.

HP-75 Magnetic Cards

Each magnetic card has two data *tracks*, both of which record the following information:

- The catalog entry of the file recorded on the track.
- The total number of tracks in the file.
- The identification number of *this* track, a number from 1 to the total number of tracks.
- The password, if any. (Passwords cannot be displayed.)
- The write-protection status of the track; that is, whether the track is protected against recording.
- Up to 650 bytes of the file itself. One track may contain information from one file only. One *card* may contain information from one or two files.



* Refer to Marking Magnetic Cards in appendix B, for more information about labeling magnetic cards.

When passing either track of a card through the card reader, always have the printed face of the card up. The order of tracks doesn't matter.

> Important: Keep magnetic cards clean and free of oil, grease, and dirt, and handle cards by their edges only. Dirt and fingerprints degrade the performance of the card reader, cause warning messages to occur, and decrease the lifespan of cards. Cards may be cleaned with isopropyl alcohol and a soft cloth. Keep cards away from sources of strong magnetic fields, such as permanent magnets, wires carrying heavy currents, power transformers, and degaussers (magnetic erasers); magnetism may permanently damage the cards. Appendix B, Owner's Information, includes more card care information.

Using the Card Reader

The HP-75 displays a variety of messages during a card reader operation. When a message exceeds one display line, the first line of the message will appear according to the current DELAY rate. To display again the first line of a card reader message, press [SHIFT] [FET] and hold down the [FET] key.

All card reader operations involve the following steps:

First, type a card reader command in EDIT mode (for example, COPY TO CARD), with either the BASIC prompt (>) or the text prompt (:) present. You may use any combination of lowercase or uppercase letters.

Press [RTN] to initiate the operation. The HP-75 responds with the appropriate message, and waits.

Example:

```
2 track(s) needed
```

A similar message will appear whenever you copy a file to a card. It's displayed for the current DELAY rate and may be recalled temporarily with [SHIFT] [FET].

```
Copy to card:  Align & [RTN]
```

Then the HP-75 waits for your response. You can cancel the operation here by pressing [ATTN].

With the card oriented in the forward direction of the desired track, insert the card so that the rightmost alignment mark is just beneath the entry slot; the card should protrude past the exit slot so that the arrow and box show. Then press [RTN] a second time. The HP-75 responds:

```
Pull card ...
```

Pull the card through the card reader. The HP-75 allows about 5 seconds for you to start pulling the card. A longer time causes the HP-75 to beep, display warning 23—bad read/write—and prompt you to try again. If you decide *not* to pull the card, wait until the HP-75 again prompts you to Align & [RTN] (about 5 seconds). Then press [ATTN] instead.

After you've pulled a card through the reader, several HP-75 responses are possible:

- If you're copying a file in memory to a card, the HP-75 will prompt you for a second pass of the same track:

```
Verify Card:  Align & [RTN]
```

  This time through, the accuracy of the information copied to the track will be verified. If the information doesn't verify on the second pass, the HP-75 will display warning 21—verify

failed—and require two more passes of the same track through the card reader, once to copy and once to verify. (If the track still fails to verify, either clean the card or use a new card.)

- If the file fills more than one track of a card, then the HP-75 will signal you when you are done with the first track:

`Track 1 done; insert track 2`	This message is displayed for the current DELAY rate, then...
`Copy to card: Align & [RTN]`	Ready for you to turn the card around or to insert another card.

- If you pull a card too fast or too slowly, the HP-75 will display warning 24—`pulled too fast`—or warning 25—`pulled too slow`—and prompt for another pull. If you pull a card *very* slowly, the HP-75 may respond as if no card has been pulled.
- The HP-75 will continue its promptings for as many passes as needed. When the operation is completed, the EDIT mode prompt and the cursor will reappear in the display.

There are five HP-75 card reader commands. In their simplest forms, they are:

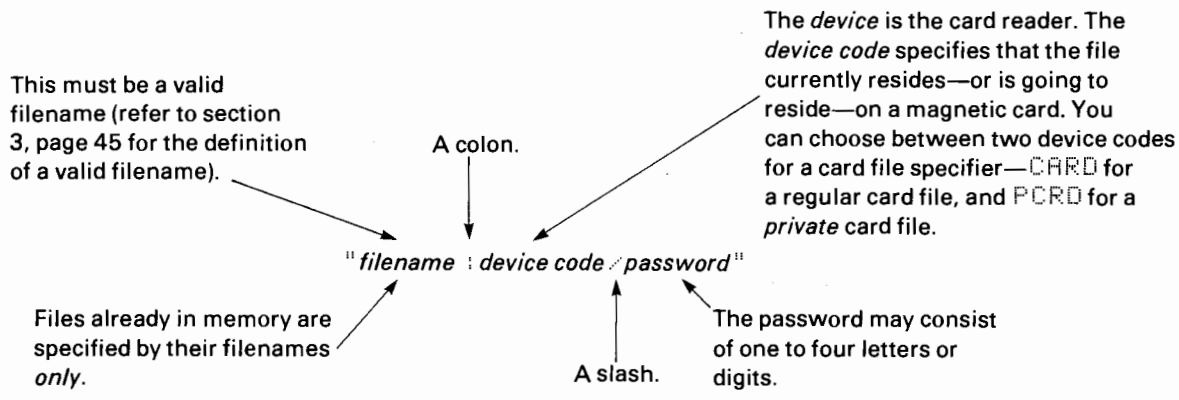
`CAT CARD`	For displaying the catalog entry of a card file.
`COPY TO CARD`	For copying the current file in memory to cards.
`COPY CARD TO '`*filename*`'`	For copying a card file to memory.
`PROTECT`	For protecting an individual track from being overwritten.
`UNPROTECT`	For removing the write-protection from a track.

After a card reader operation, the file pointer will be positioned in the same BASIC or text file at the same line where it was positioned *before* the operation. This is important when you want to edit an incoming file because you'll have to execute an EDIT command after copying the card file.

Programming Note: All five card reader commands are programmable. During program execution, a card reader command initiates the specified operation. After the card reader operation, program execution continues at the following statement of the program. (Copying the current running file is an exception—in this case all programs are deallocated, the file copied, and execution halted.)

Specifying Card Files

You have several options in *specifying*, or naming, a given card file. The simplest is using the unquoted keyword CARD in a command to specify whatever file is on the track at hand. A card *file specifier* is a quoted string of characters that refers to a *specific* card file. A card file specifier includes a *filename*, a *device code*, and optionally, a *password*.



The entire file specifier must be quoted (' ' or " "). The lowercase letters of a file specifier—including the password—are interpreted as uppercase letters.

Specifying PCRD means that the resulting private card file may be copied back to memory like any other card file, but that once returned to memory the file is private—that is, *the file may not be examined or edited in any way*. Only BASIC files, or programs, may be made private. Private BASIC files are indicated by PB in the system catalog; you may RUN, PURGE, or CALL private files but not LIST, PLIST, EDIT, COPY, or even RENAME them. The PCRD option prevents users from viewing, modifying, and duplicating prerecorded programs.

Specifying a password when copying a file in memory to a card means that the file will be "tagged" with the password. Copying a file with password *from* a card requires that you include the password in the file specifier; otherwise, error 66—invalid password—will occur and the copy operation will be cancelled. The password option limits access privilege to card files to those who know the password.

Examples:

'phone:card'	Specifies the card file PHONE. Note that 'phone' used alone would specify the file PHONE in *memory*.
'bywcal:pcrd'	Specifies the card file BYWCAL, a private card file.
'Monmtng:card/11'	Specifies the card file MONMTNG secured by password 11.
'A10352:pcrd/sky'	Specifies the private card BASIC file A10352 that is secured by password SKY. This is the highest level of security.

Note that in order to specify any card file with a password, you must also include a device code, :CARD or :PCRD; for example, 'MONMTNG:CARD/11'.

Advanced User's Notes: You can use suitable string variables or expressions as the parameters of card reader commands. For example, if string variable A$ is assigned 'raan:card/ry', then A$ can be used as the file specifier for the card file RAAN secured by password RY. (However, the *unquoted* keyword CARD may not be specified by a string variable.) Also, file specifiers may be terminated by blanks; for example, 'Ohms:card revision A' specifies the card file OHMS.

Cataloging a Card (CAT CARD)

The CAT CARD command enables you to check the catalog entry of any card file.

```
CAT CARD
```

To display the catalog of a prerecorded card—say, the MONEY program that was shipped with your HP-75—type cat card [RTN].

```
>cat card
```

```
Catalog card:  Align & [RTN]
```

The HP-75 response.

Insert any track of the MONEY program into the card reader (we use the first track) and press [RTN]:

```
Pull card ...
```

The HP-75 prompts you.

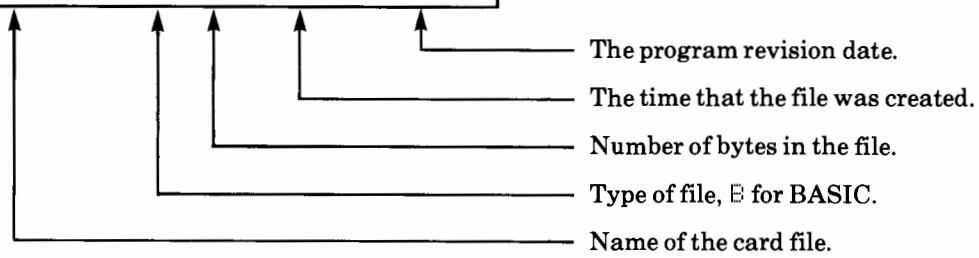
A smooth and steady motion results in the following display:

```
MONEY     (track  1 of  4)
```

The identification number of the current track and the number of tracks belonging to the card file are displayed.

After the DELAY interval, the catalog entry appears:

```
MONEY       B 2205 15:23 04/22/82
```



Press [SHIFT] [FET] to temporarily display the first line. Pressing any other key—except [←], [→], [SHIFT], or [CTL]—causes the prompt and cursor to reappear.

Note that a CAT CARD procedure only *displays* the catalog information of the track. The file itself is not transferred by CAT CARD. If a program file is a private card file—that is, if the file was originally copied to the card with the :PCRD option—then its file type will be indicated by PB. Card passwords are never displayed and need not be supplied in CAT CARD commands.

You can use CAT ':CARD' or CAT ':PCRD' in place of CAT CARD. The three commands function identically. CAT CARD commands are useful for examining the catalog information of unlabelled magnetic cards.

When you catalog a previously unused track, this two line message will be displayed:

```
BLANK     (track  1 of  1)
```

```
BLANK       B     0 00:00 01/01/00
```

The catalog entry shows that this file hasn't been used before.

From Memory to Card (COPY TO CARD)

The most general form of the COPY command is:

```
COPY source TO destination
```

where a card file may be either the *source* or *destination*. To copy a file to magnetic cards, use one of the following forms:

```
COPY  ['filename'      TO CARD
       KEYS                'file specifier'
       APPT                ':CARD'
                           ':PCRD'
      ]
```

Examples:

```
>copy to card
```

The simplest form of the COPY command copies the current BASIC or text file. All information—including the name and time of the source file—is replicated. No security is attached to the card file.

```
>copy to ':pcrd'
```

The source file must be a BASIC file; the destination card file will be protected against listing and editing.

```
>copy to 'accounts:pcrd'
```

The current file will be copied to a private card file. If the destination filename is the same as the source filename, the time and date of the card file will match those of the source file; otherwise, the current time and date will be used for the card file.

```
>copy 'other' to card
```

The specified source file (OTHER) will be copied, while the location of the file pointer won't change. Times and dates will match. No security.

```
>copy 'other' to 'other1:card'
```

Time and date will be new. No security.

```
>copy 'other' to 'other:card/h'
```

Access will be limited by password H. Times and dates will match.

```
>copy appt to 'Jan:card/wry'
```

Copies the appt file under the name JAN, secured by password WRY. New time and date.

```
>copy keys to card
```

Copies the keys file. No security. New time and date.

These examples show that different filenames for sources and destinations result in new times and dates for card files. Only BASIC files—as opposed to text, appointment, keys, and other files—may be private. For example, if LETTER is a text file, then typing `copy 'letter' to ':pcrd'` RTN will cause warning 68—`wrong file type`—to occur. The copy will be allowed to continue, but the resulting card file will be nonprivate. You may secure text, appointment, keys, and other files with *passwords* by using complete file specifiers in the copy-to-card command. Remember that you never use device codes or passwords to specify files in *memory*.

If a file consists of more than 650 bytes, then more than one track will be needed to record it. The HP-75 computes the number of tracks required at the outset of a COPY operation and begins its promptings from track 1. Once you've completed a COPY TO CARD operation, use the CAT CARD command to display the catalog entry from any of the tracks.

From Card to Memory (COPY CARD)

To copy a card file to memory, use one of these forms of the COPY command:

```
COPY   CARD          TO   'filename'
       ':CARD'            APPT
       ':PCRD'            KEYS
       'file specifier'
```

The *source* is the card file to be copied, and the *destination* is the new file in memory as it will be reported in the system catalog. You need to use a complete file specifier *only* when the card file has a password. For other card files, CARD and ':CARD' may be used interchangeably in copy-to-memory commands.

Examples:

```
>copy card to 'eh'
```

Creates file EH in memory and copies the contents of the card file to it (assuming no password). If the card file is also named EH, then the time and date of the new file will match those of the card file; otherwise, the current time and date will be used.

```
>copy ':card' to 'eh'
```

Functions identically. However, only one file in memory may exist named EH.

```
>copy 'accounts:card' to 'a'
```

Determines whether the inserted track belongs to the specified card file (ACCOUNTS). If so, creates file A in memory, with a new time and date; otherwise, warning 26—wrong name—will occur and the HP-75 will prompt for another card. If ACCOUNTS on the card is a private file, then A in memory will be a private file.

```
>copy 'other:card/h' to 'trial'
```

Checks the filename and password of the card file. If correct, creates file TRIAL in memory, with new time and date.

```
>copy 'Jan:card/wry' to appt
```

Merges the appointments from the card into the current appt file.

```
>copy card to keys
```

Creates a new keys file that redefines the keyboard as soon as the copy is completed. (There may be only one file named keys in memory.) Refer also to section 10.

```
>copy ':pcrd' to 'runonly'
```

Creates file RUNONLY in memory and copies the card file to it.

The last example shows the use of ':PCRD' to specify an incoming card file. This form of the COPY command results in a private file in memory, even though the card file itself may not be a private file. The card file specified by ':PCRD' should be a BASIC file, or else warning 68—wrong file type—will occur; however, the COPY operation will be allowed to continue and will result in a nonprivate file.

Specifying a filename for the destination that is different from the card filename will result in a new time and date in the system catalog.

When a track is pulled through the card reader during a copy-to-memory operation, the HP-75 determines whether memory is available for all of the file; if not, the HP-75 displays error 16—not enough memory—and cancels the copy operation.

All tracks from the card file must be copied for the file to be reported in the system catalog. This means that if a copy-to-memory operation is interrupted, none of the card file will remain in memory. If you copy the same file from memory to two different sets of cards, then the tracks from both sets of cards may be used interchangeably when copying the file back to memory. However, note that the two sets of cards must be *identical* copies of the same file with identical catalog entires; otherwise, warning 20—`not this file`—will occur if the tracks are mixed during a copy-to-memory operation.

Protecting Card Files (PROTECT, UNPROTECT)

The card reader PROTECT option enables you to protect card files from being erased (i.e., overwritten).

```
PROTECT
```

One PROTECT command is executed for each *track* to be protected:

```
>protect
```

```
Protect card:  Align & [RTN]
```

Initiates the operation. The next track you pull through the card reader will be write-protected.

After the track has been pulled through the card reader, the prompt and cursor reappear; the track is then protected from subsequent copy-to-card operations. PROTECT can be executed at any time but is normally executed just after a copy-to-card operation.

A separate PROTECT command is necessary for each track you wish to protect. The following program enables you to write-protect a series of tracks, one after another:

```
>10 PROTECT @ GOTO 10
```

Press [RUN] or type `run` to execute this program when you have several tracks to write-protect. The HP-75 will continue executing the PROTECT command until you interrupt the program with [ATTN].

The UNPROTECT command enables you to remove the write-protection mark from a track heading.

```
>unprotect
```

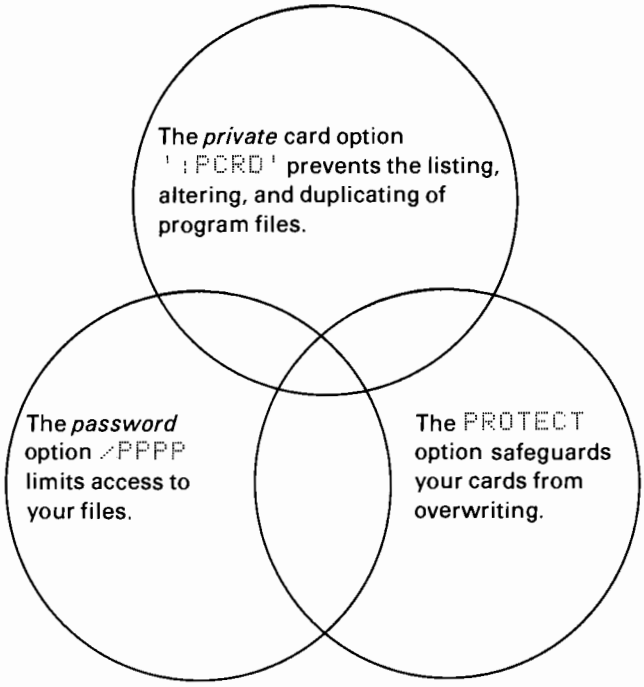
```
Unprotect card:  Align & [RTN]
```

The next track through the card reader will have its write-protection removed.

An UNPROTECT operation makes a track available again for recording information. Note that the HP-75 has no explicit card erasure instruction: A prerecorded track will be erased when you copy another file over it.

The Three P's of Protection

You've seen three ways to secure your card files:



These options may be exercised separately or may be combined for a variety of file protection. However, these capabilities are intended to prevent accidental alteration or reproduction of important files rather than to guarantee absolute security.

Section 9

HP-IL Operations

Contents

Introduction 124
Connecting the Hewlett-Packard Interface Loop 125
Assigning Device Codes (ASSIGN IO) 126
Listing Device Assignments (LIST IO) 127
Declaring Display and Printer Devices (DISPLAY IS, PRINTER IS) 128
Displaying and Printing Information (DISP, PRINT) 129
Turning the Loop Off and On (OFF IO, RESTORE IO) 130
Transmission Interruptions ([ATTN], RESTORE IO, STANDBY OFF, STANDBY ON) 130
Clearing Devices (CLEAR LOOP) 131
Mass Storage Operations 132
The Mass Storage Device (ASSIGN IO) 132
Preparing the Mass Storage Medium (INITIALIZE) 132
Mass Storage Files 133
Cataloging the Medium (CAT) 134
Copying Files To and From Mass Storage (COPY) 135
Duplicating Files on Mass Storage (COPY) 136
Renaming Mass Storage Files (RENAME) 137
Purging Mass Storage Files (PURGE) 137
Packing the Mass Storage Medium (PACK) 137
Advanced User's Information 138
Special Characters 138
Escape Codes ([CTL][BACK], CHR$(27), [CTL][↑], [CTL][↓]) 138
Display Escape Codes 139
End-of-Line Sequences (ENDLINE) 140
Programming with HP-IL 140

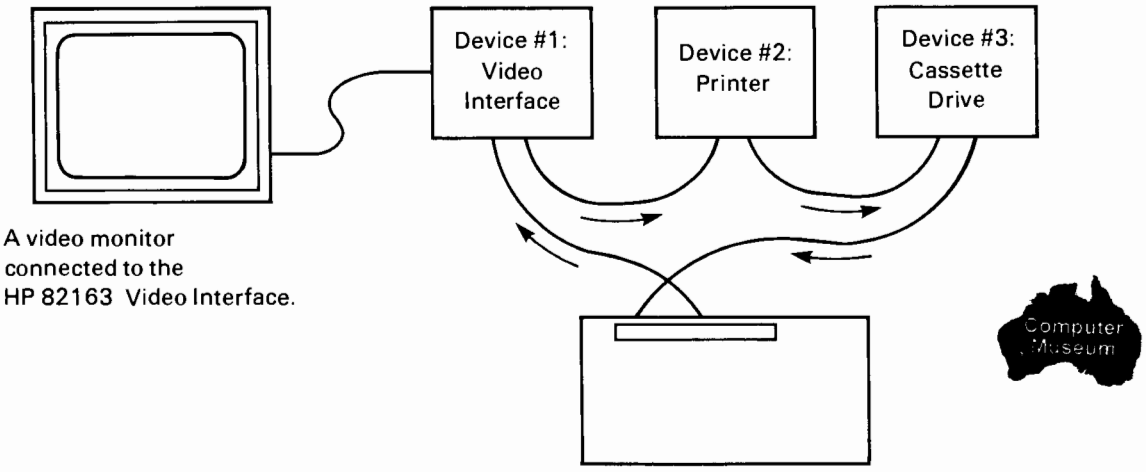
9

Introduction

By itself, the HP-75 uses the keyboard as an *input* device, the display window and beeper as *output* devices, and the card reader as both an input and output device. The Hewlett-Packard Interface Loop capabilities of the HP-75 enable you to extend the control of the computer to a variety of external I/O devices, as many as 30 in one system, including video interfaces, printers, and tape cassette drives.

Throughout this section, HP-IL operations are illustrated using the HP-75 as controller, the HP 82163 Video Interface as display device, the HP 82162A Thermal Printer as printer device, and the HP 82161A Digital Cassette Drive as mass storage device. The HP-75 and HP-IL devices are connected in series in a single loop, or communications circuit.

An HP-IL system for displaying, printing, and storing information.



Instructions and data on the Hewlett-Packard Interface Loop originate from the computer and travel from one device to the next around the circuit until that information returns to the HP-75. If the information isn't intended for a particular device, the device merely passes the information on to the next device in the loop. When the information reaches the proper device, that device responds as directed and then passes the information on. Setting up an HP-IL system involves two steps:

1. Physically connecting the devices in the loop.
2. Assigning device codes, one for each device, that enable you to select individual devices for specific functions.

Connecting the Hewlett-Packard Interface Loop

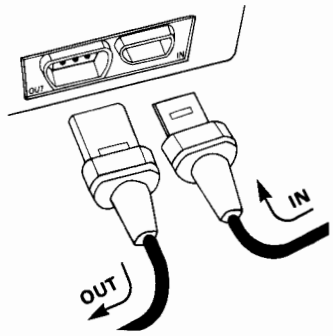
To connect one or more devices to the HP-75:

1. Ready each device according to the installation instructions in the owner's manual for the device.
2. Turn on all HP-IL devices, making certain that each has an adequate power supply. The HP-75 may be turned on or off.
3. Attach the HP-IL cables. Two 1-meter HP-IL cables were included with your HP-75; each device has a matching cable.

All connections are designed to ensure proper orientation.

4. You may connect the devices in any order, but the final configuration is important. The device connected to the OUT receptacle of the HP-75 becomes Device #1. The device connected to the OUT receptable of Device #1 becomes Device #2, and so on. The device that connects to the IN receptacle of the HP-75 is the last device on the loop.

The keying on the receptacles indicates the direction of flow around the loop.



If you wish to space the peripherals farther apart than permitted by their cables, connect additional HP-IL cables in appropriate locations in the loop. The maximum distance between any two devices is 10 meters (33 feet) with standard cable.

All HP-IL commands are programmable; for simplicity, the examples illustrate their use from the keyboard.

> **Note:** To avoid unexpected interruption of HP-IL operation, turn off the peripherals only after the HP-75 is turned off. Turn on all peripherals before turning on the HP-75.

Assigning Device Codes (ASSIGN IO)

Once the devices have been powered and connected, they are ready to be assigned device codes to identify them individually. Press [EDIT] to switch to EDIT mode and type `assignio` [RTN]:

```
>assignio
```

Either prompt (> or :) may be present.

```
3 Device(s) on loop
```

This message shows the number of devices on the loop. It's displayed according to the current DELAY rate.

This message and other HP-IL messages may be displayed again temporarily by pressing [SHIFT] [FET]. The HP-75 prompts for the device code of the first device:

```
Device # 1=':
```

Device #1 is the device connected to the OUT receptacle of the HP-75, in our example, the video interface.

> **Note:** If the HP-75 doesn't respond as indicated and the ASSIGN IO command stays "frozen" in the display, the HP-IL instructions are not travelling properly around the loop. Press [ATTN] to clear the display, make certain that all devices are powered and connected, and try again.

Device codes can be one or two letters, a letter and a digit, or a digit and a letter. Examples of acceptable device codes are `t`, `tv`, `t1`, and `1t`. (A space used as the last character of a device code will be ignored; a space may not be used as the first character.) The letters of device codes are converted internally to uppercase.

Type the first device code and press [RTN]:

```
Device # 1=':tv'
```

Specifying TV as the first device code.

```
Device # 2=':
```

When you press [RTN], the HP-75 prompts for the next device code.*

* You may press any carriage-return/line-feed key to terminate a device code entry (page 43).

Continue assigning device codes until you have named all the devices on the loop; the prompt and cursor will reappear. Device codes are stored in memory until another ASSIGN IO command is executed.

If you enter an incorrect device code, the HP-75 will report an error and will prompt for another name for the same device. If you type nothing for a device and press [RTN], the procedure is ended and only as many devices as device codes will be usable.

From this point, you use the assigned device codes to specify individual devices in HP-IL commands. Device codes must be typed between quote marks (' ' or " "), prefixed by a colon (:).

Another form of the ASSIGN IO command enables you to assign all device codes at once:

```
ASSIGN IO ':device code [,:device code...]'
```

You name the devices in their order on the loop, from #1 through #30.

Example:

```
>assignio ':tv,:p,:ca'
```

Pressing [RTN] assigns device codes TV, P, and CA to three peripherals.

The entire device list is quoted; device codes are prefixed with colons and separated by commas. Device codes are assigned from left to right; for example, TV is assigned to Device #1.

> **Advanced User's Note:** The device list may be specified by a string expression or variable. For example, if D$ is assigned ':tv,:p,:ca', then ASSIGN IO D$ assigns three device codes.

If fewer device codes than devices are declared, then all device codes will be used. If more device codes than devices are declared, then warning 59—too many names—occurs, and as many devices as exist will be assigned. This form of the ASSIGN IO command is useful in programs and key redefinitions.

Attempting to perform most HP-IL operations before assigning device codes causes error 55—ASSIGN IO needed.

Listing Device Assignments (LIST IO)

To check current device assignments, execute the LIST IO command.

```
LIST IO
```

Example:

```
>listio
```

Press [RTN].

```
 3 Device(s) assigned
Device # 1=':TV'
Device # 2=':P '
Device # 3=':CA'
```

The number of assigned devices is displayed at the current DELAY rate, and the device codes are listed one by one.

Pressing [ATTN] stops a LIST IO operation.

Declaring Display and Print Devices (DISPLAY IS, PRINTER IS)

Use the device codes of display devices and printer devices to begin using them to display and print information from the HP-75.

```
DISPLAY IS ':device code [,:device code...]'
```

```
PRINTER IS ':device code [,:device code...]'
```

The DISPLAY IS command causes the specified device or devices to echo everything that you see on the HP-75 display.

Example:

```
>display is ':tv'
```

Pressing [RTN] defines the specified device—the example shows the video interface—to be a display device.

The HP-75 display always functions as a DISPLAY IS device.

The PRINTER IS command causes the specified device or devices to respond to PRINT statements and PLIST commands.

Example:

```
>printer is ':p'
```

Pressing [RTN] declares the specified device to be a printer device.*

The HP-75 display functions as a printer device until another device is declared in a PRINTER IS command.

More than one device can be specified in a DISPLAY IS or PRINTER IS command.

Example:

```
>printer is ':p,:tv'
```

This command declares both devices to be printer devices.

If a single peripheral is declared both a printer and a display device, then it will output both printer and display information. DISPLAY IS and PRINTER IS declarations cause the specified devices to become *listeners*. Refer to your peripheral owner's manual for descriptions of listener responses to HP-IL messages.

> **Advanced User's Note:** If a display or printer device has a buffer, then it outputs information only when the buffer is full or when it's specifically prompted by the HP-75. Pressing one of the carriage-return/line-feed keys (page 43) causes all DISPLAY IS devices to print the contents of their buffers and advance to a new line. Refer to your peripheral owner's manual for buffer information.

* The MODE switch and the PRINT key of the HP 82162A Thermal Printer don't affect the printer's operation while it is connected to the HP-75.

The following commands cancel display and printer assignments.

```
DISPLAY IS *
DISPLAY IS ''
```

```
PRINTER IS *
PRINTER IS ''
```

Executing DISPLAY IS * causes subsequent display information to be directed to the HP-75 display only; executing PRINTER IS * causes subsequent print information to be directed to DISPLAY IS devices.

DISPLAY IS and PRINTER IS declarations remain effective until:

- You execute another ASSIGN IO command.
- You execute another DISPLAY IS or PRINTER IS command.
- You execute an OFF IO command (page 130).
- An interruption in loop transmission occurs (page 130).

Displaying and Printing Information (DISP, PRINT)

Two BASIC statements let you display and print information using HP-IL devices.

```
DISP [display list] [;]
```

```
PRINT [print list] [;]
```

DISP statements are directed to the HP-75 display and current DISPLAY IS devices, while PRINT statements are directed to current PRINTER IS devices.

Example:

```
>print 5555
```

Causes all PRINTER IS devices to print 5555.

If no printer is assigned on the loop, then the HP-75 display and other DISPLAY IS devices perform all printer operations.

You may sometimes wish to display or print information from more than one DISP or PRINT statement without causing a new display or print line to occur. If so, terminate the first DISP or PRINT statement with a semicolon.

Example:

```
>disp 'yes';
```

The semicolon suppresses the carriage return/line feed that normally occurs after a DISP statement.

Subsequent display or print items will be displayed or printed on the same line until a carriage-return/line-feed occurs. The DELAY and WIDTH command regulate the rate and line length of displayed information;

the PWIDTH command regulates the line length of printed information (page 39). Refer also to section 11, page 166, for programming applications of DISP and PRINT statements.

To clear the HP-75 of all previous loop information, execute ASSIGN IO * or ASSIGN IO ''. This will return all output to the display and release any system memory needed by HP-IL devices.

Turning the Loop Off and On (OFF IO, RESTORE IO)

The OFF IO command temporarily suspends communication between the HP-75 and peripherals. Any HP-IL command entered after executing OFF IO will cause error 60—RESTORE IO needed—and the HP-IL command will not be executed. RESTORE IO is the only HP-IL command that can be executed after OFF IO.

```
OFF IO
```

Example:

```
>offio
```

Causes all printer and display information to be directed to the HP-75 display only.

Use the OFF IO command before turning off or disconnecting any devices to preserve current assignments.

The RESTORE IO command restores communication between the HP-75 and peripherals after an OFF IO command.

```
RESTORE IO
```

Before executing RESTORE IO turn on the devices and connect them in their original order on the loop.

Example:

```
>restoreio
```

The devices are activated according to their former assignments.

Since the RESTORE IO command is designed to restore an HP-IL system to its previous configuration, it will not work correctly if the configuration has changed. If the same number of devices are connected but in a different order, or if more than the original number of devices are connected, the HP-75 will not recognize the change and the devices may respond unexpectedly to HP-IL instructions. If fewer devices are connected than device codes in memory, RESTORE IO causes error 59—too many names—and no devices are reassigned. If any error occurs, either reconnect the original devices in their original order and reexecute RESTORE IO, or else execute another ASSIGN IO command to initialize the current HP-IL system.

Be sure that all HP-IL devices are turned on and properly connected whenever the HP-75 might interact with them—for example, when using a PRINTER IS device, accessing a mass storage file, and executing OFF IO, RESTORE IO, and BYE. If the devices aren't ready, the HP-75 waits for a response, as discussed next.

Transmission Interruptions (ATTN , RESTORE IO, STANDBY OFF, STANDBY ON)

Device assignments are maintained when the HP-75 turns off. When the HP-75 turns back on, devices resume their previous functions. However, the HP-75 has no control over the status of the devices in the loop.

If an HP-IL cable becomes disconnected, if a device loses power, or if a device is turned off before an OFF IO command is executed, then communications between the computer and all HP-IL devices will be disrupted. Afterwards, the display may not respond to pressed keys, and error 58—loop timeout—may be displayed. If you intend to intentionally disconnect the loop after turning the HP-75 off, make sure you execute OFF IO *before* you turn off the computer.

After a transmission error, press [ATTN] to interrupt the HP-IL system; error 58—loop timeout—will be displayed if it hasn't been displayed already. Then check the cables, switches, and power supplies to correct the problem. Afterwards, execute your HP-IL command again; or, if the cause of the transmission error was a turned off HP-IL device, execute RESTORE IO. If the HP-75 won't respond to the keyboard, try unplugging then reconnecting one of the HP-IL cables followed by a RESTORE IO command.

During an HP-IL operation involving an external printer or mass storage device, the command that caused the operation will remain in the display while the device is "busy". For example, after you type plist [RTN], the PLIST command will remain in the display until the external printer is ready for another instruction.

If a loop interruption occurs while a STANDBY OFF setting is in effect and an ASSIGN IO or RESTORE IO command is executed, the HP-75 will display loop timeout, after it tries up to 30 times to assign the devices, beeping each time it is unsuccessful. If the loop is reconnected immediately, the command will finish executing normally.

If a properly powered and connected DISPLAY IS or PRINTER IS device takes longer than 10 seconds to display or print a single character (which will not happen in most instances), the HP-75 will time out, execute an OFF IO command, and continue operating without the HP-IL system. If this occurs, check the devices on the loop and execute a RESTORE IO command or any other HP-IL command, to restore device assignments. If you want the HP-75 to wait longer than 10 seconds for a device response, then type standby on [RTN]. A STANDBY ON setting causes the HP-75 to wait indefinitely for a display or printer device to print a character or to wait until you press the [ATTN] key. A STANDBY ON setting also causes the HP-75 to wait indefinitely on device responses to ASSIGN IO and RESTORE IO commands.*

If you get an error during an HP-IL operation, you can generally assume that the operation wasn't completed properly. For example, if HP-IL continuity is disrupted while a file is being copied to a mass storage device, an error is displayed; the file is bad and should be purged.

Clearing Devices (CLEAR LOOP)

The CLEAR LOOP command clears all devices on the Hewlett-Packard Interface Loop. It's a quick way to reset all devices to a ready state.

```
CLEAR LOOP
```

Each device responds to a CLEAR LOOP command according to its design. For example, after a CLEAR LOOP command, the HP 82163A Video Interface clears the display and "homes" the cursor to the upper-left corner; the HP 82162A Thermal Printer positions the carriage to column 1, clears the print buffer,

* An HP-IL device that has a STANDBY mode can be powered up by ASSIGN IO or RESTORE IO in STANDBY OFF mode. The HP-75 takes several seconds to complete the operation, then beeps. But if you attempt to do this when the HP-75 is in STANDBY ON mode, any powered-down device in the loop will cause the HP-75 to halt and wait indefinitely. To clear this condition, close the loop on itself (by connecting an HP-IL cable between the HP-75's OUT and IN plugs), press [ATTN], and wait for the HP-75 to respond. When the prompt reappears in the display, execute STANDBY OFF, reconnect the desired device(s) to the loop, and reexecute ASSIGN IO or RESTORE IO.

and switches to Single Wide and Left-Justify modes; the HP 82161A Digital Cassette Drive rewinds the cassette. For the response of a particular device, refer to the Device Clear message in the peripheral owner's manual.

You can clear individual devices by specifying their device codes in the CLEAR command.

```
CLEAR ':device code [,:device code...]'
```

Example:

```
>clear':p'
```

Pressing [RTN] resets just the P device.

Mass Storage Operations

Mass storage operations store and retrieve information external to the computer and involve three components:

- A mass storage *device*, which may be a card reader, cassette drive, or disc drive.
- A mass storage *medium*, which may be a magnetic card, tape cassette, or disc.
- The interfacing capabilities that allow the computer to operate the mass storage device.

The following pages illustrate mass storage operations using the HP 82161A Digital Cassette Drive as the mass storage *device*; the mass storage *medium* is the tape cassette in the cassette drive; the interfacing capabilities are provided by HP-IL.

Any file in memory that can be copied to a magnetic card can be copied to a mass storage medium. In addition, mass storage files can be cataloged, duplicated, secured with passwords, renamed, and purged.

The Mass Storage Device (ASSIGN IO)

Once a mass storage device has been assigned a device code with the ASSIGN IO command, the device is ready for use. The examples in this section use the device code CA to specify the cassette drive, but many mass storage devices may be used on one HP-IL loop.

Note that requesting a non-mass storage device to perform a mass storage operation results in error 92—dev not mass mem—and no further action. Declaring a mass storage device to be a DISPLAY IS or PRINTER IS device causes the HP-75 to direct display and printer information to the mass storage device (the results will be meaningless); subsequently any attempts to execute mass memory commands will result in error 63—invalid filespec—being displayed.

> **Important:** If an INITIALIZE, COPY from memory to mass storage, RENAME, or PURGE operation is interrupted, the medium may need to be reinitialized. Be certain that all devices on the loop have a dependable source of power, and do not press [ATTN] or disconnect any HP-IL cables while a mass storage device is writing to the medium.

Preparing the Mass Storage Medium (INITIALIZE)

Each mass storage medium must be initialized at least once to establish on the medium a file catalog and a format in which information will be recorded. Execute the INITIALIZE command to prepare a new medium for use.

> **Note:** If the medium has been used before, check the contents of the catalog of the medium before initializing, as described on page 134. Initializing a storage medium completely erases previous information on the medium.

```
INITIALIZE ":device code" [,number of file entries]
```

About 5 minutes are required to initialize a cassette tape in the HP 82161A Digital Cassette Drive.

Each file that you record on the medium requires one entry in the catalog of the medium. If you specify no number in the INITIALIZE command, the catalog will allow a maximum of 128 file entries. You may specify any numeric expression that rounds to an integer value greater than zero. Values less than 1 cause error 11—arg out of range. The most entries allowed for the HP 82161A Digital Cassette Drive is 453. Values greater than 453 cause error 11—arg out of range. Catalog space is reserved in blocks of eight file entries; consequently, the catalog always accommodates a multiple of eight file entries, and at least as many entries as you specify.

An uninitialized mass storage medium will cause error 96—invalid medium—if an attempt is made to read from or write to it. Note that invalid medium may also occur if the medium has been formatted by a computer other than an HP-75, if the medium is worn out or physically damaged, or if the cassette drive is interrupted during initializing.

Mass Storage Files

Information is stored on the mass storage medium as *files*, or collections of lines, that are specified by unique names. Files that may reside on a mass storage medium include:

- BASIC files, indicated by B. (Private BASIC, or PB, files are available through magnetic cards, but not through mass storage media.)
- Text files, indicated by T.
- Appointment files, indicated by A.
- Keys files, also indicated by T.
- LEX files, indicated by L.
- Interchange files, indicated by I.
- Unknown files, indicated by ?. Type ? files may be cataloged, duplicated on the medium, renamed, and purged, but not copied to memory.

You can access individual files on the mass storage medium using one of two file specifiers:

```
filename : device code
filename : device code / password
```

The *filename* is a valid filename (refer to Filenames in section 3). The *device code* is the name that was assigned to the mass storage device during the most recent ASSIGN IO. The *password* limits access to the mass storage file to those who know the password. The password can consist of one to four letters or digits.*

The complete file specifier is quoted. Examples of acceptable mass storage file specifiers are 'data:ca', 'numcalc:ca', and 'seth:ca/123'. (The reserved words keys, appt, and workfile are allowed as filenames in file specifiers; however, they are subsequently treated as ordinary filenames on the mass storage device.)

* Passwords may be assigned to files created by the HP-75. Specifying a password for an interchange file in any mass storage command causes warning 66—invalid password—but the HP-75 allows the command to be executed and ignores the password.

Advanced User's Note: String variables and expressions may be used instead of quoted strings for file specifiers. File specifiers are terminated by the ending quote mark or by the first blank after the device code or password.

Note that a password protects a file from being *copied* and not from being renamed or purged. After the password is correctly specified and the file is copied to memory, the new file in memory is available for editing.

It's possible for other computers to create filenames longer than eight characters. A maximum of 10 characters per filename may be displayed in a storage medium catalog entry; however, the HP-75 matches only the first eight characters in mass storage commands and reports an error if more than eight characters are entered for the filename.

If any file specifier names a nonexistent file on the mass storage medium, then error 62—`file not found`—is reported.

Cataloging the Medium (CAT)

To display the catalog entries of the current medium in the mass storage device, specify the name of the device in the CAT command.

```
CAT ':device code'
```

Example:

```
>cat ':ca'
```

To display the catalog entries of the medium in the CA mass storage device.

When you press [RTN], the catalog heading of the medium is displayed:

```
  Name    Type Len  Time  Date
```

Remains in the display for the current DELAY interval, then...

```
FIRST       B 4096 12:30 02/09/83
```

The catalog entry of the first file to be recorded on the medium is displayed.

Use the [↓] and [↑] keys to display the catalog entries one by one; there will be a brief delay after [↓] or [↑] is pressed as the mass storage medium is repositioned and read. [SHIFT] [↑] displays the first entry in the catalog; [SHIFT] [↓], the last. The [←] and [→] keys with modifiers [SHIFT] and [CTL] are also active while catalog entries are in the display. Pressing any other key cancels the operation and restores the prompt and cursor.

Catalog entries are displayed in the order that they have been recorded on the medium. Before the HP-75 records a new file on the medium, the HP-75 searches the catalog to find the first available location on the medium for all of the file. An initialized but empty storage medium displays only the one-line catalog heading when CAT is executed.

You may also specify an individual mass storage file in the CAT command.

```
CAT 'file specifier'
```

This form of the command displays only the catalog entry of the specified file; no password is necesary. If the HP-75 displays `file not found`, the specified name does not exist in the catalog of the medium.

Copying Files To and From Mass Storage (COPY)

The COPY command enables you to copy one file at a time from memory to a mass storage medium or one file at a time from a mass storage medium to memory. Use one of these forms of the COPY command to copy any nonprivate file in memory to mass storage:

```
COPY TO ':device code'
COPY 'filename' TO ':device code'
COPY TO 'file specifier'
COPY 'filename' TO 'file specifier'
```

Examples:

```
>copy to ':ca'
```

Copies the current file to the CA mass storage device. Names, times, and dates match.

```
>copy 'file2' to ':ca'
```

Copies FILE2 in memory to the mass storage device. Names, times, and dates match.

```
>copy to 'newfile:ca'
```

Copies the current file in memory to NEWFILE on mass storage.

```
>copy 'file3' to 'file4:ca'
```

Copies FILE3 to FILE4 on mass storage. New name, time, and date.

The source and destination files will agree in time and date if their filenames match. If the filenames differ, the resulting file will be dated according to the clock reading when the COPY command is executed.

It may take a minute or more for a file to be copied to or from a mass storage device, depending on the length of the file.

If you specify a password in a copy-to-mass storage command, the resulting file will be secured with the password.

Examples:

```
>copy 'file5' to 'file6:ca/12'
```

Copies FILE5 in memory to a mass storage file named FILE6 and secures it with password 12.

```
>copy appt to 'january:ca/12'
```

Copies the appt file in memory to a mass storage file named JANUARY and secures it with password 12.

Private BASIC files (type PB) in memory may not be copied to mass storage.

> **Note:** If a copy-to-mass storage command specifies an existing file on the storage medium, the file on the medium will be re-recorded with the information from the file in memory. Afterwards, the file may appear in a different location in the catalog of the medium.

To copy a file from a mass storage medium to memory, execute this form of the COPY command:

```
COPY 'file specifier' TO 'filename'
COPY ':device code' TO 'filename'
```

Examples:

```
>copy 'file4:ca' to 'file8'
```

Copies FILE4 from mass storage to FILE8 in memory. New time and date.

```
>copy 'file6:ca/12' to 'file7'
```

Copies FILE6, secured with password 12, from mass storage to FILE7 in memory. New time and date.

If the mass storage file is secured with a password, the password must be entered in the file specifier; otherwise, error 66—invalid password—occurs and no copy is allowed. (The HP-75 will ignore passwords if supplied for mass storage files that don't have passwords.)

If there is not enough room in memory for an incoming file, the HP-75 reports error 16—not enough memory—and the copy operation is not allowed.

If an appointment file on a mass storage medium is copied to the appt file in memory, the appointments in the mass storage file are *merged* chronologically with those in memory.

Example:

```
>copy 'january:ca/12' to appt
```

Merges the appointments in the mass storage file with those in the appt file.

All other copy-to-memory commands must specify unique filenames in memory; otherwise, error 64—duplicate name—will occur and no copy will be allowed. A file on mass storage that is copied to appt must be an appointment file (type A); a file on mass storage that is copied to keys must be a text file (type T); otherwise, error 68—wrong file type—is reported. Key redefinitions take effect as soon as a copy-to-keys operation is completed.

If a copy-to-mass storage operation is interrupted by the [ATTN] key, the partially copied file will be erased from mass storage, if possible. A brief delay will occur before the prompt and cursor reappear. *Do not* press [ATTN] again until the erasure is completed, or the mass storage medium may require reinitializing. If a copy-to-memory operation is interrupted, the partial file in memory will be purged immediately.

Duplicating Files on Mass Storage (COPY)

Use the COPY command to duplicate a file in mass storage.

```
COPY 'file specifier' TO 'file specifier'
```

Examples:

```
>copy 'file8:ca' to 'file9:ca'
```

Copies FILE8 to FILE9 on the same storage medium.

```
>copy 'file8:c1' to ':c2/1'
```

Copies FILE8 on one mass storage device to FILE8 on another mass storage device. The new file is secured with password 1.

If the duplicate file is created on the same mass storage device as the original, the destination filename must differ from the source filename. If the duplicate file is created on another device, the same filename is used for both files unless different filenames are specified. If the filename already exists on the second device, error 64—duplicate name—will occur and the copy will not be allowed. As with other COPY

operations, the time and date of creation will be the same unless the filenames are different. At least one filename must be specified.

Renaming Mass Storage Files

Three forms of the RENAME command may be used to rename files on mass storage media.

```
RENAME  'file specifier'  TO  'filename'
RENAME  'filename'  TO  'file specifier'
RENAME  'file specifier'  TO  'file specifier'
```

The three forms of the command operate identically.

Example:

```
>rename 'file0:ca' to 'file10'
```

Renames FILE0 on mass storage to FILE10 on mass storage.

The RENAME command doesn't require a password and ignores the password if supplied.

Important: If a RENAME operation is interrupted, the medium may need to be reinitialized.

Purging Mass Storage Files

Execute the PURGE command to make a mass storage file unavailable for future use.

```
PURGE  'file specifier'
```

Example:

```
>purge 'file10:ca'
```

Removes FILE10 from the catalog of the storage medium.

The PURGE command doesn't require a password and ignores the password if supplied.

Important: If a PURGE operation is interrupted, the medium may need to be reinitialized.

Packing the Mass Storage Medium (PACK)

When files are purged from a medium, they leave "gaps" between adjacent files on the medium. When recording other files on the medium, the HP-75 fills these gaps with files of equal or less length. Eventually, the medium may not have room for new files, and error 95—medium full—will occur when a copy-to-mass storage operation is begun. Use the PACK command to make space available for new files.

```
PACK  ':device code'
```

The *device code* specifies the mass storage device that will pack the medium.

Example:

```
>pack ':ca'
```

Causes the CA device to re-record existing files on the medium for optimal storage.

Packing may require anywhere from a few minutes to an hour, depending on the number and lengths of the files on the medium.

> **Important:** If a PACK operation is interrupted or fails with error 97—invalid pack—the medium will probably need to be reinitialized. Do not try to COPY or CATalog the medium. The data on the medium is invalid and the HP-75 may require a full reset if such data is copied into memory. Be certain that the loop devices have dependable power sources before executing PACK. Do not press [ATTN] while a medium is packing.

Note that packing a medium involves a considerable amount of movement of the medium in the mass storage device. To maximize the lifespan of the storage medium, pack the medium only when necessary.

Advanced User's Information

Special Characters

Section 2 introduced the complete HP-75 character set, consisting of 256 letters, digits, punctuation marks, symbols, control characters, and other special characters. The HP-75 character set corresponds to decimal codes 0 through 255.

Although all HP-IL display and printer devices will display and print characters 32 through 125; most display and printer devices recognize and respond to characters outside this range. For example, the HP 82163 Video Interface interprets decimal codes 160 through 254 as inverse video characters (black on white). [CTL][J], corresponding to decimal code 10, causes both the HP-75 display and the video interface to perform a line-feed. Refer to the owner's manual for a device to determine its response to characters whose decimal codes are 0 through 31 and 127 through 255.

Escape Codes ([CTL][BACK], CHR$(27), [CTL][↑], [CTL][↓])

One character—the escape character, ESC, decimal code 27—enables special HP-IL operations. The ESC character is not displayed on the HP-75 display.

You can specify an ESC by using CHR$(27) in a DISP or PRINT statement—this directs the ESC to the corresponding output device when the statement is executed *and* allows such a program line to be listed and viewed normally.

If you generate an ESC by pressing [CTL][BACK] from the keyboard, the display shows > and the cursor turns off—pressing the next key restores the cursor. But a program line containing an ESC character (entered as [CTL][BACK]) can't be listed or viewed normally because the ESC and the next one or more characters are treated as an *escape code*.

An *escape code* is a string of characters formed by an initial escape character and one or more following characters. When an escape code is directed to a DISPLAY IS or PRINTER IS device, the device either ignores all or part of the escape code or interprets it as a display or printer instruction. An escape code may be incorporated into a larger string expression.

For example, pressing ' [CTL][BACK] E' [RTN] causes the video interface to clear the TV display and position the video cursor to the upper-left corner. Typing disp chr$(27) &'E'; [RTN] has the same effect.

Two other keystrokes send escape codes to DISPLAY IS devices:

[CTL][↑] sends ESC S, causing DISPLAY IS devices to "roll up" one line. [CTL][↓] sends a ESC T, causing DISPLAY IS devices to "roll down" one line. Neither [CTL][↑] nor [CTL][↓] affects the HP-75 display.

Refer to your peripheral owner's manual for more escape code information.

Display Escape Codes

The HP-75 display responds to 12 escape codes as a DISPLAY IS device. The 32 display window positions are identified as 0 through 31.

Escape code	Display Window Response
ESC C	Moves the cursor one position to the right.
ESC D	Moves the cursor one position to the left.
ESC E	Moves the cursor to column 0 and clears the display.
ESC G and ESC H	Moves the cursor to column 0.
ESC J and ESC K	Clears the display from the cursor location.
ESC O and ESC P	Deletes the character at the current cursor location and left-shifts all trailing characters.
ESC <	Turns off the cursor.
ESC >	Turns on the cursor.
ESC % *c r*	Moves the cursor to the specified column, where *c* is a character corresponding to the decimal code of the column and *r* is ignored by the HP-75.

Eight of these escape codes have a similar effect on the HP 82163 Video Interface (refer to appendix D, page 294).

Example:

```
10 WIDTH INF

20 DISP CHR$(27);'%';CHR$(RND*32
);CHR$(7);

30 DISP CHR$(RND*93+33);

40 GOTO 20
```

Allows an unlimited number of characters to be placed in one display line.

Positions the cursor to a random column position. (The 7 specifies row 7 of the HP 82163 Video Interface; row parameters have no effect on the display window.)

Displays a random character. The semicolon supresses the CR/LF.

Causes the HP-75 to reposition the cursor and display another character.

When run, the program causes the following display and others like it:

```
 >D   U f kd 8[  4u2   !Q 30P= y
```

Random characters will appear at random positions until you press [ATTN].

You can use the [CTL] [BACK] keystroke to control the display and cursor directly from the keyboard. For example, the following sequence of keystrokes will cause the cursor to be moved to column 29 of the display:

' [CTL] [BACK] [SHIFT] [5] [CTL] [=] [SPACE] ' ; [RTN]

The [SHIFT] [5] displays the % sign, the character code of the [CTL] [=] keystroke is 29, and the [SPACE] key inserts a null (otherwise the next character displayed would be interpreted as part of the escape sequence).

Note that pressing [CTL] [BACK] enters the escape characters into the input buffer even though the character itself is not displayed. Note also that the sequence is enclosed in quotes to form an implicit DISP statement and that it is terminated by a semicolon to suppress the CR/LF that would otherwise occur after the string was displayed. (The CR/LF would immediately reposition the cursor to the beginning of the line.) Escape sequences executed from the keyboard must be entered as DISP statements to function properly. Use the CHR$ function to control the cursor during program execution. The PAYATTN program included with your HP-75 makes effective use of display window escape codes. Refer to appendix F for a listing of the PAYATTN program.

End-of-Line Sequences (ENDLINE)

After executing a PRINT statement (unless it ends with a semicolon), the HP-75 sends two other characters—a carriage return and a line feed—to PRINTER IS devices. A CR/LF is also sent at the end of each line during a PLIST operation and whenever the number of characters printed in one line exceeds the number allowed by the current PWIDTH setting.

You can change the normal CR/LF end-of-line sequence by using the ENDLINE command.

```
ENDLINE [0-3 character string expression]
```

You may specify up to three characters in an ENDLINE command.

Example:

```
>endline chr$(13)&chr$(10)&chr$(
10)
```

Causes double-spacing to occur after each PRINT statement.

Any HP-75 characters may be specified in an ENDLINE command; the characters will be printed at the end of each print line. Specifying the null string ('' or "") suppresses the end-of-line sequence and PLIST or PRINT output will be transmitted as a continuous string (without line breaks). ENDLINE by itself reinstates the default CR/LF end-of-line sequence.

ENDLINE commands have no effect on DISP or LIST output. The current ENDLINE setting remains in effect until another ENDLINE command is executed. To restore the normal HP-75 end-of-line sequence, type endline chr$(13)&chr$(10) [RTN] or ENDLINE by itself.

Programming with HP-IL

All HP-IL commands are programmable. In addition, all HP-IL command parameters (such as device codes) may be specified using string variables and string expressions.

Programs may assign device codes, declare display and printer devices, direct output to display and printer devices, copy files to and from mass storage, turn the HP-IL loop off and on, and set end-of-line sequences.

Section 10

Redefining the Keyboard

Contents

Introduction 142
Typing Aids (DEF KEY) 143
The keys File (CAT KEYS) 144
Undoing Key Redefinitions ([SHIFT][I/R], DEF KEY, PURGE KEYS) 144
Immediate-Execute Keys (DEF KEY) 145
Fetching Key Definitions (FETCH KEY) 146
TIME and APPT Modes 147
Multiple Key Operations (@) 147
Multiple Keys Files (RENAME) 147
Copying Keys Files To and From Mass Storage (COPY) 148
Advanced User's Information 148
- Redefining Editing and System Keys ([SHIFT][I/R]) 148
- Redefining [SHIFT][ATTN] 149
- Controlling the Cursor 149
- Editing the keys File (EDIT KEYS) 151
- String Expressions in DEF KEY Commands 152
- Escape Code Key Definitions (CHR$(27)) 152
- Key Definitions That Accept Input (INPUT) 153

10

Introduction

A unique feature of the HP-75 is its redefinable keyboard. You can assign new operations to 194 keys and keystroke combinations, including:

- 60 unshifted keys (examples: [Q], [←], [APPT]).
- 59 shifted keys (examples: [SHIFT][5], [SHIFT][←], [SHIFT][ATTN]).
- 61 control keys (examples: [CTL][Z], [CTL][8], [CTL][LOCK]).
- 14 *shifted* control keys (example: [SHIFT][CTL][TAB]).

You can redefine *keystroke combinations* that use the [SHIFT] key or [CTL] key as modifiers, although [SHIFT] and [CTL] may not themselves be redefined. Do not redefine the system reset sequences such as [SHIFT][CTL][CLR] (refer to appendix C) because they will reset your computer regardless of any attempt to redefine them.

Keys and keystroke combinations can be redefined for two functions:

- Typing aids to speed up entries from the keyboard.
- Immediate-execute keys for any HP-75 operation in any of the three modes, EDIT, TIME or APPT.

A special text file named keys stores current key redefinitions in memory. The keys file, like other text files, can be edited, renamed, and copied to and from mass storage or magnetic cards. You can store and re-use a variety of customized keyboards in memory and on cards.

Typing Aids (DEF KEY)

A typing aid is a single key or a keystroke combination that displays a string of characters when it's pressed. The FET key is a typing aid because it displays the word FETCH when you press it. Typing aids may clear the display window, display messages of any length, and position the cursor anywhere on the display line.

You use the DEF KEY command for every key redefinition.*

```
DEF KEY 'key display character', 'key definition'[;]
```

Each DEF KEY declaration redefines one key or keystroke combination. DEF KEY is programmable, and may be executed in EDIT mode with either the text or BASIC prompt in the display.

The *key display character* is the one character that corresponds to the key or keystroke combination being defined. The number of characters in the *key definition* is limited only by the line length of the display. Both parameters are enclosed by single (' ') or double (" ") quotes and separated by a comma. The optional semicolon (;) at the end determines whether the key becomes a typing aid or an immediate-execute key.

The following example shows the simplest typing aid: making one key—unshifted Q—display the character of another key—unshifted X. Type the following declaration and press RTN:

```
>def key 'q','x';
```

A comma separates the two parameters. This DEF KEY declaration ends in a semicolon.

```
>
```

The Q key is redefined.

Now when you press unshifted Q, an x is displayed. Note that SHIFT Q still displays Q and that CTL Q displays the Q character—these keystrokes haven't been affected by the DEF KEY declaration.

To temporarily restore the original key definition, press SHIFT I/R (the display character keystroke) before pressing the key itself.

Example:

1. Hold down SHIFT while pressing the I/R key.
2. Release SHIFT I/R.
3. Press Q: Lowercase q is displayed.

Typing aids are quickly created.

Example: Change the % key (SHIFT 5), so that it displays LIST. Press CLR to clear the display and type:

```
>def key '%','LIST';
```

Because LIST appears between quotes, it will be displayed as typed.

```
>
```

The redefinition is stored in memory.

* The ATTN, EDIT, and RTN keys and the SHIFT I/R (display character) keystroke may not be redefined in a DEF KEY declaration, but may be redefined indirectly. Refer to Editing the keys File, page 151.

Pressing [%], or [SHIFT] [5], will now display LIST instead of %. If you continue to depress the keystroke, a whole series of LIST's will appear in the display. To display the % symbol instead, press [SHIFT] [I/R] and *then* [SHIFT] [5].

Display characters with codes greater than 127 (decimal codes 128 through 255) will not appear in the display when a key to which they have been assigned is pressed unless the *key definition* contains the [SHIFT] [I/R] keystroke. To enter the [SHIFT] [I/R] display character (◊) itself, you must press [SHIFT] [I/R] [SHIFT] [I/R].

Example: Use the DEF KEY command to redefine the shifted [A] to display the underlined A (A̲). The *key definition* must contain the [SHIFT] [I/R] display character so you must press [SHIFT] [I/R] three times, twice to enter the [SHIFT] [I/R] display character and once to enter the A̲. The keystroke sequence to accomplish the redefinition is:

```
def key 'A', '[SHIFT][I/R][SHIFT][I/R][SHIFT][I/R][CTL][TIME]';
```

Before pressing [RTN], the display will look like this:

```
>def key 'A', '◊A';
```

```
>
```

After you press [RTN], the definition is stored in memory.

The keys File (CAT KEYS)

The keys file is a special text file that stores HP-75 key redefinitions. It's special because it's created and updated by the HP-75 operating system. However, you can examine and edit the keys file in the same ways as other text files. To view the catalog entry of the keys file, execute the CAT KEYS command:

```
>cat keys
```

```
keys          T    28 09:55 02/10/83
```

The catalog entry of the keys file, showing the filename, type (T for text), size (28 bytes), time, and date of creation.

The time and date of the keys catalog show when you created the keys file. In this section, the keys file was created when you pressed [RTN] to redefine the [O] key.

Undoing Key Redefinitions ([SHIFT] [I/R], DEF KEY, PURGE KEYS)

After you've redefined a key or keystroke combination, there are four ways to restore its original display function:

- Press [SHIFT] [I/R] before you press the key itself. This is a temporary measure.
- Execute another DEF KEY command that declares the original definition.
- Rename the keys file, as explained in Multiple Keys Files, page 147.
- Purge the entire keys file. Execute PURGE KEYS, which erases all key redefinitions and removes the keys file from the system catalog.

Example: Use the DEF KEY command to redefine unshifted [Q] to its original display function, which is to display lowercase q's. The DEF KEY declaration must include two q's, so you must press [SHIFT] [I/R] twice—once for each time you type q:

```
>def key 'q', 'q';
```

Press [SHIFT] [I/R] before pressing the [Q] key.

```
>
```

Unshifted [Q] key is restored to its original function.

If you specify a null string in the DEF KEY command—that is, a key definition composed of two quote marks and nothing else—then you disable the specified key or keystroke.

Example:

```
>def key 'q','';
```

Pressing [RTN] causes the [Q] key to do absolutely nothing afterwards. Remember to redefine the key to its original function.

Immediate-Execute Keys (DEF KEY)

Typing aids are keys that display a character or characters when they're pressed. Immediate-execute keys are keys that *display and then execute* an expression, statement, or command.

Example:

Redefine the % keystroke so that it displays and then executes the LIST command.

```
>def key '%','LIST'
```

Type the % character by pressing [SHIFT] [I/R] and then [SHIFT] [5]. The semicolon is *omitted*.

Press [RTN] to store the new definition in memory. Afterwards, when you press [SHIFT] [5], two things will happen:

1. The string LIST will be displayed momentarily.
2. The LIST command will be executed.

The result is the same as typing LIST [RTN]. Note the importance of the semicolon: The semicolon determines whether a carriage return/line feed automatically follows the display of the key definition. If a DEF KEY command ends with a semicolon, the CR/LF is suppressed and the key definition is only displayed. If the semicolon is omitted, the string is displayed momentarily before a CR/LF occurs, causing the key definition to be executed.

It's important to have a cleared display line before pressing an immediate-execute key. The HP-75 interprets the display line as a whole, and extraneous characters in the display may cause syntax errors to occur.

Any command or BASIC statement that can be executed from the keyboard—such as ASSIGN IO, PRINT, and RUN—may be entered as the key definition, in uppercase or lowercase letters.

Example: Define the [CTL] [Z] keystroke combination so that it runs the SINGSONG program from section 1.

```
>def key 'Ü','run "singsong"'
```

Pressing [CTL] [Z] displays the Ü character. Notice that the filename SINGSONG must be separately quoted. The semicolon is omitted.

```
>
```

The [CTL] [Z] keystroke combination is redefined.

Because the semicolon is omitted from the command, pressing [CTL] [Z] causes `run "singsong"` to be displayed, immediately followed by a CR/LF. It's the same as if you had typed `run "singsong"` and then pressed [RTN], causing the HP-75 to execute the RUN command.

Fetching Key Definitions (FETCH KEY)

The FETCH KEY command enables you to recall the definition of a key or keystroke combination. It's a shortcut for altering key definitions.

```
FETCH KEY 'key display character'
```

Example: Check the current definition of the unshifted [W] key. Press the [FET] key and type:

```
>FETCH key 'w'
```

```
>DEF KEY 'w','w';
```

Currently, [W] displays the w character. The cursor is positioned at the first character of the key definition.

Use the FETCH KEY command to fetch the definitions of any of the 194 redefinable keys and keystroke combinations. To fetch the current definition of a key or keystroke combination that's been redefined, use the [SHIFT] [I/R] keystroke sequence to temporarily restore the original display function of the key while typing the FETCH KEY command. For example, to fetch the definition of [CTL] [Z], which was redefined above, press [FET] and type: `key '` [SHIFT] [I/R] [CTL] [Z] `'`.

Before you press [RTN] the display will show:

```
>FETCH key 'ü'
```

Pressing [SHIFT] [I/R] and then [CTL] [Z] displays the ü.

After you press [RTN] the display will show:

```
>DEF KEY 'ü','run "singsong"'
```

The current definition of [CTL] [Z].

Changing the current key or keystroke definition involves three steps:

1. Press [DEL] or [SHIFT] [DEL] to erase the definition in the display. (You may want to press [I/R] as well for the insert cursor.)
2. Type the new definition, which may or may not end with a semicolon.
3. Press [RTN] when done, or press [ATTN] or [CLR] to cancel the operation.

For example, to redefine [CTL] [Z] to its original function, press [SHIFT] [DEL] key to delete the key definition:

```
>DEF KEY 'ü','
```

[SHIFT] [DEL] deletes from the cursor to the end of the definition.

Press [SHIFT] [I/R] again before pressing [CTL] [Z], and type the rest of the definition:

```
>DEF KEY 'ü','ü';
```

Remember the final semicolon.

Pressing [RTN] restores the original display function of [CTL] [Z].

After you've fetched a key definition using the FETCH KEY command, you may change either of its two parameters—the key definition or the key display character. Note that FETCH KEY displays *single*

quotation marks as the outer pair of delimiters. To specify a filename, device code, etc. in the key definition, use *double* quotation marks for the inner pair of quotation marks.

TIME and APPT Modes

You can use the DEF KEY command to create typing aids and immediate-execute keys for TIME and APPT modes as well as for EDIT mode.

Example:

```
>def key 'z','set'
```

Defining [Z] to display set and end with a CR/LF when pressed.

```
>
```

Now when [Z] is pressed in TIME mode, the set-time template will appear.

Remember that all DEF KEY commands are entered from EDIT mode.

Multiple Key Operations (@)

You can combine two or more EDIT mode instructions in one DEF KEY command. The @ concatenator symbol is used to join BASIC statements and EDIT mode commands.

Example: Redefine the [SHIFT] [=] keystroke so that pressing [SHIFT] [=] causes the HP-75 to perform four functions:

- Set a DELAY interval of 1 second per line (the DELAY command).
- Set a display width of 32 columns (the WIDTH command).
- Display the catalog entry of the current file (the CAT command).
- List the file (the LIST command).

One DEF KEY assignment is all that's needed:

```
>def key '|','delay1@width32@cat
@list'
```

Use [SHIFT] [=] to display the | character. Spacing and case are unimportant in the key definition.

Pressing [RTN] redefines [SHIFT] [=] to execute the four specified commands.

Note that by using compressed spacing, it's possible to create a key definition that exceeds the HP-75 line length. When a long key definition is fetched, error 67—line too long—will occur, and only the first 79 characters of the key definition will be displayed in the DEF KEY line.

Multiple Key Files (RENAME)

Only one file named keys may exist in memory. However, any number of keys files may reside in memory simultaneously, as long as they have *different* filenames from each other. When you are finished using a keys file, but want to save it for future use, give it a new name using the RENAME command:

```
RENAME KEYS TO 'filename'
```

Note that the name of the active key definition file, keys, appears unquoted in all commands.

After you rename the keys file, the system catalog will report the file with its new filename, converted to uppercase. All previously redefined keys will resume their original function. Subsequent key definitions will be stored in a new keys file created and updated by the HP-75. To reactivate a previous key file, first PURGE or RENAME the current keys file and then execute:

```
RENAME 'filename' TO KEYS
```

The specified file will become the active keys file. In this way, you may switch from one customized keyboard to another.

Copying Keys Files To and From Mass Storage (COPY)

The keys file, like other text files, can be copied to and from a mass storage medium.

For example, to copy the keys file to magnetic cards:

```
>copy keys to card
```

Omit quote marks when specifying the current keys file.

Similarly, a keys file can be copied from magnetic cards to memory.

```
>copy card to keys
```

Because only one file named keys may exist in memory at a time, you need to PURGE or RENAME the current keys file before executing COPY CARD TO KEYS. Once you've finished copying a keys file to memory, all declared keys and keystroke combinations immediately assume their new definitions. Refer also to Copying Files To and From Mass Storage, section 9, page 135.

Advanced User's Information

A number of refinements give you additional control over the HP-75 keyboard and display:

- In addition to typewriter keys (like Z), you may redefine *editing* keys (like TAB) and *system* keys (like SHIFT ATTN).
- You may use key redefinitions to control the cursor's behavior.
- You may edit the keys file directly.
- You may use string expressions to specify key display characters and key definitions.

Redefining Editing and System Keys (SHIFT I/R)

Each unshifted, shifted, and control typewriter key is associated with a unique display character. For example, unshifted A is associated with a; SHIFT A, with A; and CTL A, with ". Similarly, the editing and system keys have corresponding unshifted, shifted, and control display characters. To cause an editing or system key to display its associated character rather than to perform its keyboard function, press SHIFT I/R before pressing the key itself. For example, to display the character associated with the TAB key, press SHIFT I/R and then TAB:

```
>τ
```

The TAB key, when following SHIFT I/R displays an underlined tau symbol.

Use their display characters to specify editing and system keys in DEF KEY commands.

Example: Change the [TAB] key so that it displays three blanks.

```
>def key 'I','   ';
```

As above, press [SHIFT] [I/R] first to display the [TAB] character. Then use three spaces for the key definition.

Press [RTN] to store this redefinition. Now when you press [TAB], three blanks are displayed.

Once you redefine an editing or system key, its original function is gone. For example, the [TAB] key no longer tabs in TIME and APPT modes, even when it follows [SHIFT] [I/R]. Another DEF KEY command must be used to restore the key's original function. For [TAB], type:

```
>def key 'I','I';
```

[SHIFT] [I/R] must be pressed twice for the redefinition.

When you press [RTN], the [TAB] key is restored to its original function. Refer to Character Set, appendix D, for a list of the display characters associated with the editing and system keys.

Redefining [SHIFT] [ATTN]

Normally, pressing [SHIFT] [ATTN] causes the HP-75 to turn off. By redefining the [SHIFT] [ATTN] keystroke, you cause the HP-75 to display or execute the keystroke definition whenever [SHIFT] [ATTN] is pressed or *whenever the machine attempts to turn itself off* after its 5-minute timeout period.

Example:

```
>def key '_','time$,date$'
```

Press [SHIFT] [I/R] and then [SHIFT] [ATTN] to display the _ underscore character.

Pressing [RTN] redefines [SHIFT] [ATTN] to display the time and date. The only way to turn the HP-75 off now is to execute the BYE command:

```
>bye
```

Turns the HP-75 off.

Press [ATTN] to turn the HP-75 back on. Until [SHIFT] [ATTN] is redefined to its original function with another DEF KEY declaration, the keystroke definition will be displayed and executed at the end of each timeout interval and the HP-75 will not be allowed to turn off. By including the RUN command in a [SHIFT] [ATTN] definition, you can cause the HP-75 to run a program when formerly it would have turned off.

Controlling the Cursor

Using the DEF KEY command, [SHIFT] [I/R], and the editing keys, you can manipulate both the replace cursor (▮) and the insert cursor (▯) in a variety of ways. Simply specify the editing operations you wish to occur as part of the key definition.

Example: Define the [CTL] [1] keystroke so that it finds the cosine of any number in the display. Trigonometric mode is assumed to be OPTION ANGLE RADIANS.

First, define the keystroke sequence you would use to find the cosine of the following number in the display:

```
>4.5
```

The sequence would be:

1. Type the right parenthesis ( )).
2. Shift the cursor to the left edge of the display (SHIFT ←).
3. Turn on the insert cursor (I/R).
4. Type cos(.
5. Press RTN.

To translate the above sequence into a DEF KEY declaration you need to use SHIFT I/R to specify three of the keystrokes in the above procedure: CTL 1, SHIFT ←, and I/R. Translated into a DEF KEY declaration, the procedure becomes:

```
def key 'SHIFT I/R CTL 1','))SHIFT I/R SHIFT ← SHIFT I/R I/R cos('
```

Before you press return the display will look like this:

```
>def key '1',')&~cos('
```

Semicolon is omitted for immediate execution.

I/R display character.

SHIFT ← display character.

CTL 1 display character.

Press RTN when done. Computing the cosine of an argument in the display is possible with a single keystroke:

```
>4.5
```

What's the cosine of 4.5 radians?

```
-.210795799431
```

Pressing CTL 1 returns the answer.

Virtually all cursor positioning and line editing operations (such as clearing the display with CLR, locking the keyboard in uppercase with SHIFT LOCK, and recalling the display buffer with CTL FET) can be specified in key redefinitions for single keystroke execution.

Example: Redefine CTL 0 so that it clears the display, displays FETCH KEY "", turns on the insert cursor, left-shifts the cursor, and prepares to display the character of the next key you press.

```
>def key '0','FETCH KEY""~[(';
```

Press RTN when done.

Semicolon

← CTL 0 CLR I/R SHIFT I/R

Try fetching the definition of any key, say [LOCK]. First press [CTL] [0]:

```
>FETCH KEY "◆
```

This is the current definition of [CTL] [0].

Now press [LOCK]. Pressing [SHIFT] [I/R] first is unnecessary.

```
>FETCH KEY "μ◆
```

The [LOCK] character is displayed.

```
>DEF KEY 'μ','μ';
```

Pressing [RTN] fetches the current definition of [LOCK].

Note: If specified in a key definition and not preceded by [SHIFT] [I/R], the following keys will perform their normal functions and then terminate the execution of the definition: [TIME], [EDIT], [APPT], [FET], [↑], [↓], [RUN], [SHIFT] [ATTN].

Editing the keys File (EDIT KEYS)

A direct way to modify, add, or delete key assignments is to edit the keys file. First, move the file pointer to the keys file. Use the EDIT KEYS command.

```
>edit keys
```

Specifies the keys file. The filename is not quoted.

```
keys          T   97 09:55 02/10/83
```

The catalog of the keys file appears.

Now you can edit your key assignments directly. Press [↑] to display the first line of the file:

```
:37 LIST
```

The current definition of [SHIFT] [5]—listing the current file.

A space between the line number and the key definition indicates that this is an immediate-execute key.

The line number is the decimal code of the display character associated with the [SHIFT] [5] (the % character).

If a semicolon (;) appears between the line number and the key definition, then the corresponding key or keystroke combination is a typing aid. Otherwise, a *space* specifies an immediate-execute key.

You may delete key redefinitions, edit key definitions, and add new key definitions by deleting, modifying, and adding lines to the keys file. The character set table in appendix D lists HP-75 decimal codes, display characters, and keystrokes. Note that key definitions don't appear in the key file unless the key definitions have been changed from the original.

Be *careful* when editing the keys file. Make sure your line numbers correspond to the decimal codes of the desired keystrokes. Making a mistake (such as executing a RENUMBER command) may cause unwanted redefinitions. Although it's possible to redefine the [EDIT], [ATTN], and [RTN] keys while editing the keys file, doing so is not recommended. For example, if you redefine the [EDIT] key and then press [APPT], the only way to return to EDIT mode is to schedule a command appointment that has >EDIT BASIC in the command

field. Furthermore, only the [ATTN] key can perform certain system functions, such as interrupting program execution and turning on the HP-75. Be careful also when listing or print-listing a keys file on an external device—the peripheral may respond unexpectedly to the nonstandard display characters in the file.

To leave the `keys` file, execute another `EDIT` command.

String Expressions in `DEF KEY` Commands.

You can use a string expression as a parameter in a `DEF KEY` declaration. The most general form of the `DEF KEY` command follows:

```
DEF KEY one character string expression , string expression [;]
```

A string expression may be a quoted string (as all examples have been), a string variable, or any combination of strings. Note that only one character may be specified as the first parameter.

Example:

```
>def key chr$(26),'print'
```

Pressing [RTN] defines [CTL][Z] (specified by `CHR$(26)`) to serve as a paper-advance key for `PRINTER IS` devices.

Similarly, the definition of a key may be fetched by specifying the decimal code of the key in the `FETCH KEY` command.

Example:

```
>FETCH key chr$(26)
```

```
>DEF KEY 'Ü','print'
```

The definition of [CTL][Z].

When the key definition is fetched with `FETCH KEY`, the two parameters are represented as quoted strings. This means that `CHR$(26)` above is converted to the quoted string `'Ü'`. Using the `FETCH KEY` command for special keys such as [BACK] (backspace) and [CTL][J] (line feed) may result in unusual displays, because the backspace and line feed characters are displayed differently from other symbols. The recommended practice is to use the `CHR$` function and decimal codes to specify special keys in `DEF KEY` declarations.

Escape Code Key Definitions (`CHR$(27)`)

Escape codes can be specified as key definitions. Recall that escape codes are character strings beginning with the ESC character (decimal code 27) that may be interpreted by the display window and HP-IL devices as control information rather than as data.

Example:

```
>def key 'W','print chr$(27);"&k
1S";'
```

Redefines [SHIFT][W] to send an escape code to `PRINTER IS` devices.

If the HP 82162A Thermal Printer has been declared a PRINTER IS device, then pressing [SHIFT] [W] will switch the printer to a double wide mode.

If the escape code in a key definition is one that the HP-75 display recognizes as an instruction, then the keystroke will cause the display to respond immediately to the instruction.

Example: Redefine the [CTL] [2] key to move the cursor one position to the right. The escape code that performs this cursor movement is ESC C, so the keystroke must display two characters, CHR$(27) and C.

```
>def key '2',chr$(27)&'C';
```

The display will interpret the key definition as a control instruction.

Afterwards, [CTL] [2] will cause the HP-75 to right-shift the cursor rather than display a typing aid. Refer to Display Escape Codes, appendix D, for a list of escape codes that may be used to control the HP-75 display and other HP-IL display devices.

Key Definitions That Accept Input (INPUT)

It's possible for key definitions, while executing, to accept responses from the user by means of the INPUT statement.

Example:

```
>def key '4','input a$@copy card
to a$@edit a$'
```

Specifies three operations: INPUT, COPY, and EDIT.

After the DEF KEY declaration, the [CTL] [4] keystroke causes the HP-75 to:

- Accept user input for a filename, stored in calculator variable A$.
- Begin a card reader operation that copies a card file to the specified file in memory.
- Move the file pointer to the beginning of the specified file.

When pressed, [CTL] [4] displays the key definition, followed by a carriage-return:

```
?
```

Type 'test' [RTN] for the filename

```
Copy from card:  Align & [RTN]
```

Ready to copy a card file to TEST.

Assuming a card file is actually read, the last operation caused by [CTL] [4] is moving the file pointer to the beginning of the file:

```
TEST        B  213 12:04 02/10/83
```

The catalog entry of the file in memory.

Refer also to Assigning Values from the Keyboard, section 11, page 168.

Section 11

Programming Fundamentals

Contents

Introduction 156
Writing and Editing Programs 156
The CHECK Program 157
Running Programs ([RUN], RUN) 158
Initializing Programs 158
Interrupting Programs ([ATTN], CONT) 159
Examining Programs 159
  Listing Lines (LIST, PLIST) 159
  The File Pointer and Program Pointer ([FET], [↑], [↓]) 160
  Fetching Lines (FETCH) 161
Definitions 162
Multistatement Lines 163
Program Variables 164
  String Variables 164
  Concatenating Strings (&) 164
  Assigning Program Variable Values (LET, [=]) 165
Fundamental Statements 165
  Halting Execution (END, STOP) 165
  Suspending Execution (WAIT) 166
  Program Remarks (REM, !) 166
  Displaying and Printing Information (DISP, PRINT, TAB) 166
  Assigning Values From the Keyboard (INPUT) 168
The PAYATTN Program 170
Using File Commands With Allocated Programs 172
The MANAGER Program 173
Turning the HP-75 Off (BYE) 174
Errors 174

11

Introduction

Part III of this manual has been written with the assumption that you've had some programming experience. The following sections show you how to use HP-75 BASIC statements and commands to write syntactically correct code. If you are a complete beginner at programming, you may wish to study a book on beginning BASIC programming before continuing with this manual. On the other hand, you may have had extensive experience with programming and the BASIC language, in which case appendix H, Syntax Summary, and the *HP-75 Reference Manual* may be all you need to get started.

This section covers writing and editing programs, controlling program execution, using program variables and fundamental BASIC statements, and manipulating files during program execution.

Writing and Editing Programs

You use the following steps when writing BASIC programs:

1. If the current EDIT file is a nonempty `workfile`, either PURGE, RENAME, or NAME it.
2. Execute an EDIT command to create a new BASIC file. The following pages employ a program named

CHECK, so you should type `edit 'check',basic` RTN. The catalog of the new file CHECK will be displayed, showing a file length of 0 bytes.

3. Execute an AUTO command to begin line numbering.
4. Type the program statements one by one using the typewriter and editing keys, and press RTN to enter completed lines in the file.
5. Press ATTN to stop the line numbering.

The CHECK Program

The CHECK program returns the balance of a checking account after each check or deposit is entered. The listing below shows the CHECK program as you enter it from the keyboard. You may choose to press SHIFT LOCK now if you prefer entering programs in uppercase letters.

```
>10 !CHECK program. B=Balance;A,
A$=Amount of transaction
>20 delay1@width32
>30 a$='0'!Initialize amount.
>40 input 'Initial balance: $';b
>50 input 'Check(-) or deposit(+
): ',a$;a$
>60 a=val(a$)!String to numeric.
>70 b=b+a!Update balance.
>80 disp 'New balance=$';b
>90 wait 1
>100 if a#0 then 50
```

Note that the program has no END statement. Each BASIC file is created with an invisible end-of-file marker as the last line of the file, so including END statements is optional. The examples of this manual usually omit END statements.

To run CHECK, press RUN:

```
Initial balance: $
                        PRGM
```

The program prompts for the initial amount. Type `1000` RTN.

```
Check(-) or deposit(+): 0
                        PRGM
```

The program prompts for a check or deposit amount. Enter a check amount of $14.53. Type `-14.53` RTN.

```
New balance= $985.47
                        PRGM
```

The new balance is displayed for 2 seconds (the DELAY plus the WAIT interval).

```
Check(-) or deposit(+): -14.53
                        PRGM
```

Then the previous amount is displayed. This time, enter a deposit of $120.25. Type `120.25` over the displayed amount and press RTN.

```
New balance= $1105.72
                        PRGM
```

```
Check(-) or deposit(+): 120.25
                        PRGM
```

Press RTN here to add another $120.25 to the balance.

When done with the program, enter 0 for the check/deposit amount—you can press [CLR] first to clear the displayed amount. When you type 0 [RTN], the program stops, the **PRGM** annunciator turns off, and your final balance is left in the display.

Program Explanation. The HP-75 begins executing instructions by line number, starting at line 10. The first INPUT statement (line 40) sets the initial balance. The second INPUT statement (line 50) displays the current value of the transaction amount and accepts a new value for that amount. The VAL function (line 60) converts the input to a numeric value. Finally the IF statement (line 100) tests for an input of 0 and determines whether the program will continue at line 50 or "drop" to the end of the file.

Running Programs ([RUN], RUN)

Programs can be run in EDIT mode. Press [RUN] to run the *current* BASIC file, or execute a form of the RUN command to execute *any* program in memory.

```
RUN [line number]
RUN 'filename' [,line number]
```

Examples:

```
>run
```

Begins execution of the current program at the lowest-numbered line, the same as pressing [RUN].

```
>run 30
```

Begins execution of the current program at line 30.

```
>run 'Rabbit'
```

Begins execution of the specific file, RABBIT, at the lowest-numbered line.

```
>run 'Rabbit', 300
```

Begins execution of the specified file, RABBIT, at line 300.

If the specified line doesn't exist, execution begins at the next higher-numbered line.

You may also run programs while editing text files; simply specify the BASIC filename in a RUN command. However, if you try to run a text file, error 65—access restricted—will occur. If you try to execute RUN KEYS or RUN APPT, error 84—extra characters—will occur.

If an *appointment* comes due while the HP-75 is turned off, the computer will turn on in EDIT mode. If the appointment includes a RUN command, then the specified program will begin execution. When execution stops, the HP-75 will turn off again.

Initializing Programs

The HP-75 *initializes* programs before actually running them—that is, it allocates memory for program variables, sets (initializes) the variables to undefined values, checks for certain errors (page 174), sets up other run-time conditions, and *then* begins execution. You may notice a slight delay while long programs initialize.

Initializing occurs at three times only—when you press [RUN], execute a RUN command, or execute a CALL statement (page 230). After program initialization, the catalog entry for the program will show an increase in file size, and the MEM function will show a corresponding but larger decrease in available memory. Refer to appendix D for program memory requirements.

Interrupting Programs (ATTN , CONT)

To interrupt a program, press the ATTN key. For example, press RUN to begin the CHECK program:

```
Initial balance: $
                 PRGM
```

Now press ATTN to stop execution.

```
>
```

Although interrupted, the program remains initialized.

At this point, you can perform most keyboard operations—such as checking the time, evaluating a numeric expression, examining a file, or copying a file—without deallocating the program (the opposite of initializing). (However, certain commands deallocate all interrupted programs if any of these programs are affected by the command: COPY, DELETE, MERGE, NAME, PURGE, RENAME, RUN, and TRANSFORM.) After an interruption, if the program hasn't been deallocated, the CONT (*continue*) command causes execution to resume from where it left off. (If a program is interrupted while awaiting input, it continues at the beginning of the INPUT statement.)

```
CONT [line number]
```

Example:

```
>cont
```

```
Initial balance: $
                 PRGM
```

Execution is continued.

You can continue execution at any line of the program by specifying the line number in the CONT command. Execution will begin at that line or the next higher-numbered line. Note, however, an error may result if you bypass necessary program lines or if the program has been deallocated.

One difference between RUN and CONT is that RUN defaults to the beginning of the program, while CONT defaults to the next program statement after the interruption. Another difference is that RUN can access any program in memory, while CONT applies to the current file only. A third difference is that RUN initializes programs, destroying former variable values and program conditions; CONT keeps things as they were.

If you edit—add, change, or delete—any of the lines of a program, you deallocate that program. Executing a CONT command *before* initializing a program or *after* deallocating a program causes error 31—CONT before RUN—to occur.

Note that a program will stay initialized even when you edit another program or after the HP-75 has been turned off and turned back on. A quick way to deallocate an initialized program is to delete a nonexisting line from the program file or to run another program.

The preceding discussion applies also to programs that are interrupted by the STOP statement (page 165) and by error conditions (refer to section 17).

Examining Programs

Listing Lines (LIST, PLIST)

The LIST command causes an entire file or selected lines of a file to be listed on the HP-75 display and other DISPLAY IS devices (page 128). The PLIST command causes a listing to occur on current PRINTER IS devices (this means the HP-75 display if you aren't using an HP-IL printer).

Example:

```
>list
```

Lists the entire file. Press [ATTN] to interrupt the listing.

```
>plist 50
```

Print-lists line 50.

The DELAY command controls the rate at which a listing occurs on the display. If you press any key besides [ATTN] during a listing, the DELAY will be overridden. The WIDTH and PWIDTH commands determine the maximum number of characters that will appear in each listed line. A WIDTH setting greater than 32 causes longer lines to scroll across the display (page 40). The remaining samples of this manual show all program listings with a WIDTH setting of 32. Here's a listing of the CHECK program:

```
10 ! CHECK program. B=Balance;A,
A$=Amount of transaction
20 DELAY 1 @ WIDTH 32
30 A$='0' ! Initialize amount.
40 INPUT 'Initial balance: $';B
50 INPUT 'Check(-) or deposit(+)
: ',A$;A$
60 A=VAL(A$) ! String to numeric
.
70 B=B+A ! Update balance.
80 DISP 'New balance=$';B
90 WAIT 1
100 IF A#0 THEN 50
```

Notice several things about the listing:

- The spacing has been adjusted for readability.
- The keywords and variable names have been converted to uppercase; only program remarks and characters between quotes have their case and spacing preserved.
- The prompt and cursor don't appear. Listed lines are available for viewing but not for editing. The next key you press after a listing causes the prompt and cursor to reappear.

The File Pointer and Program Pointer ([FET], [↑], [↓])

Recall from section 3 that the HP-75 uses the *file pointer* to set the pending line—that is, the line directly accessible for editing. To fetch, or display, the pending line, press [FET] and then [RTN]:

```
>40INPUT 'Initial balance: $';B
```

The file pointer is set to the line at which the program was last interrupted by [ATTN].

The prompt and cursor indicate that the line is ready to be edited. To clear the line from the display, press [CLR], [ATTN], or [EDIT]. However, the pending line stays fixed until one of the following occurs:

- While the same program file is running, execution is stopped at a different line of the file by the [ATTN] key, by an error, or by a STOP or END statement (page 165). For example, after an error has stopped execution, pressing [FET] and then [RTN] recalls the troublesome line to the display for editing.
- [↑], [↓], [SHIFT][↑], or [SHIFT][↓] is pressed to move the file pointer.
- Another line of the file is fetched (FETCH).
- Another file is edited (EDIT or NAME).
- The current file is purged (PURGE).

Programming the HP-75 file editing commands is discussed later in this section.

The *program pointer* is the mechanism the HP-75 uses to keep track of the next statement to execute. The program pointer is set independently of the file pointer. Changing the position of the file pointer doesn't change the position of the program pointer; in other words, the file pointer has no effect on the order of program execution.

Example:

1. Press [RUN] to run the current program.
2. Press [ATTN] to interrupt the program.
3. Press [FET] and [RTN] to show the location of the file pointer.
4. Press [SHIFT] [↓] to move the file pointer to the last line of the file.
5. Type [CLR] `cont` [RTN]. The program resumes execution from where it was interrupted, *not* from the last line of the file.

A program that ends "normally"—that is, by executing an `END` statement or by reaching the end-of-file marker—won't affect the location of the file pointer.

Example:

1. Press [ATTN] to interrupt the program.
2. Press [SHIFT] [↑] to move the file pointer to the first line of the program.
3. Press [RUN] to restart the program.
4. Enter a `0` for the deposit amount to end the program.
5. Press [FET] and then [RTN] to display the pending line—it's still the first line of the file.

To summarize, when a program is interrupted, the file pointer is set to the location of the program pointer. When a program ends normally, the location of the file pointer is not changed.

Fetching Lines (FETCH)

Use the `FETCH` command to fetch a specific program line for editing.

Examples:

```
>FETCH 20
```

Specifying a line number.

```
>20 DELAY 1 @ WIDTH 32
```

Fetches line 20.

```
>FETCH 'B+A'
```

Specifying a search string.

```
>70 B=B+A ! Update balance.
```

Fetches the next line *after* the pending line that contains the search string.

```
>FETCH 'B',0
```

Specifying a search string and line number.

```
>10 ! CHECK program. B=Balance;A
```

Fetches the next line (beginning from the specified line) that contains the search string.

The *search string* must be entered exactly as it appears in the program—usually in uppercase letters—or else no match will be found.

Definitions

Each line of a BASIC program consists of the following:

- A *line number*, or unsigned integer, from 0 through 9999 that determines the order of program execution.
- A *keyword*—like INPUT—that's the core instruction of the line. Keywords are the vocabulary that the HP-75 recognizes and translates into internal machine code.
- Zero or more *parameters* that supply information for keyword instructions.

Example:

```
100 BEEP 440,.5
```

A program line with line number, keyword and parameters.

A keyword with parameters is either a BASIC *statement* or system *command*. As commonly defined, *statement* generally refers to instructions which direct program flow from within a program and *command* refers to instructions that operate on programs or the machine directly. Statements are usually thought of as lines entered into a program and commands as entered directly from the keyboard. In the HP-75 the difference between *statements* and *commands* is, to some degree, arbitrary since most statements can be entered from the keyboard and most commands can be used in programs. Nevertheless, it is useful in some instances, to make the distinction and we will do so where appropriate.

The following BASIC statements are not executable from the keyboard:

END STOP	Discussed in this section.
POP GOTO FOR* NEXT* GOSUB RETURN ON TIMER OFF TIMER	Discussed in section 12.
DEF FN LET FN END DEF OPTION BASE	Discussed in section 13.
READ RESTORE DATA	Discussed in section 14.
IMAGE	Discussed in section 16.
ON ERROR OFF ERROR	Discussed in section 17.

* You can execute a FOR-NEXT loop from the keyboard if the entire loop is concatenated with @ symbols.

The following system commands are not programmable:

```
ADJUST
EXACT
RESET
SET
STATS
EXTD
```

} Discussed in section 6.

The following system commands set a machine condition:

```
ALARM OFF
ALARM ON
ASSIGN IO
BEEP OFF
BEEP ON
DEFAULT OFF
DEFAULT ON
DEF KEY
DELAY
DISPLAY IS
ENDLINE
LOCK
MARGIN
OFF IO
OPTION ANGLE DEGREES
OPTION ANGLE RADIANS
PRINTER IS
PWIDTH
RESTORE IO
STANDBY OFF
STANDBY ON
TRACE FLOW
TRACE OFF
TRACE VARS
WIDTH
```

After a machine condition is set, either from the keyboard or from a running program, it remains in effect until another command changes it. Refer to appendix D for a list of how machine conditions are set when a system reset occurs.

Multistatement Lines

Multistatement lines are program lines that contain two or more statements or commands. The HP-75 recognizes the @ symbol as a concatenator.

Example:

```
230 DISP A @ BEEP @ RETURN
```

Joining three program statements by means of the @ symbol.

Multistatement lines offer two advantages:

- They conserve memory. Refer to System Memory Requirements in appendix D.
- They result in a slight increase in execution speed.

There are several things you must be careful about when you type multistatement lines:

- It's possible to enter lines longer than 96 characters if you use condensed spacing and abbreviated keywords. When listed or fetched, long lines cause warning 67—`line too long`—to occur and the first 94 characters of the line to be displayed. The rest of the line will be maintained in the file, but you must shorten the line in order to edit it. If you press RTN with a long line in the display, the HP-75 will try to interpret just the first 94 characters of that line as the complete program line.

- The statements DATA, DEF FN, and IMAGE should not be used in multistatement lines.
- Other statements should always appear as the last statement of the line: DIM, INTEGER, REAL, SHORT, GOTO, ON...GOTO, END, RETURN, and END DEF. For example, if there is a GOTO statement in a multistatement line, branching will occur before the next statement in the line can be executed.
- If you concatenate conditional statements (IF...THEN...ELSE), you should be aware of what happens as a result of the Boolean test. If the condition is true, all instructions concatenated after THEN will be executed; otherwise, all instructions concatenated after ELSE will be executed.

Program Variables

Three types of program variables are available to you:

- *Simple numeric* variables represent single numeric values. Simple numeric variables are discussed in section 5.
- *Numeric array* variables store multiple numeric values. Array variables are discussed in section 13.
- *String* variables store character information. String variables are discussed next.

Calculator variables are assigned values using LET, [=], INPUT, and READ #; program variables are assigned values as a result of LET, [=], INPUT, READ, and READ # statements and as parameters in a DEF FN statement.

String Variables

String variable names may consist of any letter or letter-digit combination followed by a dollar sign ($). For example, acceptable string variable names are a$, C1$, X9$, and Y$.

The HP-75 assumes that a string variable stores 32 or fewer characters unless you use a DIM statement to change the length limit (page 194). To assign characters to a string variable from the keyboard, use the variable name and an equals sign.

Example:

```
>a$='hello'
```

Pressing [RTN] assigns to A$ the five characters between quotes.

A string variable may be assigned any combination of characters, including commas, blanks, and control characters, but excluding the quotation marks that you're using to delimit the string.

Calculator string variables retain their values until you reassign their values or until you execute a CLEAR VARS command (page 81).

Concatenating Strings (&)

String *concatenation* causes one string to be appended to the end of another. To join two strings, use the string concatenator—&—the ampersand.

Examples:

```
>'not'&'with'&'standing'
```

Using & to concatenate three quoted strings.

```
notwithstanding
```

The result. Note that the strings are displayed with no leading or trailing blanks.

```
>a$='super'&b$='person'
c$=a$&b$
```

C$ is assigned the concatenation of A$ and B$.

```
>c$
```

To display the value of C$.

```
superperson
```

The concatenation of two strings.

HP-75 string *functions* are discussed in section 13.

Assigning Program Variable Values (LET, [=])

The equals sign and LET statement are frequently used to assign values to program variables, although typing LET is optional.

[LET] *simple variable* [, *simple variable...*] = *numeric expression*
[LET] *string variable* [, *string variable...*] = *string expression*

For example, the following statements are equivalent:

```
100 X=12        100 X=3*4        100 LET X=12        100 LET X=3*4
```

Multiple variable assignments are also allowed. For example:

```
30 X,Y,Z=0
40 A$,B$,T$=''
```

If a numeric variable is used in a computation but hasn't been assigned a value, warning 7—`no value`—will occur, 0 will be used as its default value, and execution will continue. Likewise, if a string variable is used before being assigned a value, `no value` will be reported, and the null string will be taken as its default value. This warning condition can be changed to an error condition (no default values and execution stops) if you disable the HP-75 error default routines (by typing `default off` [RTN]).

In a string assignment, the string must contain no more characters than the size of the string variable; otherwise, error 42—`string too long`—will occur.

In general, it's good programming practice to initialize variables—that is, assign them their initial values—at the outset of programs and subroutines.

Fundamental Statements

Halting Execution (END, STOP)

Two statements halt program execution:

```
END
```

```
STOP
```

STOP allows you to check and change variable values, perform most other keyboard operations, and afterward continue program execution with a CONT command. An END statement executed in a running program deallocates program variables, causing their values to be lost.

Pressing [ATTN] from the keyboard has the same effect as a STOP statement in a program. Also, the end of a BASIC file has the same effect as an END statement in a program.

Suspending Execution (WAIT)

A WAIT statement specifies the number of seconds the HP-75 will wait before execting the next program statement.

```
WAIT number of seconds
```

The *number of seconds* may be specified by any numeric expression. The HP-75 limits the value to 0 to 2^{26} seconds (about two years), with a resolution of 0.1 seconds. Negative parameters cause a wait of 0 seconds.

If a WAIT statement follows a DISP statement, then the total time the line will appear in the HP-75 display will be the sum of WAIT and DELAY parameters. Execution of the next statement will be delayed for the entire interval.

To cancel a WAIT statement and halt the program, press [ATTN].

Program Remarks (REM, !)

Occasionally, you may want to insert remarks in order to document your programs internally. Program remarks may be added by means of the REM (remark) statement and !, the comment indicator.

```
REM [any combination of characters]
!   [any combination of characters]
```

The keyword REM must be used only as the first word in a program line; the comment indicator, !, can be anywhere in a program statement after the line number. The ! should not be used as a comment indicator in IMAGE or DATA statements. Note that *all* characters following a ! or REM keyword are considered part of a comment so that comments should always appear last in a line.

Displaying and Printing Information (DISP, PRINT, TAB)

The DISP Statement. The DISP (*display*) statement allows numeric and string information to be output to the HP-75 display and other DISPLAY IS devices, regulated by DELAY and WIDTH settings.

```
DISP [display list]
```

The display list can contain quoted characters, variable names, other numeric and string expressions, and the TAB function (page 167), separated (and possibly terminated) by semicolons or commas.

With no parameters specified, the DISP statement displays a blank line.

Numbers are displayed with a leading blank or a minus sign and with a trailing blank. Character strings are displayed with no leading or trailing blanks.

Semicolons and commas suppress carriage return/line feeds after display items. A semicolon causes the next item to be appended to the preceding item; a comma causes the display item before it to be output left-justified in a 21-character display "zone." That is, the next item will be displayed starting 21 column positions to the right of the beginning of the preceding item (or at the beginning of the next line if the line width isn't sufficient).

Example: The following short program displays 13 characters on the same line, although it executes the DISP statement 13 times.

```
10 FOR I=14 TO 26
20 DISP CHR$(I);
30 NEXT I
```

Displays the characters whose decimal codes are 14 through 26.

When the carriage return/line feed is suppressed at the end of program execution, the prompt and cursor will appear *after* the last item in the display when you press [CLR]. You can include another DISP statement (for example, 40 DISP) to complete the output operation, allowing the prompt to reappear left-justified.

To save typing, it's usually *unnecessary to type the* DISP *keyword itself.* Just type the display items and press [RTN]. The DISP statement with parameters will be entered in the BASIC file.

Example:

Lines entered as...

```
10 "Time's up!"
```

```
20 a;b;100
```

```
30 chr$(i);
```

...are listed as...

```
10 DISP "Time's up!"
```

```
20 DISP A;B;100
```

```
30 DISP CHR$(I);
```

This implied DISP statement must be the first statement in the line.

The PRINT Statement. The PRINT statement causes information to be printed on PRINTER IS devices.

```
PRINT [print list]
```

Like the *display list*, the *print list* can contain quoted characters, variable names, numeric and string expressions, and the TAB function, separated (and possibly terminated) by commas or semicolons.

When no parameters follow the keyword PRINT, the printer devices advance one line (that is, a blank line is output to them). If a PRINT statement ends in a semicolon, then the carriage return/line feed is suppressed.

The TAB Function. You can use the TAB function with DISP and PRINT statements to display or print information at specified column positions. TAB skips to the specified column before the next parameter is displayed or printed.

```
TAB (column position)
```

You may use any numeric expression to specify a positive value for the column position. Noninteger arguments are rounded to integer values. If the column position would cause the cursor to advance forward past the limit allowed by the current WIDTH or PWIDTH setting, then the position is reduced by that width until the position falls within the proper range. A nonpositive argument causes warning 54—invalid TAB—to occur and the TAB function to default to 1.

The main consideration with TAB is the line limitation of the current WIDTH or PWIDTH setting. The following examples assume a WIDTH setting 32 or greater:

```
>print 'Here';tab(18);'Now'
```

To position the second character string (Now).

```
Here              Now
```

The first character of Now is printed in column 18.

```
>tab(30);-2
```

No DISP keyword is necessary.

```
                              -2
```

The minus sign of the number is displayed in column 30.

You should use a semicolon rather than a comma after the TAB function so that the following parameter will be displayed or printed beginning at the specified column.

Note that TAB can't cause the cursor to backspace. A parameter less than the current column location of the cursor causes the next display item to appear at the specified column on the next line.

For more information about displaying and printing information, refer to section 16, Display and Printer Formatting.

Assigning Values From the Keyboard (INPUT)

The INPUT statement enables numbers and characters to be assigned to variables from the keyboard at the request of a program. That is, the INPUT statement allows you to have keyboard interactions with executing programs. The INPUT statement may take three forms:

```
INPUT variable [,variable...]
INPUT 'prompt'; variable [, variable...]
INPUT 'prompt',prompt string expression; variable [,variable...]
```

If the INPUT statement consists of a variable list only, then when the statement is executed, a question mark (?) appears at the cursor's current location in the display as a prompt for user response:

Example:

```
100 INPUT N$
```

No prompt is specified in the statement.

Executed as:

```
?
                              PRGM
```

Prompts with a question mark. You can respond with a quoted or unquoted string for a string input.

Any combination of numeric variables and string variables may be specified in an INPUT list.

Example:

```
60 INPUT X,Y,Z
```

Specifies three simple numeric variables.

When an INPUT statement calls for more than one variable, type in the variables on the same line, separated by commas, like this:

```
?7.5,10,13.7
                    PRGM
```

Press RTN when done.

INPUT variables are filled with keyboard values from left to right. Inputs for numeric variables must be *numbers*; numeric expressions (e.g., SIN(X)) are not allowed. Inputs for string variables are interpreted as straight text—that is, as characters only. The inputs for string variables can be quoted or unquoted characters, but an unquoted item shouldn't contain a comma (since commas delimit input items). Also, leading and trailing blanks, unless part of a quoted string, aren't entered in string variables. An error in response to an INPUT statement causes the HP-75 to display an error message and prompt again for input.

When you include a *quoted prompt* in the INPUT statement, that string appears in place of the question mark.

Example:

```
100 INPUT 'Name=';N$
```

Specifies the prompt.

Executed as:

```
Name=
                    PRGM
```

The characters in the quoted prompt are *protected*; that is, you may not type over them.

You may include a DISP statement before the INPUT statement to achieve a similar effect.

Example:

```
100 DISP 'Name=';

110 INPUT N$
```

The DISP statement ends with a semicolon to suppress the CR/LF.

Executed as:

```
Name=?
                    PRGM
```

The default ? prompt appears on the same line as the displayed string.

If you include a *prompt string expression* in addition to a quoted prompt in an INPUT statement, then the current value of the string expression will be displayed, when the INPUT statement is executed, with the cursor positioned at the first character.

Example:

```
100 INPUT 'Name=',N$;N$
```

The first prompt must be a quoted string; the second prompt may be any string expression.

Assuming that the current value of N$ is Ben, then the statement is executed as:

```
Name=Ben
                    PRGM
```

The second prompt is available for editing.

Pressing [CLR] will clear from the cursor position to the end-of-line but will not clear the protected characters of the quoted prompt. You may type over the prompt string expression or reenter the value of the expression. Pressing [RTN] stores the input in variable N$, the first—and in this example, the only—variable in the INPUT list. The extended INPUT statement, as used earlier in the CHECK example program, provides a convenient way to recall and update variable values.

You can display the prompt string expression without displaying any characters from the quoted prompt; use the null string ('' or "") as the quoted prompt.

Example:

```
100 INPUT '',N$;N$
```

No characters will be displayed before the value of N$ is displayed.

You can also specify a different variable as input variable than as prompt variable:

Example:

```
100 INPUT 'Name=',N$;P$
```

The current value of N$ will be displayed, but P$ will inherit the input value.

The total number of characters you may type in response to an INPUT statement is 95, *minus* the number of displayed characters of the quoted prompt. If the length of the displayed line exceeds 32 characters, then the characters will scroll to the left as they are output. Afterwards, you can use [←] and [→] to move the cursor through all but the quoted prompt characters. You may press [SHIFT] [←] and hold the [←] key down to view the first 32 characters; as soon as the [←] key is released, the cursor appears at the end of the quoted prompt, ready for input. (If the number of quoted prompt characters is greater than 64, then the middle characters can't be viewed after they have scrolled across the display.)

The MARGIN command (page 40) sets the number of characters that may be typed in response to an INPUT statement before the beeper sounds.

Note that [RTN], [RUN], [↑], [↓], or [FET] will all terminate an INPUT response. For example, during the CHECK program, you may use the [↓] key instead of the [RTN] key to enter a check or deposit amount.

The PAYATTN Program

The PAYATTN program is prerecorded on four magnetic cards supplied with the HP-75. After you've played the game, you'll use the PAYATTN file as the target of a number of file manipulations.

The magnetic cards have recorded on them the size of the PAYATTN file—approximately 4700 bytes in its deallocated form.* When initialized, the program requires about 1100 bytes more. Check to see if you have the necessary space in memory by typing mem [RTN].

- If 5800 or a higher number is displayed, there's enough room in memory to copy and run the program.
- If a number smaller than 5800 is displayed, you may need to create additional space for the program. Purge one or more files from memory.

Next, execute a COPY command:

* The size of the PAYATTN program may vary, depending on which version of the program you're using.

```
>copy card to 'payattn'
```

The COPY command specifies the filename you want for the program.

```
Copy from card:  Align & [RTN]
```

The response indicates the HP-75 is ready for a card reader operation.

The HP-75 guides you through this copy operation as through all card reader operations, displaying messages at the current DELAY rate.

Type `edit 'payattn'` to move the file pointer to the PAYATTN file. PAYATTN challenges your ability to remember and match six pairs of hidden characters, two at a time, in a field of 12 positions. You begin each round of the game with 100 points. After the HP-75 gives you a glimpse of one character at a time, you attempt to match the character pairs with as few errors as possible. Each correct match gives you 20 points. Each mismatch costs you 25 points. If your scores drops to 0, the round is ended.

Press [RUN] to start; there will be a slight delay while the program is initialized.

```
        Pay  Attention
                    PRGM
```

The program greeting

```
Play - 1 person or 2 people:
                    PRGM
```

Type a 1 [RTN] to specify single competition.

```
Choose - easy or hard:
                    PRGM
```

Select the level of difficulty. Try *easy* for starters. Type e [RTN].

```
What are your two initials?
                    PRGM
```

Type in your two initials, in uppercase or lowercase letters (this example uses AB). The next [RTN] will start the game. However, read the next few sentences before pressing [RTN].

```
Here we go, AB  Pay attention!
                    PRGM
```

After you press [RTN], this message is displayed momentarily. The HP-75 is about to let you glimpse the characters you'll match up during the game.

```
  # X # # # # # # # # # # < Start
                    PRGM
```

The characters are "flipped" one by one.

After you've seen the characters, the HP-75 displays the two brackets ([ ]) you'll use to select characters, your initials, and the beginning point total:

```
[#]# # # # # # # # # # # AB:100
                    PRGM
```

Each # hides a character.

Use the [→] and [←] keys to select a character position. Then press the space bar to reveal the character hidden at that position.

Example:

```
 #[#]# # # # # # # # # # AB:100
                    PRGM
```

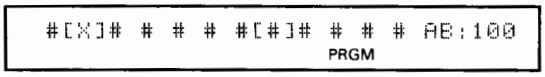
The display as you move to the second character.

```
 #[X]# # # # # # # # # # AB:100
                    PRGM
```

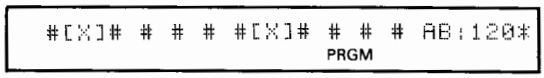
Press the space bar to play your choice.

Now you have to find the match for this character. Move to where you think it might be and press the space bar.

Example:



Let's try here.



Success! The score is increased.

After a match, the characters remain displayed, your score is increased, and you create a bonus situation (as indicated by the asterisk). A bonus means that now you're playing for double the score, 40 points instead of 20. When you miss a pair, the characters are again hidden, your score is decremented by 25, and you lose the bonus play. Three hundred points are possible.

Play a round or two to get the feel of the game. The difference between an "easy" and "hard" round is that the hard round uses punctuation marks instead of letters and displays the characters for a shorter period of time. The PAYATTN program is listed in appendix F; lines 1080 and 1090 may be changed to vary the way the program works. For example, the variables T2, T3, T5, and T6 determine the delays in the program, N9 sets the number of pairs of symbols, and B1, I1, D1, and E1 determine the scoring.

The PAYATTN file will be significantly altered during the file manipulations that follow.

Using File Commands With Allocated Programs

BASIC and text files in memory may be manipulated by a variety of system commands. One or more files may be program files that are initialized, or allocated—that is, a program that is running or waiting to continue running. Any of the file commands may be executed by a running program, or a command may be executed from the keyboard during an interrupt condition. The list that follows describes responses for such situations. You should note that a multistatement instruction executed from the keyboard responds much as a program does—that is, its execution halts before completion if one of its commands would cause a program halt, as described below.

Command	Comments
CAT	The specified catalog entry is displayed. After a CAT ALL command, execution of a running program resumes when you press another key besides an arrow key.
CAT CARD	The catalog entry of the card file will be displayed. In a running program, when a key is pressed, execution will continue at the following program statement—other card reader commands cause execution to continue as soon as the card reader operation is completed.
CONT	The current program continues execution at the specified line or where the program halted if no line number is specified.
COPY	Copying an allocated program deallocates that program.
DELETE	Deleting a line from an allocated program deallocates that program.
EDIT	The catalog entry for the EDIT file will be displayed for the current DELAY rate, and the specified file will be made available for editing.
FETCH	The specified line will be fetched to the display buffer. The line will be displayed when keyboard input is next expected.
MERGE	Merging an allocated program in any way deallocates that program.
NAME	Naming an allocated program deallocates that program.
PURGE	Purging an allocated program releases all the memory that it used.

Command	Comments
RENAME	Renaming an allocated program deallocates the program, and renaming a running program halts it.
RENUMBER	Does not deallocate or halt a running program. Line numbers specified after AUTO, CONT, DELETE, FETCH, LIST, MERGE, PLIST, RENUMBER, PRINT #, READ #, RESTORE #, and RUN statements aren't changed.
RUN	Running any program first deallocates all allocated programs.
TRANSFORM	Transforming an allocated program deallocates that program.

The file pointer must first be moved to a file (use EDIT) before a DELETE, FETCH, MERGE, NAME, or RENUMBER command can be applied to that file.

The MANAGER Program

The MANAGER program uses a number of file commands to manipulate the PAYATTN file. Type `mem` [RTN] to check available memory; MANAGER requires 300 bytes. You can "free" about 1000 bytes by deallocating PAYATTN—type `clear vars` [RTN]. (This deallocates programs and clears calculator variables.)

Then execute `EDIT 'MANAGER'` and enter the following program:

```
 10 ! File manager.
 20 DELAY 2 @ WIDTH 32
•30 EDIT 'PAYATTN'

•40 DELETE 1200,1820
 50 DELETE 1850,2420
 60 DELETE 2450,9999
•70 LIST 1830,9999 @ BEEP

•80 INPUT 'Temp. filename: ';B$
 90 COPY TO B$
 100 CAT B$
 110 RENUMBER 0,10,0,1190

 120 MERGE B$,1110,1190
 130 PURGE B$
 140 EDIT 'MANAGER'

 150 RUN 'PAYATTN'
```

Makes PAYATTN available for editing. The catalog entry is displayed for 2 seconds.

Deletes lines from PAYATTN.

Lists the end of PAYATTN. Press any key to speed up the listing.

We'll use TEMP as the name of a new file.

Copies the current PAYATTN file to TEMP.

Displays the catalog entry for TEMP.

Renumbers the beginning of the current file (PAYATTN) so that numbering begins at line 0.

Merges a portion of TEMP into PAYATTN.

Purges the temporary file.

Returns the file pointer to this program and displays the catalog entry.

Runs the current version of the program.

When you run the program, refer to the listing above to follow the file manipulations. The end result is that the program greeting portion of PAYATTN is duplicated each time the MANAGER program is run.

Turning the HP-75 Off (BYE)

The programmable BYE command turns the HP-75 off. When the HP-75 turns on again, execution resumes at the statement following the BYE command.

Example:

```
10 INPUT 'Your password: ';A$
20 LOCK A$
30 BYE @ DISP 'Correct'
```

The program locks the HP-75 with a password and turns the machine off. When you press [ATTN], the HP-75 prompts for the password; if it's entered correctly, the remainder of line 30 is executed.

If the HP-75 is turned off under program control, then appointments that come due while the HP-75 is off will be ignored until the HP-75 is turned back on and finishes executing the program.

If a STANDBY OFF setting is in effect while a program is executing, then the HP-75 will turn itself off if it is kept waiting longer than 5 minutes for keyboard input. For example, the INPUT statement requires a keyboard response, and if no response occurs for 5 minutes, then the HP-75 turns itself off. The ASSIGN IO, CAT ALL, and card reader similarly require a keyboard response. The program remains initialized while the HP-75 is off and may be continued with the CONT command after the HP-75 is turned back on.

Errors

There are four types of errors that can occur during the development and execution of your program: *syntax* errors, *initialization* errors, *run-time* errors, and *logical* errors.

You've seen that your lines are checked as you enter them for syntax (or language) errors. You can correct syntax errors by inserting, deleting, or replacing characters while editing the line.

Initialization errors occur when you have finished writing the program and try to run it. Such errors include missing line numbers, duplicate user-defined functions, illegal array dimensions, etc. You are informed of all initialization errors before the program can be run.

The third type of error occurs when the program is running. All run-time errors interrupt a running program and cause it to halt unless DEFAULT ON is in effect or you catch the errors with an ON ERROR statement (page 258). Run-time errors can include referencing a nonexistent array element, attempting to use uninitialized data, string overflows, READ-DATA variable mismatch, trying to access a nonexistent file, etc.

The fourth type of error is sometimes the most difficult to catch. Logical errors are the result of incorrect program *design* not incorrect *coding*. If a program runs from beginning to end but produces incorrect or meaningless results, if a program starts executing but never stops, or if some inputs give correct results while other inputs don't, you have encountered a logical error. Finding and eliminating logical errors is outside the scope of this manual, but avoiding logical errors is fundamental to good programming.

Section 17, Debugging Operations, provides you with the tools you need to recover from both run-time and logical errors.

Section 12

Branches, Loops, and Subroutines

Contents

Introduction 176
Unconditional Branching (GOTO) 176
Conditional Branching (IF...THEN...ELSE) 177
The ELSE Option 178
Multiple Instructions After THEN and ELSE (@) 179
Looping (FOR-NEXT) 179
Changing the Increment Value (STEP) 180
Nested Loops 181
Subroutines (GOSUB, RETURN) 182
The Computed GOTO Statement (ON...GOTO) 182
The Computed GOSUB Statement (ON...GOSUB) 183
Bypassing a Pending Subroutine Return (POP) 183
Program Timers 186
Setting a Timer (ON TIMER #) 186
Turning Off a Timer (OFF TIMER #) 187
Timer Branches and Subroutines
(ON TIMER #...GOTO and ON TIMER #...GOSUB) 188
More About Timers 189

12

Introduction

Branches, loops, and subroutines can be used to control the flow of program execution. This section covers:

- Unconditional branching with GOTO statements.
- Conditional branching with IF...THEN...ELSE statements.
- Looping with FOR-NEXT statements.
- Creating subroutines with GOSUB, RETURN, and POP statements.
- Branching with ON...GOTO and ON...GOSUB statements.
- Setting timer interrupts with ON TIMER # statements.

Branching statements alter the sequential flow of program execution and often cause program execution to skip one or more program instructions. A branching statement that specifies a line number (e.g., GOTO 100) or that causes execution to continue at a line different from the currently executing line (e.g., RETURN) may not be executed from the keyboard.

Unconditional Branching (GOTO)

The GOTO statement is simple and direct; it transfers program control to the program line whose line number you specify.

```
GOTO line number
```

This is called *unconditional* branching because a branch occurs every time the statement is executed.

Example:

```
290 GOTO 3000
```

Transfers execution to line 3000.

GOTO statements may branch to higher-numbered or lower-numbered lines and are useful for creating program loops.

Conditional Branching (IF...THEN...ELSE)

A *conditional* branch enables a program to make a decision as the result of a specified condition, or test. Use the IF...THEN statement for conditional branching. The simplest form of the statement is:

```
IF numeric expression THEN line number
```

If the condition is true—that is, if the *numeric expression* evaluates to a nonzero value—the THEN part of the statement is executed. If the condition is false—the expression evaluates to zero—execution continues at the next program line after the IF...THEN statement.

```
90 IF A THEN 1200
```

If A is nonzero, then execution continues at line 1200; otherwise, execution continues at the next line in the program.

```
90 IF C<.0001 THEN 1200
```

If C is less than .0001, the relational expression evaluates to 1 and the program branches to program line 1200.

```
90 IF A$="Y" OR A$="y" THEN 1200
```

If A$="Y" or "y", the Boolean expression is true and execution jumps to line 1200.

IF...THEN statements are most commonly used with relational operators (=, <, >, <=, >=, <>, and #), although the test can be based on the value of any arithmetic or Boolean expression.

Another form of the IF...THEN statement provides conditional execution of a statement without necessarily branching:

```
IF numeric expression THEN [allowable statement...] or [command...]
```

If the condition is true (evaluates to a nonzero number), the specified instruction or instructions are executed. When the condition is false (evaluates to zero), the instruction or instructions after THEN are skipped and execution continues at the following program line. Any combination of allowable statements or commands that will fit on a single line can be concatenated with the @ and placed after the THEN.

Examples:

```
440 IF I THEN I=I-1
```

If the value of I is nonzero, then decrement I; otherwise, ignore the instruction.

```
5 IF N=2 THEN ASSIGN IO ":tv,:p"
```

If N equals 2, then assign the HP-IL loop accordingly.

```
1010 IF MEM<C THEN PURGE C$
```

If memory is less than the value specified by C, then purge the specified file.

All commands and most BASIC statements may be specified as conditional instructions. The following statements are not allowed after THEN:

FOR	IMAGE
NEXT	INTEGER
DATA	ON ERROR
DEF FN	ON TIMER
DIM	OPTION BASE
END DEF	REAL
IF	SHORT

Error 86—illegal context—will occur if you type an IF...THEN statement that includes one of the above.

Note that an IF...THEN statement that doesn't cause branching may be executed from the keyboard.

Example:

```
>if 1>0 then disp 'true'█
```

```
true
```

Pressing [RTN] causes the IF condition to be evaluated and the THEN instruction to be executed.

The ELSE Option

A third keyword may be included in an IF...THEN statement: ELSE. If the numeric expression is evaluated as false, the program will perform the specified ELSE instruction. A variety of forms of the IF...THEN...ELSE statement are available:

```
IF numeric expression THEN line number                ELSE line number
                          [allowable statement...]          [allowable statement...]
                          [          or           ]          [          or           ]
                          [command...             ]          [command...             ]
```

Examples:

```
490 IF I=1 THEN E$='$' ELSE E$=
'%'
```

Depending on the value of I, E$ will assume one of two values.

```
4560 IF ABS(X)<.001 THEN RETURN
ELSE 4400
```

Depending on the value of X, either the subroutine finishes executing or a branch to line 4400 occurs.

```
620 IF A*B#0 THEN DEFAULT OFF EL
SE DEFAULT ON
```

The values of A and B determine the type of error processing.

The same restrictions apply for ELSE as for THEN—you may follow ELSE with any system command or any allowable BASIC statement or a combination of commands and allowable statements or with a line number.

Multiple Instructions After THEN and ELSE (@)

You may include more than one command or statement after a THEN or ELSE keyword using the @ concatenator symbol.

Examples:

```
120 IF A<1 THEN BEEP @ DISP 'Nu
mber too small.' @ GOTO 110
```

Causes three actions to occur if the specified condition is true.

```
2000 IF T=-1 THEN STOP ELSE A=0
@ BEEP 700
```

Causes the HP-75 to beep and set A=0 when T isn't equal to −1.

To summarize, if the test condition is true, all THEN instructions will be executed. If the test condition is false, all ELSE instructions will be executed. The length of a multi-instruction IF...THEN...ELSE statement is limited only by the line length of the HP-75. Note that an IF...THEN statement may not contain another IF...THEN statement.

Looping (FOR-NEXT)

Repeatedly executing a series of statements is known as *looping*. The loop in the CHECK program (page 157) causes the program to continue executing until it is interrupted with [ATTN] or until the IF condition tests false and the loop is exited.

A clear and efficient way to control looping is to use FOR-NEXT statements. The FOR and NEXT statements enclose a series of statements, enabling you to repeat those statements a specified number of times.

```
FOR loop counter = initial value TO final value [STEP increment value]
        ⋮
NEXT loop counter
```

The FOR statement defines the beginning of the loop and specifies the number of times the loop is to be executed. The loop counter must be a simple numeric variable, like I, or I1. The *initial value*, *final value*, and *increment value* can be any numeric expression. If a STEP *increment value* is not specified, the *loop counter* increments by 1 after each pass through the loop.

The FOR statement does the following:

- It sets the loop counter to the initial value.
- It causes the HP-75 to store the final value for the loop counter.
- It tests for the exit condition by comparing the current value of the loop counter with the final value.

While the value of the loop counter is less than or equal to the final value (for a positive increment value), execution continues at the next statement after FOR.

The NEXT statement does the following:

- It increments (or decrements) the loop counter.
- It returns control to the test condition of the FOR statement and thereby defines the end of the loop.

When the value of the loop counter becomes greater than the final value of the FOR statement (or when the value of the loop counter becomes less than the final value when a negative increment value is used), then the loop is exited and program control is passed to the next statement after NEXT.

The initial and final values of the FOR statement are computed only once—as the program first enters the loop. It's not possible to change the final value of the loop counter from inside the loop. Only the FOR statement of a loop will be executed if the initial value of the loop counter is greater than the final value (or if the initial value is less than the final value when a negative increment value is used).

Example:

```
 10 FOR I=1 TO 28
•20 DISP STR$(MOD(I,10));

 30 NEXT I
•40 DISP I
```

The STR$ function causes the value of (MOD(I,10)) to be displayed with no leading and no trailing blanks (page 201).

Displays the value of I after the loop is exited.

Before running the program set a DELAY of 0 seconds and a WIDTH of 32 or more columns. The program displays the following:

```
12345678901234567890123456789 29
```

The output shows that the NEXT statement increments the loop counter past the final value.

Note: This program may be easily modified to show the 32 column positions on the HP-75 display window. Change line 10 to FOR I=1 TO 32, and change line 40 to DISP.

You can execute a FOR-NEXT loop from the keyboard if the entire loop is concatenated with @ symbols and if execution is "contained" within the display line—that is, if execution does not branch to another line.

Example:

```
>for i=1 to 5@beep i*200@next i
```

Pressing [RTN] causes the BEEP statement to be executed five times.

Thus, it's possible to include FOR-NEXT loops as part of key redefinitions.

Changing the Increment Value (STEP)

The loop counter variable of a FOR-NEXT loop may assume any numeric value, including negative and non-integer values. The STEP keyword in the FOR statement allows you to specify any increment or decrement value for the loop counter.

Example: Display the decimal fractions from .5 to 0 in decreasing steps of 1/16, or .0625.

```
•10 FOR I=.5 TO 0 STEP -.0625
 20 DISP I;
 30 NEXT I
```

Specifies a negative, fractional STEP value.

The initial value of I is .5. Each time NEXT I is executed, I is decremented by .0625. When I becomes less than the final value (0), the loop is exited. The program generates two lines of output (assuming WIDTH 32):

```
 .5   .4375   .375   .3125   .25
 .1875   .125   .0625   0
```

The loop is executed nine times.

Note: Because the DISP statement in line 20 ends with a semicolon, no carriage return occurs at the end of the program to return the cursor to the left edge of the display. Therefore, the next display character that's entered from the keyboard will appear on the same line as and after the last item in the display (the 0). Press [ATTN] or [EDIT] to clear the line.

If the final loop counter value is greater than or equal to the initial loop counter value and if the STEP value is 0, then the FOR statement will cause an endless loop.

Nested Loops

When one loop is contained entirely within another, the inner loop is said to be *nested.* A loop can be contained within a loop that is contained within a loop (up to 255 nested loops), as long as the loops don't overlap each other.

Example:

Incorrect Nesting

```
10 DISP 'I','J'
20 FOR I=1 TO 3
30 FOR J=1001 TO 1003
40 DISP I,J
50 NEXT I
60 NEXT J
```

Correct Nesting

```
10 DISP 'I','J'
20 FOR I=1 TO 3
30 FOR J=1001 TO 1003
40 DISP I,J
50 NEXT J
60 NEXT I
```

The incorrectly nested loop causes error 47—no matching FOR—at line 60 when execution exits the I loop. The correctly nested example above displays the following:

```
I                    J
 1                    1001
 1                    1002
 1                    1003
 2                    1001
 2                    1002
 2                    1003
 3                    1001
 3                    1002
 3                    1003
```

Program Explanation. The J loop is completed before I is incremented. That is, when J is incremented to 1004 by the NEXT J statement, execution moves to the NEXT I statement. Each time the FOR J... statement is executed, J is initialized to 1001. When I reaches a value of 4, the outer loop is exited and the program ends.

You may branch out of a FOR-NEXT loop, but branching into a loop will cause error 47—no matching FOR—if a NEXT statement is executed before the program has executed the corresponding FOR statement.

Subroutines (GOSUB, RETURN)

Often, the same sequence of statements is executed in several places within a program. By using a subroutine you can key in the group of statements only once and then access those statements from different places within the program. If you group all of the often-used routines at the end of your program, you can make the design of the program easier to understand.

```
GOSUB line number
```

The GOSUB statement transfers program control to the subroutine you wish to execute. The *line number* is the first line of the subroutine.

Example:

```
90 GOSUB 800
```

When executed, the statement passes control to the subroutine beginning at line 800.

A common programming practice is to include a remark in the line before the first line of a subroutine so that the purpose of the subroutine is apparent in a program listing.

The last executed statement of a subroutine must be a RETURN statement.

```
RETURN
```

As soon as a RETURN is encountered, program control is transferred to the next statement after the corresponding GOSUB statement. This statement may or may not appear in the same line as the GOSUB statement.

Example:

```
90 GOSUB 800 @ DISP 'Back home'
```

When the subroutine encounters the matching RETURN after line 800, execution returns to the DISP statement.

All main program variables are accessible to subroutines. If the value of the variable is changed within a subroutine, it is also changed in the main program.

Subroutines may be *nested*; that is, program execution may branch to a second subroutine before the RETURN statement of the first is executed. Up to 255 levels of nesting are allowed. When a RETURN is encountered, control returns to the subroutine that was most recently executing, at the statement immediately following the GOSUB statement.

The Computed GOTO Statement (ON...GOTO)

The computed GOTO statement enables you to transfer program control to one or more program lines, depending on the value of a numeric expression.

```
ON numeric expression GOTO line number [, line number...]
```

The numeric expression is evaluated and rounded to an integer value. A value of 1 causes control to be

transferred to the first program line specified in the ON...GOTO statement; a value of 2 causes control to be transferred to the second line specified in the statement, and so on.

Example:

```
200 ON R GOTO 25, 80, 150
```

This statement says, if R=1, go to line 25, if R=2, go to line 80, and if R=3, go to line 150. A rounded value for R less than 1 or greater than the number of lines in the list will cause error 11—arg out of range—when the statement is executed.

The Computed GOSUB Statement (ON...GOSUB)

The computed GOSUB statement enables you to access any of one or more subroutines based on the value of a numeric expression. It operates exactly like an ON...GOTO statement except that ON...GOSUB transfers control to the first statement of a subroutine.

ON *numeric expression* GOSUB line number [, *line number*...]

The numeric expression is evaluated and rounded to an integer value. A value of 1 causes the subroutine at the first line number in the list to be accessed, and so on.

The corresponding RETURN statement of the subroutine causes program execution to return to the statement immediately following the ON...GOSUB statement.

Example:

```
 100 INPUT 'Argument, 1-3: ';A,I

•200 ON I GOSUB 500,700,900

 300 BEEP
 400 STOP
•500 DISP SIN(A)
 600 RETURN
•700 DISP COS(A)
 800 RETURN
•900 DISP TAN(A)
 1000 RETURN
```

This statement means:
If I=1, then GOSUB 500
If I=2, then GOSUB 700
If I=3, then GOSUB 900

First subroutine.

Second subroutine.

Third subroutine.

The RETURN statement of each subroutine above returns control to the next statement following the computed GOSUB statement, that is, to the BEEP statement of line 300.

If the value of the numeric expression rounds to less than 1 or greater than the number of lines in the list, error 11—arg out of range—occurs.

Bypassing a Pending Subroutine Return (POP)

When a GOSUB is executed, a *pending return* condition is created to "remember" which statement program control should return to when the next RETURN statement is executed. Executing GOSUB several times without executing a RETURN creates a corresponding number of subroutine levels and pending returns.

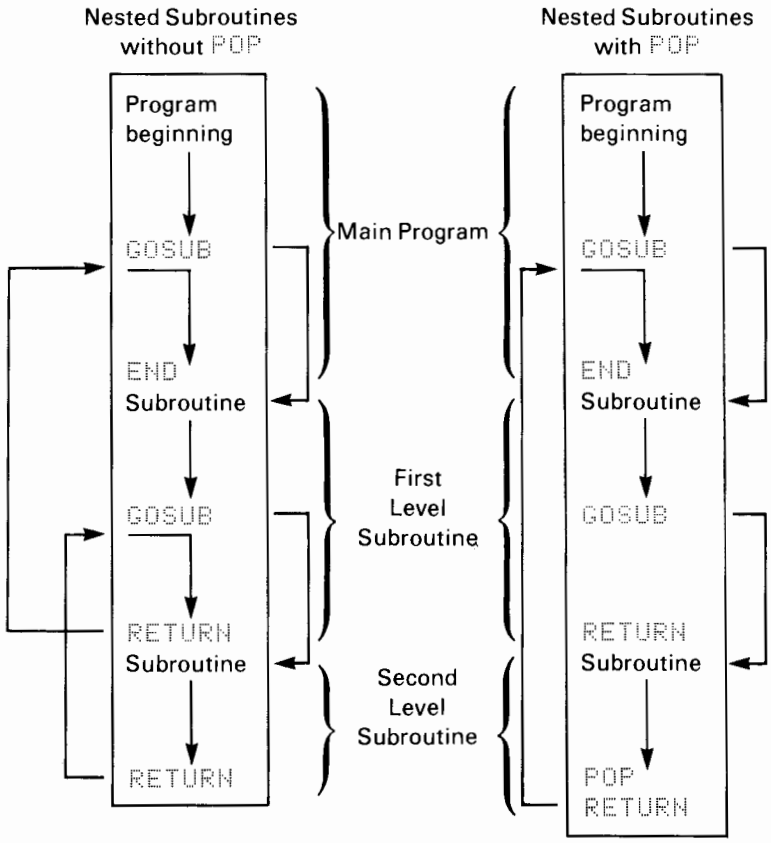
However, there may be times when you want to alter the path of returns from subroutines so that one or more levels are bypassed. If this is done by simply branching around one or more RETURN statements, there is a possibility of error conditions due to:

- Subsequent RETURN statements causing execution to return to the wrong location in the program.
- Accumulating more than 255 levels of nested subroutines.

The POP statement helps you avoid such errors by cancelling the pending return from the most recent GOSUB statement.

POP

One POP statement causes program execution to bypass one subroutine level.



Example: The following program can be subdivided as follows:

- Main program—lines 10 through 40.
- First subroutine—lines 50 through 80.
- Second subroutine—lines 90 through 120.

When the program is run without executing the POP statement in line 110, program execution follows the path shown on the left in the preceding illustration; when the POP statement *is* executed, program execution follows the path shown on the right.

```
 10 DISP 'Main Program'
 20 GOSUB 50 ! to the first
 subroutine.
 30 DISP 'Back to Main' ! from
 first or second subroutine.
 40 STOP
•50 DISP '  First Subroutine'
 60 GOSUB 90 ! to the second
 subroutine.
 70 DISP '  Back to First' ! from
 second subroutine.
 80 RETURN ! to the main program.
•90 DISP '    Second Subroutine'
•100 INPUT '    pop? y/n: ';A$
•110 IF A$='y' THEN POP

 120 RETURN ! if (N), to pending
 subroutine; if (Y), to main
 program.
```

Type the quoted string with two leading blanks.

Include four leading blanks in the quoted strings.

If the user chooses the POP option, then the pending return will be bypassed.

Set a DELAY rate of 1 and run the program:

```
Main Program
  First Subroutine
    Second Subroutine
    pop? y/n: n
  Back to First
Back to Main
```

Decline the POP option by typing n RTN.
The normal return path from a nested subroutine.

Run the program again, this time specifying the POP option:

```
Main Program
  First Subroutine
    Second Subroutine
    pop? y/n: y
Back to Main
```

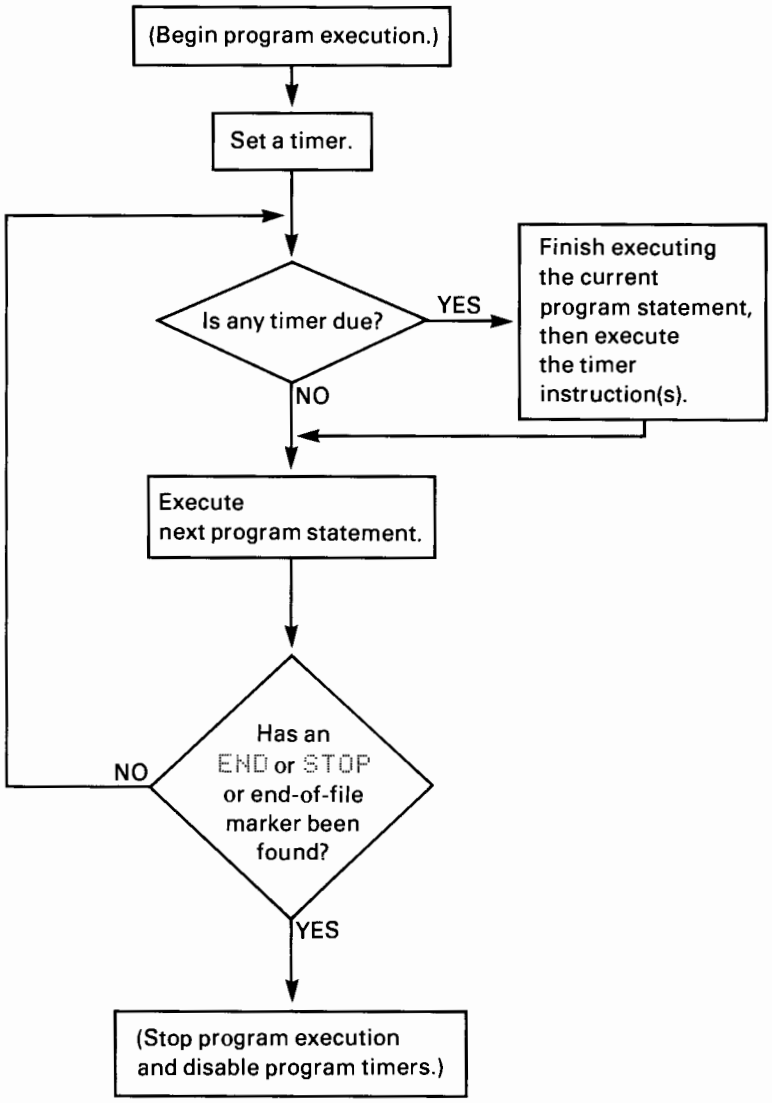
POP cancels the most recent pending return.

The POP statement may be executed repeatedly to exit from a nested subroutine many levels deep. However, if the number of executed POP statements exceeds the number of subroutine levels and if afterwards a RETURN statement is encountered, then error 50—RETURN w/o GOSUB will occur. The RETURN statement causes the error because there will be no pending return condition.

Program Timers

One or more timers may be set in an HP-75 program to execute any command or allowable statement at specified time intervals during the program.

The flowchart below illustrates how a program timer controls program flow.



Setting a Timer (ON TIMER #)

Use the ON TIMER # statement to set any of 1001 available program timers.

```
ON TIMER # timer number , seconds  [allowable statement...] or [command...]
```

Timer Number. The timer number can be any numeric expression that rounds to an integer value from 0 through 1000. If an existing timer number is duplicated in a new ON TIMER statement later in a program, the second timer resets and replaces the first timer.

Seconds. The number of seconds can be specified by any numeric expression. Timer interrupts are accurate to tenths of a second. The minimum time you can specify is 0.1 seconds; shorter time parameters and

negative values default to 0.1 seconds. The maximum time you can specify is virtually unlimited—in the range of centuries ($2^{41} - 1$ seconds).

Allowable Statement or Command. Although many timer applications involve GOTO or GOSUB branches, any BASIC statement that is allowed after a THEN keyword (page 178) and any system command may be specified in an ON TIMER # statement. Statements and commands may be concatenated in an ON TIMER # instruction by means of the @ symbol.

Example:

```
10 ON TIMER # 1, 1 DISP TIME$

20 GOTO 20
```

Sets timer #1 so that it interrupts the program every second to display the time.
Causes execution to loop at this line, waiting for the timer interrupt.

A timer interrupt will be delayed until all statements in the current line have been executed (although a timer interrupt will break a multistatement line after a GOSUB, NEXT, CALL, GOTO, or CONT statement in that line). After the timer has performed its specified instruction or instructions, the program continues as if no interruption at all has occurred. That is, execution returns to the line following the one which was executing when the timer interrupt occurred. In the example above, this means execution returns to line 20 because line 20 is the line that is normally executed after the GOTO statement.

An END statement or an end-of-file mark will clear the timers that were declared in the program. If you stop program execution with [ATTN] or STOP, the timers of that program will be disabled. Executing CONT will restart the timers from a count which will vary depending on the last time the timer operation was interrupted. To avoid unexpected timer interrupts, don't use CONT in programs with timers.

One feature of program timers is that they can cause the HP-75 to *turn on* at specified time intervals if the HP-75 has been turned off previously under program control. If a program executes a BYE command, the HP-75 turns off but keeps all current timers active. When the next timer comes due, the HP-75 turns back on, the timer instruction or instructions are executed, and then program execution continues at the statement following the BYE command.

Example:

```
10 ON TIMER # 1,2 BEEP

20 BYE
30 DISP TAB(12);'On!'
```

Timer #1 is set to interrupt the program every 2 seconds. This will happen while the HP-75 is turned off. The BEEP instruction is executed and control passes to line 30.

Except in this situation—when a timer remains functioning although the HP-75 is off—a timer is capable of interrupting execution only while the program that set the timer is actively running. When one program calls another, the timers in the first program will continue counting, but they will cause no interrupts while the second program is executing. Refer also to Program Calls, page 230.

Turning Off a Timer (OFF TIMER #)

The OFF TIMER # statement allows a program to turn off individual timers.

```
OFF TIMER # timer number
```

A program will disable an individual timer when it executes an OFF TIMER # statement for that timer. No further interrupts will occur from the specified timer unless the timer is set by another ON TIMER # statement.

Timer Branches and Subroutines (ON TIMER #...GOTO and ON TIMER #...GOSUB)

A common application of program timers is to run entire routines at specified intervals. This is accomplished by ON TIMER #...GOTO and ON TIMER #...GOSUB statements.

GOTO. A GOTO instruction in an ON TIMER # statement transfers execution to some other line in the program. When the specified timer comes due, execution branches from the end of the currently executing line to the beginning of the specified line.

Example:

```
•10 ON TIMER # 5, 10 GOTO 500
 20 GOTO 20
 500 DISP 'Ten Seconds'
 510 BEEP 100
•520 DISP 'Continue'
 .
 .
 .
```

The interrupt will transfer execution to line 500.

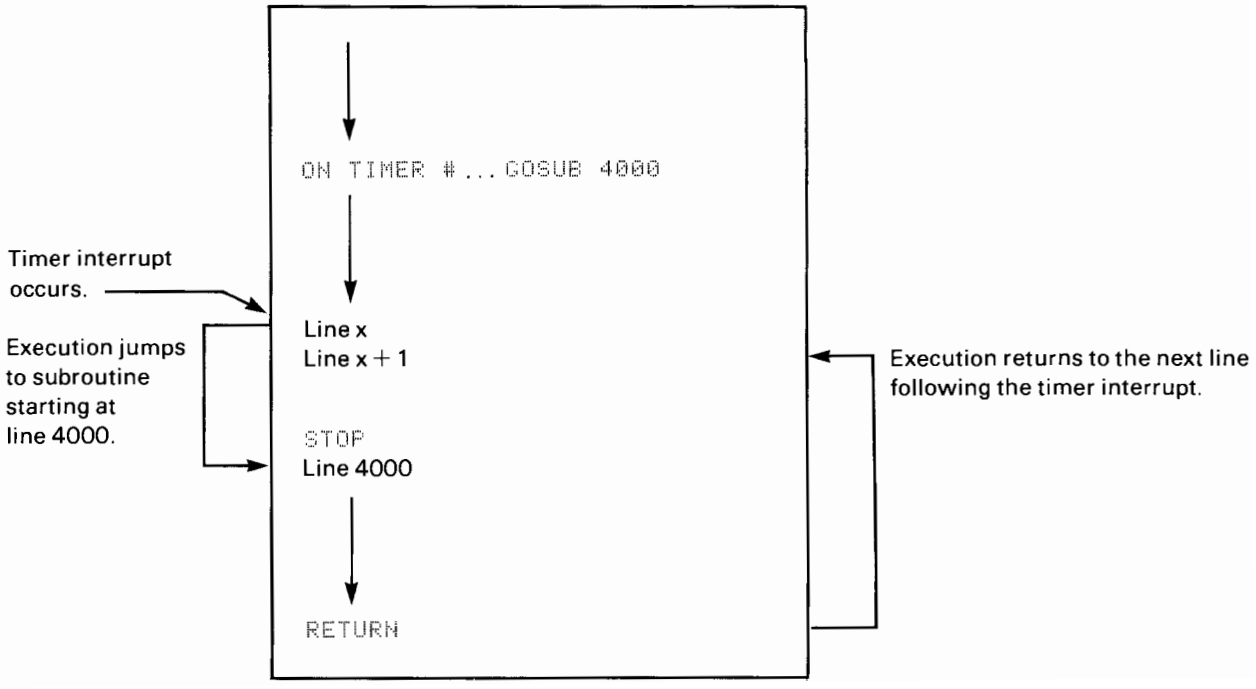
Execution continues as it does after any other unconditional branch.

You shouldn't include another statement after a GOTO statement as an ON TIMER # instruction because it won't ever be executed.

```
ON TIMER # 5, 30 GOTO 500 @ DISP 'Go'
                Executed    Ignored
```

The ON TIMER #...GOTO option allows no provision for returning to the place in the program where the timer interrupt occurred.

GOSUB. Use a GOSUB instruction in an ON TIMER # statement to return execution to the point in the program where the interrupt occurred—that is, the line following the timer interrupt—as soon as the timer subroutine finishes executing.



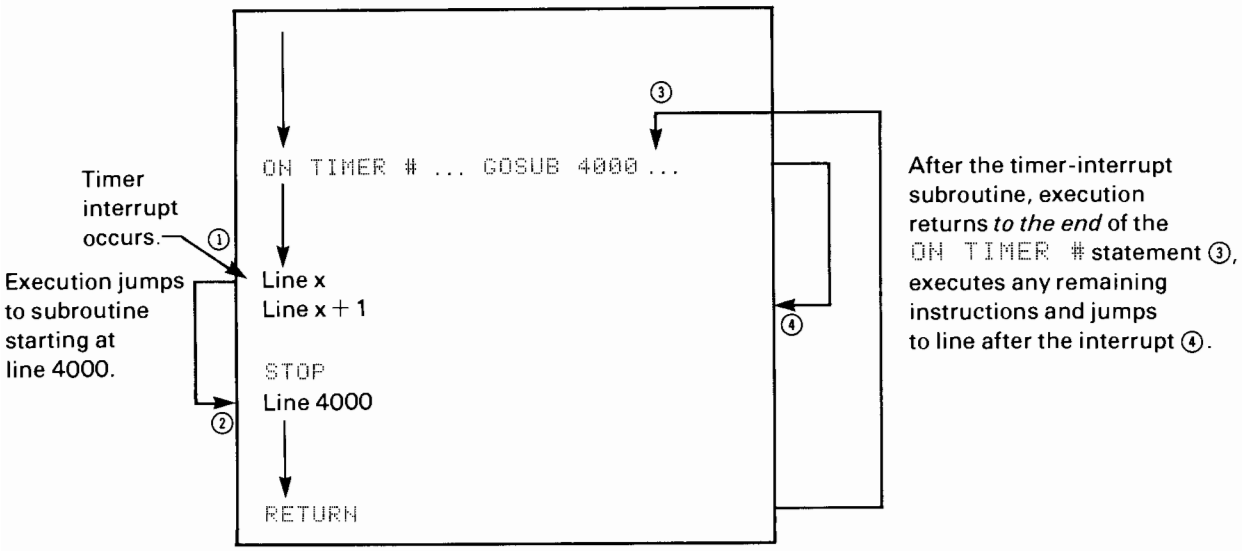
When RETURN is executed as a result of an ON TIMER #...GOSUB statement, program execution returns to the line after the interrupt.

Note what will happen if you include other instructions in the ON TIMER # statement *after* the GOSUB statement.

```
ON TIMER # 5, 30 GOSUB 4000 @ DISP 'Here'
```

Executed first — Executed after the subroutine

Visually, here's what happens:



That is, when RETURN is executed as a result of the ON TIMER #...GOSUB statement, program execution returns to the ON TIMER # statement, looks for further instructions, executes them (if any), and *then* returns to the line after the interrupt.

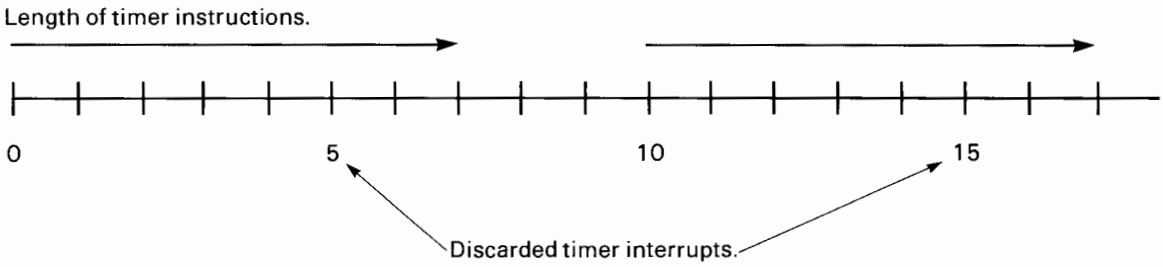
More About Timers

Timer interrupt routines can be nested in the same way as subroutines. This means that if one timer interrupt routine is in progress when another timer interrupt comes due, the unexecuted portion of the first timer routine will be held as a pending subroutine while the second timer interrupt is executed. When the second timer routine is completed, the first timer routine will be resumed.

Error 49—GOSUB overflow—will occur if a program attempts to nest timer interrupts more than 255 levels deep. If timers are interrupting so fast that they system can't get anything else done, an unintentional timer nesting may occur that can eventually exceed the 255 level maximum and result in GOSUB overflow.

Timer routines can't include ON TIMER # declarations. If you attempt to insert an ON TIMER # declaration into a timer routine, error 86—illegal context—will result.

A timer will not interrupt itself. For example, if a timer is supposed to interrupt every 5 seconds but the execution of the ON TIMER # instructions requires 7 seconds, then subsequent interrupts from the same timer will be ignored until the entire set of instructions is completed.



Although the timer comes due every 5 seconds, the timer routine will be executed only every 10 seconds.

Each timer interrupt is actually executed as a subroutine; that is, each ON TIMER # statement creates a pending subroutine condition that is completed when all the timer instructions have been executed. It is the pending subroutine that enables execution to continue at the statement following the one being executed when the timer comes due. No RETURN statement is necessary because each ON TIMER # statement includes an invisible RETURN that is executed after the final timer instruction. You may explicitly write a RETURN in an ON TIMER # statement that will function identically to the hidden RETURN.

Example:

```
100 ON TIMER # 1,30 BEEP@RETURN
```

The RETURN statement completes the timer instructions. Including the RETURN is optional.

A similar pending subroutine condition is created when program execution is transferred to an ON ERROR statement (section 17, page 259).

If a program timer comes due during an INPUT statement (before you have pressed [RTN]), then the timer instructions will be executed *after* the INPUT variables have been assigned their values from the keyboard.

Computer
Museum

Arrays, Strings, and User-Defined Functions

Contents

Introduction 192
Setting the Lower Bound (OPTION BASE) 193
Declaring Arrays 193
  Array Dimensions (DIM) 194
  Type Declaration Statements (REAL, SHORT, INTEGER) 194
Assigning Values to Arrays (LET, INPUT, FOR-NEXT) 195
String Expressions 196
  Substrings 197
  Manipulating Strings 197
String Functions 198
  The Length Function (LEN) 199
  The Position Function (POS) 200
  Converting Strings to Numbers (VAL) 200
  Converting Numbers to Strings (STR$) 201
  Converting Lowercase to Uppercase (UPRC$) 201
  Keyboard Control of Programs (KEY$) 202
  The Catalog Function (CAT$) 202
String Expression Comparisons 203
The PUT Statement 204
Simulating String Arrays 205
User-Defined Functions 205
  Single-Line Definitions (DEF FN) 205
  Multiple-Line Functions(DEF FN, LET FN, END DEF) 207
  The ROOTS Program 207
  Other Considerations 208

13

Introduction

An array variable (or simply, an array) is a collection of data items of the same type under one name. The HP-75 allows arrays to have one or two dimensions and to store numeric values of REAL, SHORT, and INTEGER precision.

A one-dimensional array might be thought of as a list, or vertical column, of items. A two-dimensional array (often called a matrix) is like a table of values; there may be several columns, each with several rows.

column	1	2	3	...	c
row 1	a_{11}	a_{12}	a_{13}	...	a_{1c}
2	a_{21}	a_{22}	a_{23}	...	a_{2c}
3	a_{31}	a_{32}	a_{33}	...	a_{3c}
⋮	⋮	⋮	⋮	⋮	⋮
r	a_{r1}	a_{r2}	a_{r3}	...	a_{rc}

A two-dimensional array has $r \times c$ elements, where r is the number of rows and c is the number of columns.

The *subscripts*, i and j, of each element a_{ij} label the row (i) and column (j) locations of the element.

The list of numbers, 117.50, 403.50, 514.79, and 603.48 (ordered one through four), is a group of similar data items (real numbers); we can manipulate the list efficiently as a one-dimensional array. We treat the list as pairs of values and *subscripts*. Subscripts reference the location of values, or *elements*, in the array. If we name the array variable S(), we specify the individual elements of S() with subscripts, for example, S(2) and S(3).

We can use a two-dimensional array to store the values in the following table:

Day of Month	Low Temperature	High Temperature	Precipitation
21	12	20	.51
22	10	17	.02
23	−6	12	.00

This table contains 3 rows and 4 columns for a total of 12 values. The elements of the table must be specified by row *and* column subscripts (*r*,*c*). When subscript numbering begins at 1, the elements are identified as follows:

```
D(1,1)=21    D(1,2)=12    D(1,3)=20    D(1,4)=.51
D(2,1)=22    D(2,2)=10    D(2,3)=17    D(2,4)=.02
D(3,1)=23    D(3,2)=-6    D(3,3)=12    D(3,4)=.00
```

Array names are the same as simple variable names; an array name may be a letter from A through Z, or a letter immediately followed by a digit from 0 through 9. The parentheses following array names distinguish array variables from simple numeric and string variables. The maximum array size is determined by available memory. (Refer to appendix D, page 292.) Non-integer subscripts are rounded to the nearest integer; negative subscripts aren't allowed.

Setting the Lower Bound (OPTION BASE)

The HP-75 assumes that all array subscripts begin at 0 unless you specify otherwise with an OPTION BASE 1 statement.

```
OPTION BASE 1
            0
```

This statement must come before any array variables are referenced in a program. OPTION BASE 1 tells the computer to set the lower bound of all arrays in the current program to 1.

Each OPTION BASE statement is a *local* declaration. It applies to the currently executing program only.

Declaring Arrays

You may declare the size of array and *type* of array elements with four BASIC statements: DIM, REAL, SHORT, and INTEGER. Memory is allocated for arrays and other program variables in declaration statements during program initialization.

If not explicitly declared, an array is dimensioned for REAL values and for subscript upper bounds of 10 when the array name first appears in a program. For example, if the two-dimensional array A(,) doesn't appear in a declaration statement and if the program is set to OPTION BASE 0, then A(,)—when first assigned a value—will be dimensioned to store 11 × 11 REAL precision values, whose subscripts range from 0 through 10.

Note that if you declare a variable, the declaration must occur before it's assigned a value; otherwise, error 35—DIM exist var—will occur. It's good programming practice to declare array variables early in programs for program documentation purposes. A program can have more than one declaration statement, but a variable can be declared only once in the program.

DIM, REAL, SHORT, and INTEGER statements are not allowed in IF...THEN statements and must appear as the last statement in multistatement lines.

Array Dimensions (DIM)

The DIM (*dimension*) statement is used to allocate memory both for REAL precision numeric arrays and for string variables.

```
DIM item [, item...]
```

The dimensioned item can be:

- A numeric array variable, with subscripts enclosed within parentheses.
- A string variable; with the number of characters enclosed within brackets. If undeclared, a string variable is implicitly dimensioned to store 32 characters.

Examples:

```
20 OPTION BASE 1

30 DIM A(100)

40 DIM B(3,2),C$[96]
```

OPTION BASE declarations must appear before *any* array names are referenced in the program.

Declares a one-dimensional array A() of 100 elements: A(1),...A(100).

Declares a two-dimensional array B(,) of 6 elements (3 rows by 2 columns) and a string C$ of 96 characters maximum.

The DIM statement specifies the upper bound of an array and the maximum number of characters that a string variable may have. Specifying an out-of-range subscript for an array variable will cause error 27—invalid subscript; assigning too many characters to a string variable will cause error 42—string too long.

Note that subscripts in DIM statements must be entered as nonnegative integers; otherwise, error 89—bad parameter—will occur.

Type Declaration Statements (REAL, SHORT, INTEGER)

All numeric variables (simple and array) are assumed to be full precision variables (REAL), unless they appear in a type declaration statement.

```
REAL numeric variable [(subscripts)] [,numeric variable [(subscripts)]...]
```

```
SHORT numeric variable [(subscripts)] [,numeric variable [(subscripts)]...]
```

```
INTEGER numeric variable [(subscripts)] [,numeric variable [(subscripts)]...]
```

Refer to section 5, page 80, for declaring simple numeric variables. When applied to numeric arrays, REAL, SHORT, and INTEGER statements specify the precision of the individual array elements.

Examples:

```
10 OPTION BASE 0

20 INTEGER A,B,C(10)

30 SHORT P(20,25),P1,P2


40 REAL X5,D(4,4)
```

Declares the lower bound of all arrays in the program.

Declares variables A and B to be integers; declares and dimensions array C() to 11 integer elements.

Declares and dimensions 546 (21 × 26) short-precision elements for array P(,); declares variables P1, and P2 to be short-precision.

Declares array D(,) and variable X5 to be type REAL.

Note that REAL declarations serve mainly for program documentation; array and simple numeric variables are dimensioned for REAL precision values by default.

Assigning Values to Arrays (LET, INPUT, FOR-NEXT)

Elements of a numeric array are assigned values in the same manner as simple variables: from the keyboard or from within a program. For instance, M(1,2) and A3(4) refer to elements in arrays M(,) and A3(), respectively, and may be assigned values. LET and INPUT statements are commonly used to assign values to array elements.

Example:

```
>a3(4)=45
```

Pressing RTN assigns the 5th element of array A3() the value 45 (assuming OPTION BASE 0).

This is an implied LET statement executed from the keyboard. It causes the computer to dimension array A3() to hold 11 REAL precision elements (subscripts 0 through 10) before assigning a value to the 5th element. After being assigned a value, the specified element may be used in the same ways as a simple numeric variable.

Example:

```
>a3(4)/5
```

```
 9
```

Using an element of A3() for a keyboard calculation.

FOR-NEXT loops provide a means of controlling array elements.

Example:

```
10 OPTION BASE 1
20 DIM A(5)
30 FOR I=1 TO 5
40 INPUT A(I)
50 NEXT I
60 FOR J=1 TO 5
70 DISP 'A(';J;')=';A(J)
80 NEXT J
```

Assigns each array element individually.

Displays the array elements.

Two-dimensional arrays are often manipulated by nested FOR-NEXT loops. The following example initializes a three-row by five-column array of integers by assigning all 15 elements a value of 0.

Example:

```
10 OPTION BASE 1
20 INTEGER K(3,5)
30 FOR I=1 TO 3
40 FOR J=1 TO 5
50 K(I,J)=0
60 NEXT J
70 NEXT I
80 STOP
```

Nested FOR-NEXT loops. All elements of row 1 are set to zero, then all elements of row 2, etc.

The STOP statement is scheduled so that you can check array values from the keyboard before the program is deallocated and the values are lost.

Example:

```
>K(3,5)
```

```
 0
```

To display the value of the specified element.

DATA statements may be used to store array values within a BASIC file so that a program may read and assign the values directly. Refer to section 14, Storing and Retrieving Data.

String Expressions

A string expression, or character string, is a group of characters that may be manipulated as a single unit. The simplest form of a string expression is text within quotes. This is called a quoted string, a literal string, or a string constant.

The forms that string expressions can take are:

- Quoted string, limited to characters that can be displayed directly from the keyboard (including commas and spaces but excluding the quotation marks that delimit the text).
- String variable, a name representing a location in memory that holds character information.
- Substring, a portion of a string variable.
- String function, an operation that returns character information.
- Any concatenation of the above, using the & operator.

As with numeric expressions, a string expression may be enclosed in parentheses if necessary.

A string variable is implicitly dimensioned to store 32 characters, unless the size of a string variable is specified in a DIM statement.

Example:

```
10 DIM A$[15], F$[28], H$[1000]
```

Dimensions A$ for 15 characters, F$ for 28 characters, and H$ for 1000 characters.

Brackets (not parentheses) must enclose the number of characters to be included in the string variable.

Substrings

Substrings, or portions of string variables, are specified by their beginning character position or their beginning and ending character positions. A subscript may be any numeric expression that rounds to a positive value not larger than the dimensioned size of the string and not larger than the actual length of the string.

To specify a single character of a string, use the form A$[I,I]. To specify the trailing part of a string, use the form A$[I].

Example:

```
200 A$[2,5]='Altogether'

220 DISP A$[1,Y]
240 A$[20]=B$
```

Assigns positions 2 through 5 of A$ the characters Alto.

Displays the 1st through *y*th character of A$.

Assigns the end of A$, from position 20, the characters of B$.

Manipulating Strings

The HP-75 allows you to modify string variables and substrings in a variety of ways. Use assignment statements to replace characters in string variables.

Example:

```
>C$='HI'@ D$='LOW' @ DISP C$,D$
```

```
HI                    LOW
```

The initial assignments of C$ and D$.

```
>C$=D$ @ DISP C$,D$
```

```
LOW                   LOW
```

C$ is assigned the characters in D$.

When a string variable is assigned an initial value, the HP-75 reserves memory for the dimensioned or default string length. Thereafter, it keeps track of the current string length. Redefining the string variable updates the current length and revises the affected characters. Redefining a substring increases the current length if necessary and revises only the specified characters. You can replace any part of a string variable with all or part of another string expression. Specify the subscripts of the characters to be changed in the variable.

Example:

```
>h$='stick'
```

Assigns an initial value to H$.

```
>h$[1,2]='th'
```

Changes the characters in H$ to thick.

If characters added to a string are not contiguous (in other words, some middle character positions are left unassigned in a string variable), spaces will fill the unassigned positions.

Example:

```
>H$[7,9]='ly' @ disp h$
```

```
thick ly
```

A blank is inserted in the variable.

Characters in the replacement string are entered in the string variable from left to right up to the limit allowed by the dimensioned size of the variable or by the variable subscripts.

Note that a null string is specified when the second subscript is one less than the first subscript. For example, H$[5,4] has the same value as ''. Error 42—string too long—will occur if the first subscript is greater than the second by more than 1.

Be careful when adding characters to a string when the characters are not contiguous—previously defined characters may still be present.

Example:

```
 10 A$='123456789'
•20 DISP A$
 30 A$='X'
•40 DISP A$
 50 A$[5]='Y'
•60 DISP A$
```

Displays 123456789.

Displays X.

Displays X234Y. The middle characters in variable A$ were never changed.

To avoid confusion, delete the end of the string as follows:

```
25 A$[2]=''
```

Replaces characters 2 through the end of A$ with null characters, reducing the string length.

Now when you run the program, the string in line 60 will be displayed as expected:

```
X   Y
```

String Functions

Seven functions allow you a great deal of control over character strings. The arguments S$ and S1$ may be quoted strings, string variables, or any other string expression.

Function and Argument	Returns
LEN(S$)	The current length of S$.
POS(S$,S1$)	The position of S1$ in S$.
VAL(S$)	The numeric value represented by S$ where S$ may consist of digits, decimal, and exponent.
STR$(*numeric expression*)	The character string equivalent of the numeric expression.
UPRC$(S$)	The uppercase equivalent of S$.
KEY$	The character associated with the currently depressed key or keystroke combination, or the null string if no key is depressed.
CAT$(*file number*)	The catalog entry of the specified file, or the null string if no catalog entry exists for the specified file number.

Arguments must be enclosed between parentheses. Technically, LEN, POS, and VAL are numeric functions because they return numeric values. They are included here because they make many string manipulations possible.

The Length Function (LEN)

The LEN function returns the current number of characters in a string expression. Note that a string variable isn't always "full"; that is, the length isn't necessarily the dimensioned size.

Example:

```
>len('987')
```

```
3
```

Length of the quoted string.

Example: Write a program that will let you enter a character string of up to 40 characters in length. Then, using the LEN function, fill a variable with the characters in reverse order. For instance, if you input CAT, the program should display TAC.

```
10 DIM W$[40],R$[40]

20 R$=''
30 INPUT 'word? ';W$
40 FOR I=LEN(W$) TO 1 STEP -1

50 R$=R$&W$[I,I]

60 NEXT I
70 DISP R$
```

Dimensions the string variable to be a maximum of 40 characters long.
Initializes R$ to the null string.
Uses a quoted INPUT prompt.
Uses length of word for loop counter and counts in reverse order.
With the string concatenator, adds characters to variable R$ in reverse order.
Defines end of FOR-NEXT loop.
Displays reverse word.

After you enter the program, try spelling some words backwards!

```
word? 
                                PRGM
```

Type `international` RTN.

```
lanoitanretni
```

Leaves the result in the display.

After a string has been modified, LEN may return unexpected results.

Example:

```
 10 A$='AND'
•20 DISP LEN(A$)
 30 A$[3]='T'
•40 DISP A$
•50 DISP LEN(A$)
```

Displays 3.

Displays ANT.
Displays 32, the default size of A$.

The length of A$ has increased to 32 because A$[3] (in line 30) is an open-ended specifier. This will happen whenever open-ended specifiers are used to replace a portion of the string including the last character. To avoid confusion, use a full specifier, A$[3,3].

The Position Function (POS)

The POS function determines the position of a string within a string.

If the second string is contained within the first, the POS function returns the position of the first character of the second string within the first string. If the second string is not contained within the first string, or if the second string is the null string, the value returned by the function is zero. If the second string occurs in more than one place within the first string, only the first occurrence is given by the function.

Examples:

```
>pos('ababc1234','123')
```

Finds position of second string in first string.

```
 6
```

Second string begins at 6th character position.

```
>a$='composer' @ b$='pose'
```

Assigns values to two variables.

```
>pos(a$,b$)
```

```
 4
```

Returns the beginning "index" of the substring.

The POS function may be used to insert and remove string information from large string variables.

Converting Strings to Numbers (VAL)

Normally, the characters in a string are not recognized as numeric data and can't be used in numeric calculations. The reason is that you usually want the string to be quoted literally, character for character. With the VAL function, the numeric value of a string of digits, including sign, decimal point, and exponent, can be used in calculations.

Examples:

```
>val('-3.3e12')
```

Entering a quoted string.

```
-3.3E12
```

This is a number, not a string, and can be assigned to a numeric variable.

```
100 INPUT A$ @ X=VAL(A$)
```

The INPUT statement accepts a string (presumably of digits). Then variable X is assigned the equivalent numeric value.

A similar use of the VAL function is made in the CHECK program (page 156).

The first character in the string argument must be a digit, a plus or minus sign, a decimal point, or a space. A leading plus sign and leading spaces are ignored; a leading minus sign is taken into account. The remaining characters in the string or substring must be digits, a decimal point, or an E. An E character after a numeric and followed by digits (including sign) is interpreted as an exponent of 10.

The number returned by VAL will be expressed in floating point or exponential notation, depending on its magnitude (page 73).

The string argument can contain more than one number. All contiguous numerics are considered a part of the same number until a nonnumeric character is reached in the string.

Example:

```
>val('3333rtrt6666')
```

```
3333
```

The VAL function converts the string to a number until it reaches a nonnumeric character.

You can convert another numeric in a string variable besides the first by specifying its subscripts. For example, if N$ contains the quoted string above, then the second numeric value can be specified by VAL(N$[9,12]).

Converting Numbers to Strings (STR$)

The STR$ function is nearly the inverse of the VAL function. With the STR$ function, you can convert a number to a string representation of the number.

Example:

```
>v$=str$(120) @ disp v$&' pages'
```

```
120 pages
```

V$ contains the characters 120 and may be used in other string expressions.

STR$ expresses the number in floating point or in exponential notation (where necessary), including minus sign, decimal point, and exponent. Note that the string is output with no leading or trailing blank.

Converting Lowercase to Uppercase (UPRC$)

The UPRC$ function allows you to convert a string with lowercase letters to a string composed of all uppercase letters.

Example:

```
>m$='**"An anniversary"**'
```

Assigns M$ a combination of letters and symbols.

```
>uprc$(m$)
```

```
**"AN ANNIVERSARY"**
```

All lowercase letters have been converted.

Lowercase letters have different decimal codes than uppercase letters. The uppercase function allows strings to be compared without regard to upper and lowercase. For example, part of a program might be:

```
:
30 INPUT A$
40 IF UPRC$(A$[1,1])='Y' THEN 80
:
```

User may enter Y, y, yes, YES, etc., and the program will branch to statement 80.

Note that UPRC$ has no effect on underlined lowercase letters (decimal codes 225 through 250).

Keyboard Control of Programs (KEY$)

Although the keyboard is normally disabled during program execution, the HP-75 continually monitors whether keys are being pressed. The KEY$ function returns the character associated with any pressed key or keystroke combination, thereby allowing "live" keyboard branching.

KEY$ returns a single character of information. It returns the character of any key still pending in the "key waiting buffer" (described under "The PUT Statement," page 204). If no key is pending, then the null string is returned. Executing KEY$ removes the pending key from the key buffer.

It's possible to put program execution in an "idle loop", causing the HP-75 to wait for a specific keystroke.

```
90 WIDTH 32
100 K$=KEY$ @ IF K$='' THEN 100
110 DISP 'Character= ';K$;' Code
=';NUM(K$);
120 IF K$="T" THEN DISP TIME$
130 IF K$=CHR$(136) THEN DISP DA
TE$
140 GOTO 100
```

If no key is depressed at the beginning of this line, then K$ equals the null string and execution begins again at the beginning of the line.

Displays the character and decimal code associated with the pressed key or keystroke.

Checks for a specific keystroke, [SHIFT] [T].

Checks for the [I/R] key.

Returns execution to the idle loop.

Press [RUN] to run the program. Then press unshifted [A]:

```
Character= a Code= 97
```

Displays the character and decimal code of unshifted [A].

Now try the [I/R] key:

```
Character= _ Code=136 02/15/83
```

The date is displayed, as specified. IF...THEN statements are useful for determining which key has been pressed.

KEY$ makes possible a variety of keyboard interactions. You can use KEY$ to begin *any* programmable operation, such as branching to a completely new routine, when a designated key or keystroke is pressed. The PAYATTN program (page 170) and NAMELIST program (page 213) included in the Owner's Pac make extensive use of KEY$ branching routines.

Note that to interrupt a KEY$ idle loop, it may be necessary to press the [ATTN] key twice in a row. The first time pressed, [ATTN] may be interpreted as an ordinary key and may cause KEY$ to return ▲, decimal code 128, instead of interrupting program execution.

The Catalog Function (CAT$)

The CAT$ function enables a program to access the complete system catalog for name, type, size, time, and date information. CAT$ operates on numeric arguments to return 32-character strings.

Files are specified by their order in memory. The higher the argument, the older the file. For example, CAT$(1) specifies the most recently created file.

CAT$(0) specifies the current edit file. This may or may not be the same as CAT$(1), the newest file in memory.

CAT$ of a negative number specifies the file in which the program pointer is currently located.

CAT$ rounds arguments to integer values. If an argument is greater than the number of files in memory, then CAT$ of that argument returns the null string.

Example: Write a PRINTALL program that prints the catalog entry of every file in memory followed by a listing of the file, from newest file to oldest file. If the appt and keys files exist, print only their catalog entries. First, create the program file by typing edit 'printall' [RTN]. Then enter the following steps:

```
 10 PWIDTH 32
•20 N=1
•30 C$=CAT$(N)
•40 IF C$='' THEN STOP
•50 IF C$[1,4]='appt' OR C$[1,4]=
 'keys' THEN PRINT C$ @ GOTO 80
•60 PRINT C$ @ PLIST C$

•70 PRINT
 80 N=N+1 @ GOTO 30
```

To specify the newest file in memory.
Fills C$ with the characters of a catalog entry.
Stops the program if the supply of files is exhausted.
Checks for the appt and keys files and prints their catalog entries only.
Prints the catalog entry of any other file and then print-lists the file.
Prints a blank line between files.

If the HP-75 display window is the current PRINTER IS device and if the newest file in memory is PRINTALL, then the first file to be print-listed will be the PRINTALL file itself. Press [RUN]:

```
PRINTALL    B   168 09:40 02/15/83
10 PWIDTH 32
20 N=1
⋮
```

The catalog entry of PRINTALL reflects its initialized condition because it is the currently executing program.
And so on, through memory.

Each catalog entry includes a number of embedded blanks between filename and file type, file type and file length, etc. Because the first blank after a filename terminates the filename specifier, the entire catalog entry string (C$ in the example) may be used as the parameter of the PLIST command and of other file commands, such as PURGE and RENAME.

Note: The names of the workfile, keys, and appt files may not be specified by string variables in file commands. For example, if F$ is assigned the value 'keys', then PLIST F$ will cause the HP-75 to search memory for a user-file named KEYS rather than the system keys file.

String Expression Comparisons

The relational operators may be used to compare character strings. The relational operators are =, >, >=, <, <=, <>, and #.

Example:

```
'A'<'B'
```

```
 1
```

The expression is "true" so it evaluates to 1.

The basis of comparison in the preceding example is the decimal code of character A (65) versus the decimal code of B (66).

Two strings are considered equal if they contain exactly the same characters in the same order. For inequalities, two strings are compared character by character until a difference is found. If one string ends before a difference is found, the shorter string is considered the lesser. If a difference is found, the decimal codes of the *two differing characters* are used to determine which string is the lesser.

Example:

```
'AAbcde'<'ABb'█
```

```
1
```

The second character (A vs. B) determines the outcome of the test.

The PUT Statement

The PUT statement enables programs to use key and keystroke information without the actual pressing of keys.

PUT *one-character string expression*

The HP-75 has a "key waiting" buffer that stores the display character associated with the currently depressed key or keystroke. For example, type wait 10 [RTN] to cause the HP-75 to wait for 10 seconds and then press keys [Q], [W], [E], [R], [T], and [Y] while the HP-75 is waiting.

```
>q█
```

Afterwards, the display shows that only the first character in the sequence was stored in the key waiting buffer.

As another example, when you press [SHIFT] [LOCK] to lock the keyboard in uppercase, the HP-75 enters the ⌟ character, decimal code 172 in the key waiting buffer.

The PUT statement causes a *program* to put a character in the key waiting buffer. The effect is as though the key or keystroke had actually been pressed.

Examples:

```
10 PUT '⌟'


20 INPUT 'Your name: ';A$
⋮
990 PUT CHR$(129)

1000 END
```

The ⌟ may be typed by pressing [SHIFT] [I/R] and then [SHIFT] [LOCK]. This statement stores the [SHIFT] [LOCK] keystroke sequence in the key waiting buffer. The keyboard is locked in uppercase.

Puts ⌟ in the key waiting buffer, the same as pressing [TIME].
The HP-75 is switched to TIME mode.

Note that you may use a decimal code and the CHR$ function to specify a key or keystroke, as in statement 990 above. This is useful to avoid unexpected results when listing the program on an external printer.

Only *one* key character may be put into the key waiting buffer at a time. The key code in the key waiting buffer is lost whenever an end-of-line is generated by an output operation such as DISP, LIST, or CAT. Consequently, it is best to place a PUT statement immediately before a statement that enables the keyboard, such as INPUT, STOP, or END. Note that specifying the [ATTN] key (decimal code 128) in a PUT statement interrupts execution, the same as pressing [ATTN].

PUT causes the *current* definition of the specified key to be accessed. For example, if [SHIFT] [Z] is redefined to execute a PRINT statement, then PUT 'Z' will cause the PRINT statement to be executed.

Simulating String Arrays

The HP-75 substring capabilities allow the simulation of one-dimensional string array variables. One method is to use one large string variable as the array itself and then substrings of uniform lengths, or "fields," as the string array elements.

Example:

```
•10 DIM A$[200]
 20 FOR I=1 TO 5
 30 DISP 'String element #';I;
•40 INPUT A$[I*25-24,I*25]

 50 NEXT I
 60 FOR I=5 TO 1 STEP-1
•70 DISP I;A$[(I-1)*25+1,I*25]

 80 NEXT I
```

Dimensions the string variable.

The first element (I equal to 1) occupies character positions 1 through 25; the second element, positions 26 through 50; and so on.

Displays the index and value of each field of the "array."

The choice of 25 as the multiplier of the string variable subscripts causes a field of 25 characters to be reserved for each substring.

When run, the program accepts five string elements and assigns them to the fields in A$. After the first FOR-NEXT loop, the elements are displayed in reverse order in the second FOR-NEXT loop.

The POS function may also be used to locate the indices of substrings within simulated string arrays.

User-Defined Functions

The HP-75 allows you to define your own functions within a program and then use them in the same way as system functions while the program is executing. User-defined functions may operate on zero to 36 arguments, or *parameters*, to return single values. *Numeric* functions return numeric values; *string* functions return string values.

Single-Line Definitions (DEF FN)

The DEF FN statement is used as the first statement of all function definitions. It may be used alone in a single-line definition:

```
DEF FN numeric variable name [(parameter [,parameter...])] = numeric expression
DEF FN string variable name [(parameter [,parameter...])] = string expression
```

A numeric function name consists of the letters FN followed by a numeric variable name (for example, FNS and FNA3). A string function name consists of FN followed by a string variable name (for example, FNZ$ and FNZ1$).

The parameter list may be any combination of simple numeric or string variables, separated by commas. Array names are not allowed.

Examples:

```
20 DEF FNR(X,Y) = SQR(X^2+Y^2)
30 DEF FNH(D) = (EXP(1)^D+EXP(1
)^(-D))/2
40 DEF FNS$(I) = A$[I*5-4,I*5]
50 DEF FNE$(C,R) = CHR$(27)&'%'&
CHR$(C)&CHR$(R)
```

Computes $\sqrt{x^2+y^2}$.
Computes the hyperbolic cosine of the parameter.

Returns a substring of string A$.
Returns an escape code to position the cursor.

A function definition can appear anywhere in the program, before or after the function is referenced. A common practice is to place function definitions at the beginning of main programs.

The functions defined above are referenced in the following segment:

```
•60 A,B=30 @ C=FNR(A,B)


 70 DISP A;B;C
 80 X=1 @ Z=FNH(X)
 90 DISP X;Z
•100 A$='1234567890'

 110 DISP FNS$(1)
 120 DISP FNE$(16,15);'HERE'
```

Parameter values and program execution are passed to the function whenever the function is referenced in the main program.

Although A$ is a main program variable, it is available to FNS$.

When run, the program displays four lines:

```
 30  30  42.4264068712
 1  1.54308063482
12345
                HERE
```

Square root of the sum of squares.
Hyperbolic cosine of 1.
First five characters in A$.
The specified string, beginning at column 16.

The main program may pass the values of constants, variables, array elements, and expressions to user-defined functions through the parameter list in the main program. In addition, all main program variables are *global*, that is, available to all user-defined functions. For example, A$, a main program string variable, is accessed directly by FNS$.

Some restrictions:

- User-defined functions may not be defined or used from the keyboard.
- Variables appearing only in the function definition parameter list are *local* to the function and remain unknown to the main program. Parameter variables can only be accessed by instructions that are physically between the DEF FN and END FN lines.
- A DEF FN statement must be the first and only statement in a program line.
- The length of a string parameter passed between a main program and a function defaults to 32 characters. But you can declare larger strings in the parameter list of a DEF FN statement (page 209).
- Multiline string functions return 32-character strings. A longer string causes error 42—string too long—to occur.
- Single-line string functions can return strings of any length that will fit on a line.
- The number of parameters a function can accept is limited by the line length of the DEF FN statement. The upper limit is 36 parameters.

- A function may not be defined *recursively*—in terms of itself. That is, the function name may not appear on the right side of the equals sign in the definition.
- Each function must have a unique name.

A function need not have an argument to return a value. (Recall the PI, EPS, and INF system functions.)

Examples:

```
1000 DEF FNW$='Report on Work
Environment'
1010 DEF FNP=6.625E-27
```

FNW$ will return a string constant.

FNP will return Planck's constant.

Multiple-Line Functions (DEF FN, LET FN, END DEF)

You can define more sophisiticated functions using a block, or series, of program statements for the function definition. There are three basics parts to multiple-line function definitions. The first statement must be a DEF FN statement.

```
DEF FN numeric variable name [(parameter [,parameter...])]
DEF FN string variable name [(parameter [,parameter...])]
```

Note that no value is assigned the function in this form of the DEF FN statement. At least one of the statements in the function definition should assign the function name a value.

```
[LET] FN numeric variable name = numeric expression
[LET] FN string variable name = string expression
```

Note that the LET keyword is optional in the statement and that the parameter list is omitted. The assignment statement transfers the computed value of the function to the function name in the main program. More than one function assignment statement may occur in the same function definition.

The last statement of a multiple-line function must be the END DEF (*end definition*) statement.

```
END DEF
```

The END DEF statement must appear as the last statement in a program line and isn't allowed after THEN.

Any number of statements can be included between the DEF FN and END DEF statements. But one of these statements should assign the final value of the function to the function name.

The ROOTS Program

The following ROOTS program computes the real roots of a quadratic equation of the form:

$$ax^2 + bx + c = 0.$$

The roots will be $\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$, where $a \neq 0$ and $b^2 \geq 4ac$.

```
 10 ! Quadratic Program
•20 DEF FNQ(X,Y,Z)
•30 IF X=0 OR Y^2<4*X*Z THEN FNQ=
 -1 ELSE FNQ=SQR(Y^2-4*X*Z)
•40 END DEF
 50 ! BEGIN Quadratic
 60 INPUT 'Enter a,b,c: ';A,B,C
 70 IF FNQ(A,B,C)=-1 THEN DISP 'P
 lease try again.' @ STOP
•80 R1=(-B+FNQ(A,B,C))/(2*A)
•90 R2=(-B-FNQ(A,B,C))/(2*A)
 100 DISP 'Root 1=';R1
 110 DISP 'Root 2=';R2;'  Check:'
 120 DISP 'ax^2 + bx + c=';
•130 DISP A*R1^2+B*R1+C
•140 DISP 'and';A*R2^2+B*R2+C
 150 END
```

The function FNQ checks for legal values and then computes $\sqrt{b^2-4ac}$.

FNQ is used to compute roots 1 and 2.

Checks the roots of the original equation.

Supply appropriate values when running the program. Example:

```
Enter a,b,c: ■
                    PRGM
```

Type 2,3,-1 [RTN].

The program displays four lines according to the current DELAY setting.

```
Root 1= .280776406405
Root 2=-1.78077640641  Check:
ax^2 + bx + c= 0
and .000000000003
```

R1 and R2 are good approximations.

Other Considerations

The purpose of user-defined functions is to process main program values in order to return single values through the function name; it's best to restrict user-defined functions to this purpose. For example, if a function contains a PRINT or DISP statement and is itself referenced in a main program PRINT or DISP list, you may not get the output you expect. If you branch in and out of a function definition, you may cause unexpected errors.

It's good practice to use different variable names for function parameters than for main program variables so you won't confuse the two.

All main program *variables* are within the scope of user-defined functions; that is, a function may use any main program variable value by referencing it. However, changing the value in the function will cause a corresponding change in the main program; you should be aware of any undesirable "side effects" that may result.

The parameters of a user-defined function can't be accessed or changed from the keyboard, unlike main program variables. For example, you can't assign a different value to a function parameter while the program is halted by a STOP statement.

The length of string parameters passed to a function defaults to 32 characters. You can specify larger strings in the function definition by enclosing the length within brackets following the string name. For instance:

```
4000 DEF FNS$(A$,B$[8],C$[96])
```

Allocates memory for string parameters of 32 (A$), 8 (B$), and 96 (C$) characters.

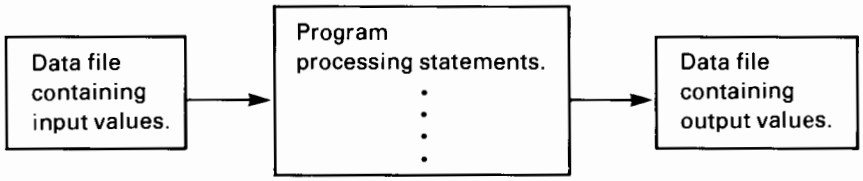
Storing and Retrieving Data

Contents

Introduction 210
Reading Data Within a Program (DATA, READ) 210
Rereading Data (RESTORE) 213
The NAMELIST Program 213
Creating and Accessing Data Files (ASSIGN #) 216
Storing Data in a File (PRINT #) 217
Reading Data From a File (READ #) 219
Closing Data Files 219
Serial and Random Access 220
Special Forms of PRINT # and READ # 221
Moving the Data Pointer (RESTORE #) 221
Statement Summary 222
Storing and Retrieving Arrays 223
Accessing Text Files 224
The FINDIT Program 227
Long Data Lines 228

14 Introduction

The internal file structure of the HP-75 enables you to create a variety of input and output operations. For example, a program may operate on the values in an input file and print the results to an output file.



Any BASIC file may store data values mixed with other programming statements; for convenience, we use the term *data file* to refer to a BASIC file that consists entirely of data values. In this section you will learn how to store and retrieve data within a program file using the DATA, READ, and RESTORE statements and how to create and use *data files* by using the ASSIGN #, PRINT #, READ #, and RESTORE # statements. The DATA statement stores numeric and character information for later use. The PRINT # statement enables a program to put DATA statements into other BASIC and text files in memory. The RESTORE and RESTORE # statements enable a program to control which data will be read from or written to a program or data file.

Reading Data Within a Program (DATA, READ)

The DATA statement contains a list of the numbers and characters that are to be stored within a BASIC file.

```
DATA number or text [, number or text...]
```

DATA items must be separated by commas. Two consecutive commas in a DATA statement indicate a null string, the equivalent of '', consisting of zero characters.

Examples:

```
100 DATA 1900,2100,2200
```

DATA items may be numbers,

```
110 DATA functional,'policies'
```

unquoted strings, quoted strings (with ' ' or " " as delimiters),

```
120 DATA 1900,function,2200,,''
```

or a combination. The last two items are null strings.

Each DATA statement entered from the keyboard may hold one display line of information—up to 96 characters, including line number, spaces, DATA keyword, data items, commas, and two positions reserved for prompt and cursor. (Refer also to Long Data Lines, page 228.) The values in the DATA statement are stored in the BASIC file until the file is purged or the DATA statement itself is changed or deleted.

Notice that DATA items must be *constants*. Variable names and keywords are interpreted as straight text. Numeric constants may include a leading plus or minus sign and a decimal point and may be expressed in exponential notation with exponent E, e.g., -2.819E-29. Very large and very small numbers are converted to exponential notation in the DATA statement when RTN is pressed.

Here are some restrictions on *unquoted* text:

- Leading and trailing blanks are ignored.
- Commas in the text are interpreted as delimiters between parameters.
- If the first character is a digit, a +, or a -,the parameter is evaluated as a number.

These restrictions don't apply to quoted text.

Example:

```
122 DATA '2b,',' or ',not,' 2b'
```

Use quote marks for strings that have leading digits, commas, or leading and trailing blanks.

DATA statements are not allowed after THEN or ELSE. Also, DATA statements should not appear in multistatement lines because the @ symbol after a DATA keyword will be interpreted as an ordinary text character instead of a statement concatenator; if the DATA statement *follows* another statement in the same line, the HP-75 will not be able to access the DATA items. The HP-75 reports error 88—bad statement—if DATA is executed from the keyboard.

The READ statement specifies the variable whose values are to be assigned from data values within the program itself.

```
READ variable name [,variable name...]
```

The READ list may consist of simple numeric variables, numeric array elements, string variables, and substrings, separated by commas.

Example:

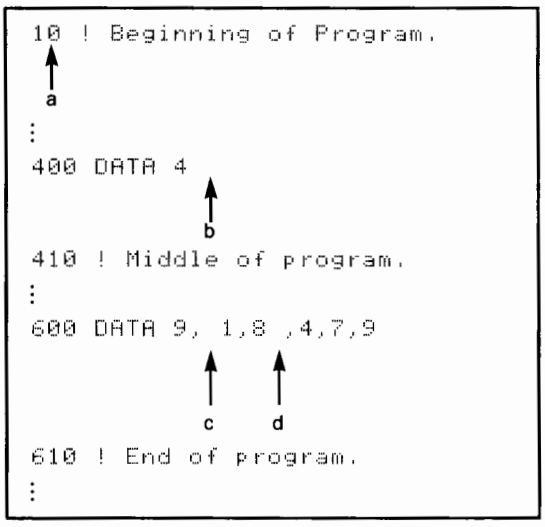
```
210 READ A,B(3),C$,D$[5]
```

This statement assigns values to two numeric and two string variables.

Entire arrays can be read in a single READ statement. Just omit the subscripts within the parentheses, but include the comma for two-dimensional arrays. For example, READ A(),B(,) reads all elements of the one-dimensional array A and the two-dimensional array B.

The operating system uses a file mechanism, called a *data pointer*, to track the DATA item that is to be read next. The leftmost item of the lowest-numbered DATA statement is read first. Afterwards, the data pointer advances one item to the right and continues to do so each time another value is read. After the last value in a DATA statement, the data pointer advances to the first value of the next higher-numbered DATA statement, and so on through the DATA statements.

Example:



a. The data pointer for the program is initially set to the first line of the file.

b. Executing READ X reads the first data item in the file, 4, and leaves the data pointer after that item.

c. Executing READ Y reads the next item in the file, 9, and leaves the data pointer after that item.

d. Executing READ Z,T reads the next two items, 1 and 8, and again repositions the data pointer, and so on.

The important point is that the *order* of DATA items from left to right and from lowest-numbered to highest-numbered DATA statement determines the order of their use.

```
400 DATA 4,9,1,8,4,7,9
```

```
400 DATA 4
600 DATA 9,1,8,4,7,9
```

```
400 DATA 4
600 DATA 9,1,8
800 DATA 4,7
803 DATA 9
```

These three data groups will be accessed identically in sequential, or *serial* reads.

If a program tries to read more values than there are DATA items, then error 34—no data—will occur during the READ statement. READ statements cannot be executed from the keyboard.

Each numeric variable in a READ statement must correspond to a numeric constant in a DATA statement. The precision of a numeric DATA value is determined by the variable's precision in the program (REAL, SHORT, or INTEGER), and rounding will occur if necessary when the numeric variable is assigned the value. String variables in the READ list must be dimensioned properly to accommodate the number of characters read from a DATA statement, or error 42—string too long—will occur and no assignment will be made. If a number (for example, 5.55) is read into a string variable, the program will treat it as character information rather than numeric information.

DATA statements can be placed before or after READ statements. Our examples show DATA statements at the end of programs. DATA statements are ignored during program execution if there are no corresponding READ statements.

Example: The following program reads numbers sequentially from a DATA statement and computes their squares:

```
10 READ A
20 FOR I=1 TO A

30 READ X
40 DISP X; 'squared=';X^2

50 NEXT I
60 DATA 4,1,1,2,3,5,8,13
```

Reads the first number from the DATA list in line 60.
The value of A (4) determines the number of loop iterations.
DATA items are read sequentially.
Variable X inherits the value of the data items, one by one.
A total of four items are read during the loop.
Extra data items (5, 8, and 13) are ignored.

Press RUN:

```
1 squared= 1
1 squared= 1
2 squared= 4
3 squared= 9
```

Changing the value of the first DATA item (4) will change the number of times through the loop. FOR-NEXT loops are an efficient way to access DATA values.

Rereading Data (RESTORE)

The RESTORE statement moves the data pointer either to a specified line of the file or to the lowest-numbered DATA statement in the file (if no line is specified).

```
RESTORE [line number]
```

The *line number* must be an unsigned integer from 0 through 9999. The specified line must begin with a DATA keyword, but need not include any items after the DATA keyword.

Example:

```
10 FOR I=1 TO 8
20 READ A
30 BEEP A,.3
40 NEXT I
•50 RESTORE


60 GOTO 10
70 DATA 130,164,195,219,233
80 DATA 219,195,164
```

Beeps a musical scale.

Repositions the data pointer to read the first value of the lowest-numbered DATA statement (the value 130, in line 70).

RESTORE statements cannot be executed from the keyboard.

The NAMELIST Program

The preceding text describes how to read DATA statements that are embedded in the currently executing program. The NAMELIST program included with your HP-75 uses three powerful statements, ASSIGN #, PRINT #, and READ #, to store and retrieve information in a data file outside the program itself.

When you run the NAMELIST program, you create data files containing lists of names and associated information. You can create an address file or phone directory, a list of birthdays or anniversaries, or any other kind of list in which up to 32 characters of text information is associated with a name.

Find the two magnetic cards for the NAMELIST program and execute the COPY command:

```
>copy card to 'namelist'
```

```
Copy from card:  Align & [RTN]
```

The NAMELIST file is recorded on four card tracks—four passes through the card reader are necessary.

After you've copied the file to memory, the prompt and cursor will reappear. Assuming you've PURGEd, NAMEd, or RENAMEd a nonempty workfile, execute an EDIT command:

```
>edit 'namelist'
```

```
NAMELIST  B 2481 15:53 04/22/82
```

The size and date of the program may vary, depending on your version.

The fully initialized program requires about 3000 bytes of memory. When you run the NAMELIST program, you will be prompted to enter the name you want to associate with the information in the data file.* Press [RUN]:

```
Enter filename?
                              PRGM
```

The filename may be any valid filename either quoted or unquoted.

```
Enter filename? d
                              PRGM
```

We use a single letter.

```
Name List
                              PRGM
```

The message signals the program's readiness to enter, search for, and display information from the D file.

Each entry may consist of a last name, a first name, and up to 32 characters of text. The NAMELIST program uses the KEY$ function (discussed in section 13) to "take over" the keyboard so that the following keys are given special functions while the program is running.

[+]	To add an entry to the file.
[↑] and [↓]	To display other entries, arranged alphabetically. First-name-only entries will be listed ahead of all last-name entries.
[SHIFT] [↑]	To display the first entry of the file.
[SHIFT] [↓]	To display the last entry of the file.
[A] - [Z]	To display the first last-name entry under that initial.
[=]	To search for an entry, given a first name, a last name, or both.
[SHIFT] [=]	Used after [=] to display the next entry having the same first name or last name.

* Later, you may enter the name of an existing file that you want to update.

[/] To cause "split-screen" viewing and listing of entries, with names in left screen and associated information in right screen. Pressed again to restore a compact display format.

[←] and [SHIFT] [←] To view the left screen of an entry.

[→] and [SHIFT] [→] To view the right screen of an entry.

[*] To print-list the entire file on current PRINTER IS devices.

[-] To delete an entry.

[·] To end the NAMELIST program.

You may also interrupt the program by pressing [ATTN]. For our example we will use NAMELIST to create a telephone directory. Press the [+] key now to add the first entry in the D file—William Tell's name and telephone number (111-555-3400, extension 2205).

```
Last Name: Tell
First Name: William
Note: (111) 555-3400 x2205


Tell,William  (111) 555-3400 x2
```

Prompt to enter the last name. Type it and press [RTN].

Prompt to enter the first name.

Prompt to enter the text in the information field (note field). Type in the phone number in a convenient format. Remember that the note field will hold a maximum of 32 characters.

First entry is completed. The distinction between uppercase and lowercase letters is preserved.

Press [→] or [SHIFT] [→] to view the end of the entry:

```
1, William  (111) 555-3400 x2205
                          PRGM
```

The normal, compact format for completed entries.

Press [/] to cause split-screen viewing:

```
Tell,William
                          PRGM
```

The left screen shows the name.

```
(111) 555-3400 x2205
                          PRGM
```

Press [→] or [SHIFT] [→]. The right screen shows the number.

Press [/] to restore the compact display format. Completed entries will remain in the display until you press another key.

The preceding example used all of the fields in the entry. You can bypass fields by pressing [RTN] without typing an entry. Press [+] for another entry.

```
Last Name:
First Name: John
Note: x3634
 John  x3634
```

No entry. Press [RTN] for the next prompt.

Entering John's name.

Enter telephone extension.

Completed entry.

Press [+] once more for a third example:

```
Last Name: Tell
First Name: Helen
Note: 555-2000
Tell, Helen  555-2000
```

Entering Helen Tell's phone number.

While the program is running, use the [↑] and [↓] keys to review the entries. Then press [*] to list the entries. (If you have previously declared an external printer, the entries will be print-listed.) Pressing [/] before you press [*] causes the note field to be separated from the names by a string of periods and to be listed beginning at column 33.

Restrictions on NAMELIST entries:

- Entries can contain any character on the keyboard except a comma.
- Either a first or a last name must be entered for each entry.
- Names are sorted without regard to case; however, embedded blanks may cause a name to be placed ahead of its spelling without blanks.
- A maximum of 32 characters can be entered for any field
- Up to 96 characters total may be typed in an entry, but only the first window of information will be displayed when [←] is pressed and only the last window of information will be displayed when [→] is pressed (assuming compressed display format).

To locate an entry, either press the key corresponding to the first letter of the last name, or press [=] and supply the last name, the first name, or both. The heading of the program, Name List, will be displayed if no match is found. If you find an unwanted entry, press [-] to delete it from the directory. Finally, press [.] or [ATTN] to stop the program.

Now execute a CAT ALL command; the D file, the most recently created file, will appear first in the system catalog.

```
D            B  105 11:05 02/15/83
```

The catalog entry of D, the data file that's been created and accessed by the NAMELIST program.

A listing of D will show the formats used by the NAMELIST program to store the individual entries. Type `list 'd'` [RTN]:

```
1 DATA '','John','x3634'

2 DATA 'Tell','Helen','255-2000'
3 DATA 'Tell','William','(111) 5
55-3300 x2205'
```

A null string ('') represents a first-name-only entry.

The program stores all entries—names and numbers—as quoted strings.

To add more entries to the D file, press [RUN] to restart the program, specify D as the filename, and press [+]. The number of entries in the directory file, as well as the number of directory *files*, is limited only by available memory.

Creating and Accessing Data Files (ASSIGN #)

The ASSIGN # statement is used to create new data files and to make existing data files available for program input and output.

```
ASSIGN # file number TO 'filename'
```

The ASSIGN # statement assigns the *file number* you specify to the *filename* you specify. The file number can be any numeric expression that rounds to an integer from 1 through 9999. This means that each program can address up to 9,999 files, depending on available memory. The quoted filename can be specified by a string expression.

Example:

```
 10 INPUT 'What file? '; A$
•20 ASSIGN # 1 TO A$
 ⋮
```

Assigns the number 1 to the file specified by A$.

When an ASSIGN # statement is executed:

- A new BASIC file of that name is created if the named file doesn't exist already in memory.
- The file is *opened*; that is, a unique data pointer is set to the first line of the file or to line 0 if the file is empty.

About file numbers:

- File numbers are used in all communications between programs and data files.
- There is a one-to-one correspondence between file numbers and data pointers.
- Several data pointers can be associated with the same file through a series of ASSIGN # statements that specify the same file.
- You can also assign file numbers by executing ASSIGN # from the keyboard.
- File numbers are *global*—that is, *any* program can reference the same file by specifying the same file number.
- A file number stays assigned to the same file indefinitely or until the file number appears in another ASSIGN # statement.

Storing Data in a File (PRINT #)

A data file opened by an ASSIGN # statement is ready for data access. Use the PRINT # statement to print values to a particular data file.

```
PRINT # file number [,line number] ; expression [, expression...]
```

Each PRINT # statement causes the HP-75 to move the data pointer to the beginning of a line in the specified data file, to evaluate the expression in the PRINT # list, and to print the values to the file as DATA items.

Here's an example to execute from the keyboard:

```
>assign#2 to 'new'
```

Press [RTN] to create and open the NEW file and to position the data pointer to line 0.

```
>print#2;'start',1,sqr(2)
```

This statement moves the data pointer to line 1 of NEW and prints three values to the file.

Then execute:

```
>list 'new'
```

To list the contents of the file.

```
 1 DATA 'start',1,1.41421356237
```

One string and two numeric constants have been printed to line 1 of the NEW file.

If no line number is specified in the PRINT # statement, then the data pointer is moved from its current location to the next DATA line in the file before the data items are printed. If no DATA line exists after the data pointer, then PRINT # causes a DATA line to be created with a line number one greater than the last line in the file. In the example above, the NEW file is empty when the PRINT # statement is executed, so the PRINT # statement causes the creation of line 1 in the file (one greater than line 0).

A second example:

```
>print#2;'create a line'
```

Pressing RTN causes line 2 to be created in the data file and causes the value of the string to be printed to that line.

If a line number *is* specified in the PRINT # statement, then the DATA items are printed to that line. You may use any numeric expression that rounds to an integer from 0 through 9999 to specify the line number. If the PRINT # statement specifies a nonexistent line, then the line is first created and the values are printed afterward. The following example demonstrates the movement of data pointer #2 as other values are printed to the NEW file.

```
90 C=2 @ B=200 @ A$='there'
100 PRINT # 2, 99; 'here'

110 PRINT # 2; A$
120 PRINT # C, B; 'once'
130 PRINT # C; 'more'
140 PRINT # C; 3*6,A$
```

Initializes three variables.
Creates line 99 of NEW and prints 'here'; leaves the data pointer at the end of the line.
Creates line 100 and prints 'there'.
Creates line 200 and prints 'once'.
Creates line 201 and prints 'more'.
Prints 18 and 'there' in line 202.

After this program has been run, a listing of NEW shows:

```
1 DATA 'start',1,1.4121356237
2 DATA 'create a line'
99 DATA 'here'
100 DATA 'there'
200 DATA 'once'
201 DATA 'more'
202 DATA 18,'there'
```

Data values have been printed to the NEW file at the specified locations.

PRINT # statements position the data pointer initially to the beginning of a DATA statement and may cause existing file lines to be overwritten. After a PRINT # statement, the data pointer is left at the end of the line.

If the PRINT # statement specifies an existing line in the file, it must be a DATA line; otherwise, error 34—no data—will occur, no printing will be done, and the data pointer will be left at the beginning of the specified line.

If an undefined numeric or string variable appears in the PRINT # list, then warning 7—no value—will occur and a zero (if a numeric variable) or a null string (if a string variable) will be printed to the file.

However, if the HP-75 has been set to DEFAULT OFF, then execution will stop and no default value will be printed to the file.

If the last line of the file is 9999 and a PRINT # statement attempts to create a higher-numbered line, then error 90—bad line number—will occur, no printing will be done, and the data pointer will be left at the end of line 9999.

If a PRINT # statement attempts to make a program print to itself, then error 51—PRINT # to runfile—will occur, nothing will be printed, and the program will be deallocated.

Reading Data From a File (READ #)

You can use the READ # statement to read any data value in memory.

```
READ # file number [,line number]; variable [,variable...]
```

The *file number* and *line number* can be any numeric expressions that round to integer values in the proper range, 1 through 9999 and 0 through 9999, respectively. The variables may be simple numeric variables, numeric array elements, string variables, and substrings. The data pointer moves from left to right across the DATA statement as the READ variables are filled from left to right.

Here are several examples based on the PRINT # statements above:

```
900 READ # 2, 200; U$

910 READ # 2; T$[3,6]

920 READ # 2; V9,T$[7,9]

930 READ # 2, 202; C

940 READ # 2, 99; X$
950 DISP V$;T$[3,6];V9;T$[7,9];C
```

Positions the data pointer to line 200 and assigns U$ the value of once.

Assigns a substring of T$ the four characters more from the next data value, in line 201.

Assigns V9 the value 18 and assigns a substring of T$ the value the, both from line 202.

Assigns C the same value, 18, from the same line 202.

Assigns X$ the value here from line 99.

Displays variables.

If a READ # statement causes the data pointer to move past the last DATA item in a file, then error 34—no data—will occur, and the data pointer will be left at the end of the last DATA statement in the file. Similarly, if a line number is specified and the variable list would cause the data pointer to move past the end of that line, an error will occur and the data pointer will remain at the end of the line.

Like ASSIGN # and PRINT # statements, READ # statements may be executed from the keyboard. Note that the line numbers specified in PRINT # and READ # statements will *not* be changed by a RENUMBER command.

Closing Data Files

When you finish using a data file, you can *close* the file with any of three ASSIGN # statements.

```
ASSIGN # file number TO *
ASSIGN # file number TO ''
ASSIGN # file number TO '*'
```

Closing a file means that you've ended the association between file number and filename. It's not necessary to close HP-75 data files; however, there are several reasons for doing so:

- To protect the values in a file from being overwritten.
- To conserve memory. Each open file requires 15 bytes of memory. Closing the file means the HP-75 reclaims that amount of memory.
- To show in your programs when a file operation has been completed.

You may also wish to purge unused data files to conserve memory.

Serial and Random Access

Specifying line numbers in READ # and PRINT # statements is optional, depending on whether you want to address your DATA statements individually—called *random access*—or sequentially—called *serial access*. You don't specify line numbers for *serial access*; you do specify line numbers for *random access*. A common practice is to position the data pointer to an initial line in the data file by specifying the line number (random access) and then to print or read serially from there.

The following "stopwatch" program is an example of a serial print operation. The program prints to a data file the number of elapsed seconds every time you press a key, enabling you to record as many time splits as you'd like.

```
10 ASSIGN # 3 TO 'WATCH'
20 T0=TIME @ I=1
30 IF KEY$='' THEN GOTO 30
40 PRINT # 3 ; TIME-T0

50 DISP I @ I=I+1
60 GOTO 30
```

Creates and opens the data file.
Initializes T0 and I.
Causes the program to wait for a pressed key.
Prints (serially) the elapsed time since the start of the program.
Displays and increments the counter value.

Program Explanation. When you press [RUN], the program stores the beginning time in variable T0. From then on, pressing a key causes execution to jump to the PRINT # statement and print the elapsed time to the WATCH file. To examine the values in the data file, type `list 'watch'` [RTN].

The following example shows the use of a READ # statement for random access to the WATCH file.

```
•110 ASSIGN # 45 TO 'watch'

 120 INPUT 'what split? ';A
•130 READ # 45,A ; B

 140 DISP B;'seconds.'
 150 GOTO 120
```

Sets data pointer #45 to the first line of the WATCH file.

Uses the value of A to position the data pointer and to read the value from that line.

When a serial PRINT # statement is first executed, or when serial printing exceeds the line length of the current DATA statement, then either:

- The HP-75 advances the data pointer to the next higher-numbered DATA statement in the file, skipping lines between DATA statements, or
- The HP-75 creates a new DATA statement in the file by incrementing the last line number in the file by 1. This occurs when no other DATA statement exists after the location of the data pointer.

Serial READ # statements advance the data pointer from one DATA item to the next, skipping the lines

between DATA statements. Serial PRINT # and READ # statements enable you to print to and read from data files without referencing line numbers.

On the other hand, *random* PRINT # and READ # statements enable you to print to and read from specific lines but do not allow the data pointer to move past the end of line. First, a check occurs for the line specified in a random PRINT # or READ # statement; error 34—no data—will occur if the line is nonexistent (READ # only) or is a non-DATA line and the data pointer will be left at the beginning of that line. Assuming the line number is valid, if the items in the random PRINT # statement exceed the line length, then as many complete items as allowed by the line length will be printed, error 28—record overflow—will occur, and the data pointer will be left at the end of the specified line. If the random READ # statement tries to read more values than are stored in the specified DATA line, then error 34—no data—will occur, and the data pointer will be left at the end of the specified line. The restrictions on random PRINT # and READ # statements ensure that the data pointer won't move farther than expected.

Note that you can't cause a data pointer to move backwards across a DATA statement. Each PRINT # statement starts printing forward from the beginning of a DATA statement; each READ # statement begins reading forward from the location of the data pointer.

Special Forms of PRINT # and READ

There are two special forms of random PRINT # and READ # statements that include neither semicolons nor variable lists:

```
PRINT # file number , line number
```

```
READ # file number , line number
```

The random PRINT # statement moves the data pointer to the specified line, *deletes* it, but leaves the data pointer positioned at that line. The specified line must be either a DATA statement or a nonexistent line; otherwise, error 34—no data—will occur.

Example:

```
400 PRINT # 6, 200
```

Moves data pointer #6 to line 200, deletes that line from the file, and leaves the data pointer at the beginning of line 200.

The random READ # statement simply moves a data pointer to the beginning of the specified DATA statement.

Example:

```
410 READ # 6, 1000
```

Moves data pointer #6 to the beginning of the DATA statement in line 1000.

If either PRINT # or READ # statement specifies a non-DATA line or a READ # specifies a nonexistent line, then error 34—no data—will occur.

Moving the Data Pointer (RESTORE #)

The RESTORE # statement moves the data pointer either to a specified line of a file or to the lowest-numbered DATA statement in the file (if no line number is specified).

```
RESTORE # file number [, line number]
```

Both parameters may be numeric expressions. The *line number* must specify a currently existing DATA statement in the file; otherwise, error 34—no data—will occur.

The following statements are equivalent:

```
410 RESTORE # 6, 900
```

```
410 READ # 6, 900
```

} Both statements position the data pointer to the DATA statement in line 900.

Note that a program may reference itself with READ # and RESTORE # statements, just as with READ and RESTORE statements.

Statement Summary

Following is a summary of the ASSIGN #, PRINT #, READ #, and RESTORE # statements. The abbreviations *fn* and *ln* represent file number and line number, respectively, where $1 \leqslant fn \leqslant 9999$ and $0 \leqslant ln \leqslant 9999$.

Statement	Action	Error Conditions
ASSIGN # *fn* TO '*name*'	Sets data pointer to first line of specified file (line 0 of a newly created file).	If *name* is an invalid filename or *fn* is not in the range $1 \leqslant fn \leqslant 9999$.
ASSIGN # *fn* TO *	Ends the association between file and data pointer.	If *fn* is not in the range $1 \leqslant fn \leqslant 9999$.
PRINT # *fn* ; *items*	Finds next DATA line; if none, creates new DATA line at end of file. Prints items, advancing data pointer through the file.	If attempting to create a line greater than 9999.
PRINT # *fn*, *ln* ; *items*	Finds DATA line *ln*; if nonexistent, creates DATA line *ln*. Prints items and leaves data pointer at the end of line.	If *ln* is an existing, non-DATA line, or if items exceed one line length.
PRINT # *fn*, *ln*	Finds DATA line *ln*, deletes, and sets data pointer to beginning of *ln*.	If *ln* is an existing, non-DATA line.
READ # *fn* ; *items*	Reads items, beginning from location of data pointer and advancing data pointer through the file.	If attempting to read past the end-of-file.
READ # *fn*, *ln* ; *items*	Finds line *ln*. Reads items and leaves data pointer at the end-of-line.	If *ln* doesn't exist, if *ln* is an existing, non-DATA line, or if attempting to read past the end-of-line.
READ # *fn*, *ln*	Moves data pointer to beginning of *ln*.	If *ln* doesn't exist or is a non-DATA line.
RESTORE # *fn*	Moves data pointer to beginning of first DATA line in file.	If no DATA line exists in file.
RESTORE # *fn*, *ln*	Moves data pointer to beginning of *ln*.	If *ln* doesn't exist or is a non-DATA line.

All PRINT # statements, RESTORE, RESTORE # and random READ # statements initially move the data pointer to the beginning of a line.

Note that file numbers in PRINT #, READ #, and RESTORE # statements must refer to files that have been previously opened with ASSIGN #; otherwise, error 45—missing ASSIGN#—will occur.

BASIC files may reference themselves in READ # and RESTORE # statements, but error 51—PRINT# to runfile—will occur if a program attempts to print to itself.

Printing to or reading from private BASIC files, LEX files, and interchange files is not allowed. Error 65—access restricted—will occur if a PRINT # or READ # statement addresses a PB, L, or I file. The line numbers specified in PRINT #, READ #, and RESTORE # are *not* updated by RENUMBER commands.

Storing and Retrieving Arrays

Entire arrays can be stored and retrieved in single PRINT # and READ # statements. A comma between parentheses (e.g., B(,)) indicates a two-dimensional array.

Examples:

```
100 PRINT # 1: A()

200 READ # 1: B(,)
```

Prints all values of A(), a one-dimensional array, to the specified file.
Assigns values to the elements of array B(,) from the specified file.

A PRINT # statement causes an array to be printed as a series of data items, from lowest-numbered element to highest-numbered element, according to the lower bound and dimensioned size of the array. As much of an array as is allowed by line length will be printed in a single DATA statement. If an array contains any null, or unassigned, elements, warning 7—no value—will occur when the PRINT # statement is executed, and the data value for the null element will be set to zero (assuming a DEFAULT ON setting). You can interrupt the PRINT # operation by pressing [ATTN].

Two-dimensional arrays are printed and read element by element, with the last subscript varying the fastest; in other words, two-dimensional arrays are handled one row at a time. For example, if array B(,) has a lower bound of 1 and consists of the following values:

B(1,1) = 10	B(1,2) = 20	B(1,3) = 30
B(2,1) = 40	B(2,2) = 50	B(2,3) = 60
B(3,1) = 70	B(3,2) = 80	B(3,3) = 90

and if the array is printed to line 100 of a data file, then a listing of the data file will show:

```
100 DATA 10,20,30,40,50,60,70,80
,90
```

The array values are stored in "row major" order.

The following program fills array variable T(,) with four test scores and one average from three students, then prints array T(,) to the TESTS data file.

```
 10 !**scores and averages**
•20 OPTION BASE 1 @ DIM T(3,5)
 30 FOR I=1 TO 3
•40 T(I,5)=0
 50 FOR J=1 TO 4
•60 INPUT T(I,J)
•70 T(I,5)=T(I,5)+T(I,J)/4
 80 NEXT J
 90 NEXT I
•100 ASSIGN # 1 TO 'tests'
•110 PRINT # 1,2 ; T(,)
 120 DISP 'Array is stored.'
•130 CAT 'test' @ LIST 'test'
```

Dimensions T(,).

Initializes student average to zero.

Fills one element at a time.
Computes new average for the student.

Creates and opens data file TESTS.
Prints entire array to line 2 of TESTS.

Displays catalog entry and lists the data file.

Run the program and then list the data file—the 12 test scores appear in the same order they were entered from the keyboard; and test averages appear as every fifth DATA item.

You can read array information from a data file in much the same manner. For example:

```
•200 DIM S(3,5) ! OPTION BASE 1
 still in effect.
•210 READ # 1,2 ; S(,)
 220 DISP 'Averages: ';
 230 FOR L=1 TO 3
•240 DISP S(L,5);
 250 NEXT L
 260 DISP
•270 ASSIGN # 1 TO *
```

Declares another array of the same size and type as the original T(,).
Fills the entire array from the values in the data file.

Displays student averages.

Closes the file.

Note that an array variable in a READ # statement will be filled with as many data values as it has elements, in row major order.

You can also manipulate array data values element-by-element.

Examples:

```
•300 PRINT # 40 ; B(1,P1),B(2,P2)
 ,B(3,P3)
 ⋮
•500 FOR K=1 TO 10
 510 READ # 30 ; A(K)
 520 NEXT K
```

Prints three values of B(,) to file #40.

Assigns values to elements 1 through 10 of A()
from file #30.

For large arrays, *serial*—rather than random—PRINT # and READ # statements are recommended because serial printing and reading allows data values to be printed to and read from successive DATA statements, whereas random printing and reading may cause errors due to line length overflows.

Accessing Text Files

Programs can access *text* files as well as BASIC files through ASSIGN #, PRINT #, READ #, and RESTORE # statements. This capability enables programs to read, process, and write lines of text.

Example:

```
•10 ASSIGN # 1 TO 'Example',TEXT
•20 PRINT # 1 ; ' You can print q
 uoted strings, string variables,
  functions, or expressions.'
 30 B$=''
 40 FOR I=97 TO 122
•50 B$=B$&CHR$(I)
 60 NEXT I
 70 FOR J=1 TO 3
•80 PRINT # 1 ; B$

 90 NEXT J
```

Creates and opens a text file.

Prints the characters of a quoted string to the text file.

Collects the letters of the alphabet in B$, one by one.

Prints the characters of the string variable to the text file.

```
1 You can print quoted strings,
string variables, functions, or
expressions.
2abcdefghijklmnopqrstuvwxyz
3abcdefghijklmnopqrstuvwxyz
4abcdefghijklmnopqrstuvwxyz
```

The value of B$ is printed three times. Note that leading blanks are not inserted.

To create and open a text file, use the extended form of the ASSIGN # statement:

```
ASSIGN # file number TO 'filename' , TEXT
                                    BASIC
```

The ASSIGN # statement creates a file of the specified type if the file doesn't exist and sets the data pointer to line 0. It's unnecessary to specify TEXT or BASIC if the file already exists; the ASSIGN # statement sets a data pointer to the first line of the text or BASIC file.

After opening a text file, use the PRINT #, READ #, and RESTORE # statements to manipulate the contents of the file:

- PRINT # prints the values of string expressions to the file.
- READ # reads characters from the file into string variables and substrings.
- RESTORE # moves the data pointer to the first line of the file or to the beginning of the specified line.

Here are the main differences between accessing BASIC and text files:

Differences In:	Opened BASIC File	Opened Text File
Composition	Consists of zero or more numbered lines of DATA statements; may be mixed with other program statements.	Consists of zero or more numbered lines of text.
DATA Items	May be either characters or numbers, separated by comas.	Each line is interpreted as one item, that is, a contiguous string of characters.
Error Conditions	If line specified in a PRINT #, READ #, or RESTORE # statement is an existing, non-DATA line.	If line specified in a READ # or RESTORE # statement doesn't exist.

A PRINT # list addressed to a text file must consist of strings only, which may be quoted strings, string variables, substrings, string functions, or any concatenation. Error 65—access restricted—will occur if an attempt is made to print a numeric value to a text file. If two or more strings appear in the same PRINT # statement, then the characters of both strings will be joined in the same text file line. When using PRINT # to write text files from BASIC programs, it is a good practice to make sure that a *blank* is the first character of each text line. If digits directly follow a line-number—without a separating blank—then the operating system will interpret the entire numeric string as a line-number when a PLIST command is executed. Such lines will either not be listed or will be listed incorrectly and out of order.

The following example assumes that SCRATCH is an existing text file and that A$ and B$ are previously declared string variables:

```
100 ASSIGN # 3 TO 'scratch'
120 PRINT # 3,100 ; A$,B$


130 PRINT # 3; A$[1,8]&UPRC$(B$)



140 PRINT # 3, 99

150 RESTORE # 3
```

Opens the text file.
The random PRINT # statement finds line 100 or else creates line 100 (if nonexistent) and prints the values of both variables to line 100.
The serial PRINT # statement prints the value of the string expression to the next line of the file, creating a new line at the end of the file (101) if necessary.
Deletes line 99 but leaves the data pointer positioned to the beginning of that line.
Restores the data pointer to the first line of the file.

If a PRINT # item causes the line length of the text file to be exceeded, then:

- If in a *serial* print operation, the data pointer is advanced to the next line of the file—creating a new line at the end of the file if necessary—and that item and remaining items are printed to the new line.
- If in a *random* print operation, the data pointer is left at the end of the current line, error 28—record overflow—is reported, and that item and remaining items are not printed to the text file.

Each string variable in a READ # list is filled with the characters from one line of text. Multiple string variables in a READ # list cause succeeding lines of text to be read.

Examples:

```
300 DIM C$[96],D$[96]
310 READ # 1,100 ; C$
320 READ # 1 ; D$
330 READ # 1 ; C$,D$
```

Dimensions variables C$ and D$.
Reads all characters from line 100 into C$.
Reads all characters from the next line into D$.
Reads all characters from the next line into C$ and then from the next line into D$.

READ # statements leave the data pointer positioned at the end-of-line after the last character read from the text file.

Errors may be caused by improper reads from text files:

- A string variable in a READ # statement must be dimensioned to accomodate all the characters in the line read from the text file; otherwise, error 42—string too long—will occur.
- The READ # list can consist of string variables and substrings only; error 65—access restricted—will occur if an attempt is made to read text into a numeric variable.
- The optional line number in a READ # statement must refer to an existing line in the file; otherwise, error 34—no data—will occur.

- In a random read operation, the data pointer is not allowed past the end of the line. Consequently, multiple variables may not be specified in a random READ # statement.

The FINDIT Program

The following FINDIT program uses ASSIGN #, PRINT #, READ #, and RESTORE # statements to find string patterns in a text file named SOURCE.

When initialized, the program with remarks requires about 1200 bytes.

FINDIT **Program**

```
10 ! Findit
20 DELAY 2 @ WIDTH INF
30 DIM A$[96] ! A$=line; S$=sear
ch string.
40 ASSIGN # 10 TO 'source'
50 PRINT # 10,9999 ; 'end' ! add
 endmark.
60 RESTORE # 10 @ C=0 @ P=0 !
 sets pointer, line counter, and
line position.
70 INPUT 'Search string: ';S$
80 READ # 10 ; A$
90 IF A$='end' THEN 150 ! reache
d endmark.
100 C=C+1 ! increment counter.
110 P=POS(UPRC$(A$),UPRC$(S$)) !
 P is 0 if S$ not in line; else
P is position.
120 IF P<>0 THEN E=LEN(S$)+P-1 @
 GOTO 150 ! found it! E is end
position.
130 READ # 10 ; A$ ! read next
line.
140 GOTO 90 ! to examine line.
150 DISP C; ! line count.
160 IF P<>0 THEN A$[P,E]=FNU$(A$
[P,E]) @ DISP ': ';A$ ELSE DISP
'lines in text; not found.'
170 PRINT # 10,9999 ! delete end
mark.
180 ASSIGN # 10 TO * ! done
190 END
200 DEF FNU$(T$) ! To return T$
underlined.
210 FOR K=1 TO LEN(S$)
220 T$[K,K]=CHR$(NUM(T$[K])+128)
 ! replace character with _.
230 NEXT K
240 FNU$=T$
250 END DEF
```

SOURCE **Text File**

```
10 This file may be composed of
 any arbitrary arrangement of
20 characters entered from the
30 keyboard or printed to the
40 file from another BASIC
50 program.
```

The FINDIT program first prints an endmark in line 9999 of the SOURCE file so that the program will stop searching if it encounters the end of file. Given a string input of zero or more characters, the program searches through the SOURCE file for the first occurrence of the string, in lowercase or uppercase letters. If the pattern is found, the program displays the line number in which the pattern appears (relative to the first line of the SOURCE file) and displays the entire line with the pattern underlined. If the specified string is not found in the file, the program displays the number of lines in the file and the message not found.

When run, the program prompts for the search string:

```
Search string: 
                PRGM
```

```
Search string: basic
                PRGM
```

Assuming your SOURCE file is as above, search for the word 'BASIC'.

```
 4 :  file from another BASIC
```

The string is located in the fourth line of the text file.

Try running the program again and searching for a nonexistent pattern:

```
Search string: nonsense
                PRGM
```

```
 5 lines in text; not found.
```

Long Data Lines

A maximum of 94 characters may be typed in one display line; two positions are reserved for the prompt and cursor. However, PRINT # statements enable you to print data lines to BASIC and text files more than twice as long.

The upper limit for a data line printed to a BASIC file is 253 bytes. Each item in the DATA statement will use two or more bytes of the available 253 bytes:

- Each integer requires 4 bytes.
- Each non-integer requires 9 bytes.
- Each string requires 2 bytes plus 1 byte for each character in the string.

Note that none of the 253 bytes are used for line number, DATA keyword, spaces, quotation marks, and commas.

The upper limit for a data line printed to a *text* file is 255 characters including the line number.

When a long data line is afterwards listed or fetched, warning 67—line too long—will be reported and only the first 94 characters of the line will be displayed. Although the end of a long line may not be examined or edited directly, all of the values in that line may be accessed with READ and READ # statements.

Error conditions:

- If more than 253 bytes are printed to a BASIC file in a random PRINT # statement, then error 28—record overflow—will occur and only as many items as will fit in the line will be printed.
- If more than 255 characters are printed to a text file in a random PRINT # statement, then error 28—record overflow—will occur and only as many string expressions as will fit in the line will be printed.

In contrast, a serial PRINT # statement causes the data pointer to advance to the next available line and print the remainder of the data items in that line.

Section 15

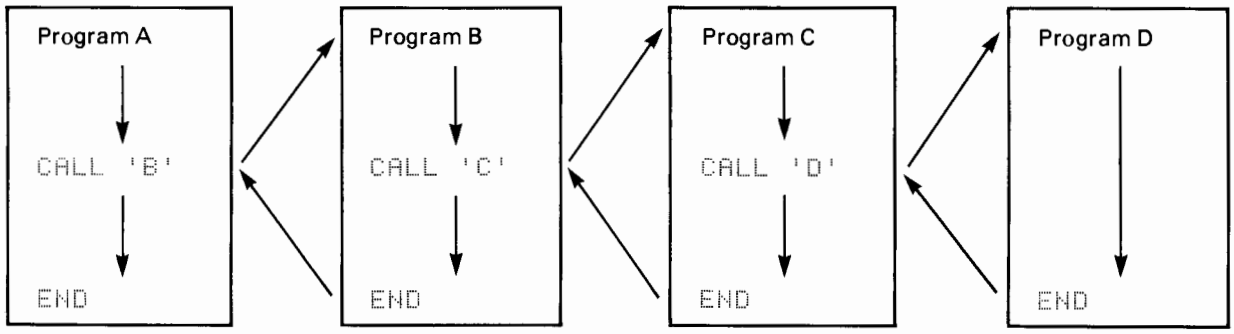
Program Calls

Contents

Introduction 230
Calling Programs (CALL, END) 230
Comparing RUN and CALL 231
The FIRST and SECOND Programs 231
Passing Values Between Programs (ASSIGN #, PRINT #, READ #) 232
Global and Local Declarations 233
Recursive Calls 234

Introduction

Any program may *call*, or execute, another. That program may in turn call another, which may call a third, and so on.



When a called program finishes executing, it returns control back to the program that called it, after the CALL statement. Thus, main programs can consist of a series of program calls to *procedures*, often referred to as *subprograms*. For example, frequently used routines may be stored in memory as "library procedures," accessible to all programs. Values can be shared between programs by means of data files; parameter passing by *address* and by *value* are possible.

Calling Programs (CALL, END)

Program calls cause an unconditional branch of program execution. Place a CALL statement in the calling program at the point where you want the branch to occur.

```
CALL 'filename'
```

The *filename* may be specified by any string expression that names an existing program file in memory. Any file that can be run can be called.

When the called program encounters an END statement or the end-of-file marker, it returns execution to the calling program at the first statement following the CALL statement. The number of program calls without corresponding returns is limited only by available memory.

Comparing RUN and CALL

RUN and CALL instructions share several characteristics:

- Neither RUN nor CALL affects the values of calculator variables.
- Both RUN and CALL initialize the values of program variables.
- Both run and called programs are deallocated and their variable values are lost when execution *ends*; both types of programs remain initialized when execution is *interrupted* with [ATTN] or STOP.
- Both RUN and CALL can be executed from the keyboard.
- Any program can RUN or CALL any other program.
- The file pointer is not moved from the current EDIT file when RUN and CALL are executed.

The differences are that:

- RUN may specify a beginning line number; CALL always begins execution at the lowest-numbered line of a program.
- Execution does not return to a program that runs another program.
- The variable values of a calling program are preserved while the called program is executing. RUN deallocates all program variables except those of the currently executing program.

The FIRST and SECOND Programs

To examine a simple calling operation, write a program named FIRST and another program named SECOND that will be called by FIRST. Start with SECOND: Type `edit 'second'` [RTN] and enter the four-line program.

```
10 DISP 'This is SECOND.'
20 A=2
30 B$='second'
40 STOP
```

} To display a message and assign two variable values.

To interrupt execution.

Write the FIRST program now: Type `edit 'first'` [RTN] and enter the program.

```
10 DISP 'This is FIRST.'
20 A=1
30 B$='first'
•40 CALL 'second'

50 DISP 'Back to FIRST.'
60 STOP
```

Execution will transfer to SECOND and return to the following line (50).

After entering both programs, press [RUN]:

```
This is FIRST.
This is SECOND.
```

Execution stops at line 40 of SECOND. However, the file pointer stays fixed at FIRST—that is, in the last file you edited—at the line where it was left when [RTN] was pressed. Verify the location of the file pointer by

typing `cat` [RTN] for the filename and `FETCH` [RTN] for the current line. Now press [ATTN], then check the values of variables `A` and `B$`:

```
>a,b$
```

```
 2                    second
```

The values as assigned by the called program.

The example is to show that you may check and change only the variable values of the interrupted program (the called program `SECOND`), even though the file pointer may be positioned in another program (the calling program `FIRST`). However, the values of the calling program variables are saved in memory until execution returns to the calling program.

Note also that you may check and change the values of current *calculator* variables during a program interruption, assuming they have differing names from the interrupted-program variables.

Type `cont` [RTN] to resume program execution:

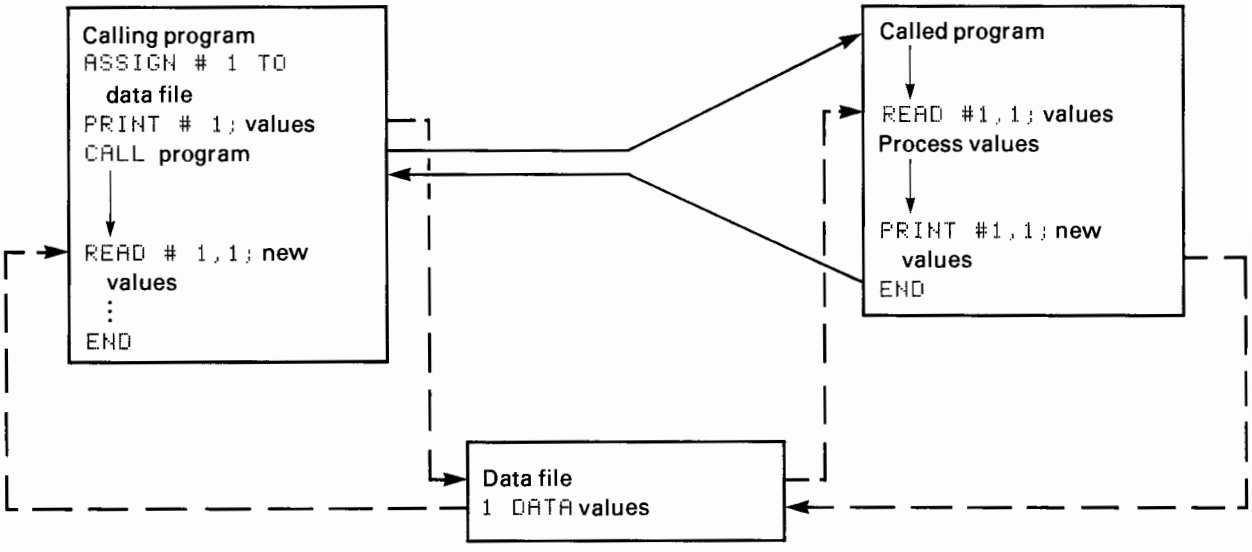
```
Back to FIRST.
```

Execution is now interrupted at the `STOP` statement in line 60 of `FIRST`. When you check the values of `A` and `B$`, you'll see that they've resumed their original values, `1` and `first`, respectively. When the `SECOND` program was continued, it immediately encountered the end-of-file marker and returned execution to line 50 of `FIRST`. In the process, the `SECOND` program was deallocated and its variable values were lost.

If an error occurs after a program call, then an error message is displayed and execution stops at the line in the called program that caused the error; however, the file pointer stays fixed in the last file you edited before running the program.

Passing Values Between Programs (`ASSIGN #, PRINT #, READ #`)

Programs may share any number of values by means of data files.



The diagram shows that the calling program creates a data file, prints to it the values to be passed, and then calls the second program. The called program reads the values from the data file, processes them, and prints the new values to the data file. When execution returns to the calling program, the program reads the new values from the data file.

Any filename, file number, or file line may be specified for the data file, as long as the calling program and called program access the same DATA items. The following two programs compute the polar coordinates of a point, given its rectangular coordinates. Type edit 'main' [RTN] and enter the calling program:

```
 10 ! --MAIN--
 20 OPTION ANGLE RADIANS
 30 INPUT 'Rectangular coordinate
 s: ';X,Y
•40 ASSIGN # 1 TO 'data'

•50 PRINT # 1 ; X,Y
 60 CALL 'polar'
•70 READ # 1,2 ; R,T
 80 DISP 'The polar coordinates o
 f';X;Y;'are';R;T
```

DATA provides temporary storage for coordinate values.

Prints the values of X and Y to the data file.

Reads the new values from the data file.

Then type edit 'polar' [RTN] and enter the POLAR procedure:

```
 10 ! --POLAR--
•20 READ # 1,1 ; X9,Y9
 30 R9=SQR(X9^2+Y9^2)
 40 T9=ANGLE(X9,Y9)
•50 PRINT # 1,2 ; R9,T9
 60 END
```

Accesses main program variable values.

Prints new values to the data file.

Now run the MAIN program:

```
Rectangular coordinates: ■
                         PRGM
```

Type 3,4 and press [RTN].

```
The polar coordinates of 3 4
are 5   .927295218002
```

Note that the POLAR procedure reads from line 1 of the data file and prints to line 2. It could instead read from and print to the same line, assuming that the main program also addresses that line.

Note also that the values of the main program parameters X and Y aren't affected by what the called program does to the values. This is referred to as passing by *value*. Passing by *address*—when the calling program parameter values *are* changed by the called program—is also possible; the calling program should assign the processed values to the original parameters as it reads the new values from the data file.

You may choose to have the calling program purge a data file when the values transfer is complete, or you may choose to list the data file to check the values that have been printed to it. Commonly, you'll need to reference only one or two lines of a data file when passing parameter values.

Global and Local Declarations

Any program may set a machine condition, such as the DELAY rate or the trigonometric mode. A *global* declaration, e.g., OPTION ANGLE RADIANS, remains in effect until another global declaration, e.g., OPTION ANGLE DEGREES, is executed.

All commands that set machine conditions (page 163) as well as certain BASIC statements act globally. For example, the file numbers declared in ASSIGN # statements are global—any opened data file may be accessed from any program as well as from the keyboard. Once an ASSIGN # statement is executed, the specified file number stays associated with the specified file until another ASSIGN # statement is executed that redeclares the file number.

Other statements, called *local* declarations, affect only the operation of the currently executing program. The DIM, IMAGE, LET, and OPTION BASE statements are examples of local declarations. Branching statements, subroutines, and user-defined functions are all local to the currently executing program.

Program timers also operate locally. When execution is transferred by a CALL statement, an active timer will continue its count but will not cause any interrupts. When execution returns to the calling program, the timer will interrrupt when its next interval comes due, as if nothing had intervened. For example, if a timer is set for a 10-second interval and 5 seconds before it comes due another program is called—taking 7 seconds to run—then the timer will be ignored during the call and the next interrupt will occur 8 seconds after execution returns to the calling program.

Although timer interrupts are local, timer *numbers* are global. For example, there can be only one timer #33. If a program declares timer #33 and then calls another program, and if the called program itself declares timer #33, then timer #33 becomes the "property" of the called progam. The calling program must subsequently redeclare the timer to regain control. This means that any called program may turn off a timer set by any other program.

The sequence of random numbers returned by the RND function (page 83) continues unbroken during program calls. For example, if program A displays three random numbers, x_1, x_2, and x_3, and calls program B, then the next random number available to B will be the fourth random number in the sequence, x_4 (unless, of course, B contains a RANDOMIZE statement, which will start a new sequence). If B is *run* rather than called, the sequence of random numbers will begin at a starting number local to B.

Refer to section 17 for a description of ON ERROR, OFF ERROR, TRACE FLOW, TRACE VARS, and TRACE OFF declarations.

Recursive Calls

The CALL statement allows *recursion*—that is, procedures may call *themselves*.

Example: The factorial function may be defined recursively as follows:

For all nonnegative integers, x,

$$\text{FAC}(x) = \begin{cases} 1 & \text{if } x = 0. \\ x \times \text{Fac}(x-1) & \text{otherwise.} \end{cases}$$

This is a recursive definition because the function in the second instance is defined in terms of itself.

The following two programs show how recursive calls can be used to compute factorials. Type `edit 'shell'` RTN and enter the main program:

```
 10 !--shell--
•20 ASSIGN # 1 TO 'temp'
 30 INPUT 'Factorial of? ';M
•40 M=ABS(IP(M))
•50 PRINT # 1,1 ; M
```

Opens a data file for passing values.

Converts M to a positive integer value.
Prints the value of M to the data file.

```
 60 CALL 'fac'
•70 READ # 1,1 ; N

 80 DISP M;'factorial is';N
 90 GOTO 30
```

Receives the final value from the final execution of FAC.

Then type `edit 'fac'` RTN to write the recursive procedure:

```
 10 !--fac--
 20 READ # 1,1 ; X
 30 IF X=0 THEN PRINT # 1,1 ; 1 @
  END
 40 PRINT # 1,1 ; X-1

 50 CALL 'fac'
 60 READ # 1,1 ; Y


 70 X=X*Y

 80 PRINT # 1,1 ; X

 90 END
```

X inherits its value from the calling program.

Tests the current value of parameter X for the exit condition.

If the test fails, X is decremented and the new value printed to the file.

The recursive CALL.

Execution from the last call returns here. Y is assigned the value returned from the previous procedure.

Computes the factorial for the current level of recursion.

Returns the current value of the factorial to the data file.

Returns execution to the calling program.

After you've entered both programs, run SHELL and find the factorial of 6:

```
Factorial of? 6
                        PRGM
```

Type 6 and RTN.

```
6 factorial is 720
```

The answer, after 6 calls to FAC.

You may examine the way succeeding program calls affect the size of available memory. Insert a line 15 in the FAC procedure:

```
15 DISP MEM
```

To display the amount of available memory right after the procedure is called.

When you run SHELL now, you'll notice a decrease in available memory after each call; memory is required for saving program variable values, return addresses, and so on. The number of layers of program calls, including recursive calls, is limited only by available memory.

Other forms of recursion are allowed:

Mutual Recursion

A calls B
B calls A

Indirect Recursion

A calls B
B calls C
C calls A

Here's a simple example of mutual recursion:

The FRED program.

```
10 DISP 'This is Fred-call Mary'
20 CALL 'Mary'
```

The MARY program.

```
10 DISP 'This is Mary-call Fred'
20 CALL 'Fred'
```

These two programs will cause an endless build-up of program calls until execution is interrupted (by pressing [ATTN]) or until memory runs out (causing error 16—`not enough memory`). Note that when you edit either program, the system will deallocate *both* programs and reclaim the memory formerly required for keeping track of variable values and program returns. (Running a different program or typing `clear vars` [RTN] deallocates the programs, too.)

The CAT$ function discussed in section 13 is useful for tracing program flow when control is passed between programs. Remember that CAT$ of a negative argument returns the catalog entry of the currently executing program. For example, `DISP CAT$(-1)` at the beginning of a procedure will display the catalog entry of the procedure when it begins executing.

Section 16

Display and Printer Formatting

Contents

Introduction 238
Using IMAGE 238
Delimiters (,, /) 239
Blank Spaces (X, %) 240
String Specifications (' ', " ", a, A) 240
Numeric Specification 241
  Digit Symbols (d, D, z, Z, *) 241
  Radix Symbols (., r, R) 243
  Sign Symbols (s, S, m, M) 243
  Digit Separator Symbols (c, C, p, P) 244
  Exponent Symbol (e, E) 245
Compact Field Specifier (k, K) 245
Replication (( )) 246
Reusing the Image Format String 246
Numeric Field Overflow 247
Formatting in DISP USING and PRINT USING Statements 247
Image Format Summary 249

Introduction

16 You can control display and printer output using:

- Commas, semicolons, and the TAB function in DISP and PRINT statements.
- WIDTH and PWIDTH commands.
- Control characters and escape codes.
- The ENDLINE statement (for PRINT and PLIST only).

Three statements, DISP USING, PRINT USING, and IMAGE, provide even more formatting control over output to DISPLAY IS and PRINTER IS devices.

Using IMAGE

The IMAGE statement specifies the format by which numbers and strings in the DISP USING and PRINT USING statements will be printed or displayed.

```
DISP USING line number [;disp using list]
```

```
PRINT USING line number [;print using list]
```

```
IMAGE format string
```

The *line number* in a DISP USING or PRINT USING statement must be an unsigned integer, 0 through 9999, and must refer to a valid IMAGE statement, which may be placed anywhere in the program. The *disp using* and *print using* lists may be comprised of any numeric or string expressions.

The *format string* in an IMAGE statement is a series of characters that specifies the desired format or formats for output. The HP-75 will accept *any* combination of characters after an IMAGE keyword with or without quote marks. The HP-75 checks the IMAGE statement for correct format specifications when the IMAGE statement is executed during a program and not when you're typing the statement itself. An IMAGE statement shouldn't appear in a multistatement line; if it follows another statement, an IMAGE statement will never be executed; if another statement follows the IMAGE keyword, that statement won't be recognized as a statement. An IMAGE statement executed from the keyboard will be ignored.

The items in the *disp using* or *print using* list are separated by commas or semicolons. However, the commas and semicolons don't affect the format as they do in DISP and PRINT statements; they merely separate the items in the list. The output is totally controlled by the *format string* of the IMAGE statement.

The IMAGE format string is composed of *field specifiers*, separated by delimiters. Each specifier is itself composed of symbols that determine the format of a single item in the *disp using* or *print using* list. The symbols in the IMAGE statement specify the number of digits, the placement of a comma, decimal point, or blanks—virtually anything having to do with numeric and string output and carriage control.

Example:

```
300 IMAGE dddcddd.dd, 10a
```

This IMAGE format string is composed of two field specifiers: The first specifies digits, comma, and decimal point; the second specifies 10 characters of text.

Each item in the *disp using* or *print using* list must correspond to an appropriate numeric or string field specifier.

If you intend to print to an external printer, it must be addressed with a PRINTER IS statement. If no PRINTER IS device is specified, all printer output will be directed to the HP-75 display and other DISPLAY IS devices.

Delimiters (, , /)

Two delimiters are used to separate field specifiers:

- A comma (,) is used to separate two specifiers in an IMAGE statement.
- A slash (/) may be used to separate two string specifiers, but its main function is to perform a carriage-return and line-feed (CR/LF). The slash can be separated from other specifiers by a comma.

Example:

```
 10 DISP USING 20
•20 IMAGE 'cost',3/,'discount'
```

3/ is equivalent to ///. The commas are optional.

Executed as:

```
cost


discount
```

Displays `cost` and performs 1st CR/LF.
Blank display; performs 2nd CR/LF.
Blank display; performs 3rd CR/LF.
Displays `discount`.

The symbols `3/` indicate that three carriage returns and line feeds are to be performed between displaying the quoted strings. Thus, two blank display lines are output. (If the output were sent to a printer, two blank lines would be printed between `cost` and `discount`.)

The following `IMAGE` statement will output three blank lines before printing `cost`:

```
20 IMAGE 3/ 'cost'
```

If *n*/ is at the beginning of an image format string, *n* blank lines are output. If *n*/ follows any field specifier in an image format string, *n* − 1 blank lines are output.

Note that `DISP USING` and `PRINT USING` statements generate a carriage-return/line-feed at the end of the *disp using* and *print using* list. It's possible to suppress the CR/LF with a semicolon in the `DISP USING` and `PRINT USING` statements as it is in `DISP` and `PRINT` statements.

Blank Spaces (`x`, `X`)

`x` (or `X`) Specifies a blank space.

A number preceding `x` specifies the number of blanks; for instance, `4x` means four blanks. (`xxxx` also specifies four blanks.)

String Specifications (`' '`, `" "`, `a`, `A`)

Text can be specified in two ways:

`' '` (or `" "`) Text enclosed within quotation marks is displayed or printed exactly as it is quoted, without the outer pair of quotes. You may specify quoted strings in either the *disp using* or *print using* list or in the `IMAGE` statement.

Example:

```
30 IMAGE '^^',8x,'Results',8x,'^
^'
40 DISP USING 30
```

`8x` causes eight blanks to be output.

The location of an `IMAGE` statement with respect to the `DISP USING` statement is arbitrary.

Executed as:

```
^^        Results        ^^
```

a (or A) Specifies a single character. A number preceding a specifies the number of characters that will be displayed or printed. When using the a string specifier, all text is left-justified.

The above example could also have been written:

```
 50 A$='Results'
•60 IMAGE '^^',8x,7a,8x,'^^'

 70 DISP USING 60 ; A$
```

7a specifies a field comprised of seven characters of text.

Or like this:

```
 80 A$='Results'
•90 IMAGE aa,8x,7a,8x,aa
 100 DISP USING 90 ; '^^',A$,'^^'
```

aa can also be represented as 2a.

If the string item in the *disp using* or *print using* list is longer than the number of characters specified, the string is truncated.

Example:

```
 110 DISP USING 120 ; 'residence'
•120 IMAGE 6a
```

Allows only six characters to be output.

Executed as:

```
reside
```

If the item is shorter, the rest of the field (to the right) is filled with blanks. The following IMAGE statement will output a field of 80 characters.

```
120 IMAGE 80a
```

The string residence will be followed by 71 (80 − 9) blanks.

Numeric Specification

A variety of symbols can be used to specify numbers: digit symbols, sign symbols, radix symbols, separator symbols, and an exponent symbol.

Digit Symbols (d, D, z, Z, *)

d (or D) Specifies a digit position. A number preceding d specifies the number of digit positions. If the number of d's to the left of the decimal point or radix specify a field larger than the numeric item, then the item is right-justified in the field and leading zeros are replaced with spaces. If the number of d's to the right of the decimal point or radix specify a field larger than the numeric item, then the item is left-justified in the field with trailing zeros. If the fractional part of the numeric item is larger than the number of d's to the right of the decimal point or radix, then the item is rounded to fit the specified field.

Example:

```
130 DISP USING 140 ; 250,25.5
140 IMAGE 5d,2x,dd.dd
```

Executed as:

```
  250  25.50
```

Note that D and d are the only digit symbols that can be used to specify digits to the *right* of a decimal point or radix. For example, to specify a decimal point and two digits, the symbols .dd, .2d, .DD, etc. must be used.

z (or Z) Specifies a digit position—leading zeros are replaced with zeros as a fill character. You cannot use a z to the right of a radix symbol. Again, a number preceding z specifies the number of digit positions.

Example:

```
150 DISP USING 160 ; 256,321
160 IMAGE 5z,2x,zzzzz
```

Executed as:

```
00256  00321
```

* An asterisk also specifies a digit position, but leading zeros are replaced with asterisks as a fill character. You cannot use an * to the right of a radix symbol. A number preceding * specifies the number of asterisks.

Example:

```
170 IMAGE 5*,2x,5z,2x,5d
180 DISP USING 170 ; 99,77,55
```

Executed as:

```
***99  00077     55
```

As you can see, any digit symbol, *, z, or d, can be used to specify the integer portion of any number. However, you can't arbitrarily mix the symbols in an IMAGE format. For instance, if d is used to specify a digit position of a number, all of the number must be specified with d's, except that the digit symbol specifying the one's place can be a z regardless of the other symbols.

Example:

```
 190 DISP USING 200 ; 357,972
•200 IMAGE ddzz,2x,d*zzz*
```

An invalid statement.

The IMAGE statement contains two invalid specifiers (ddzz and d*zzz*) and will cause error 52—invalid IMAGE—to occur when the DISP USING statement is executed. However, the following image strings are valid:

```
200 IMAGE dddz,2x,*****z
```

Executed as:

```
 357  ***972
```

Whenever an IMAGE field specifier contains an unrecognized symbol, an error 52—invalid IMAGE—will occur when the corresponding DISP USING or PRINT USING statement is executed and execution will stop at the line containing the DISP USING or PRINT USING statement. Refer also to Numeric Field Overflow, page 247.

Radix Symbols (. , r, R)

A radix indicator is the symbol that separates the integer part of a number from the fractional part. In the United States, this is customarily the decimal point, as in 34.7. In Europe, this is frequently the comma as in 34,7. *One radix symbol at most can appear in a numeric specifier.* Only the symbol d can be used to specify a digit to the right of the radix indicator.

.	Specifies a decimal point in that position.
r (or R)	Specifies a comma radix indicator in that position.

Examples:

```
210 DISP USING 220 ; 473.1,25.39
2,76.5
220 IMAGE ddd.dd,2x,***.ddd,2x,z
zzrdd
230 IMAGE ddd.ddd,4x,3z.3d,4x,.d
d
240 DISP USING 230 ; .756,99.99,
.879
```

Executed as:

```
473.10  *25.392  076,50
   .756    099.990     .88
```

Note that .879 has been rounded to .88 since the image statement specified only two digits to the right of the radix.

Sign Symbols (s, S, m, M)

Two sign symbols control the output of the sign characters + and −. Only one sign symbol at most can appear in a numeric specifier. When no sign symbol is specified, any minus sign occupies a digit position.

s (or S)	Specifies output of a sign: + if the number is positive, − if the number is negative.
m (or M)	Specifies output of a sign: − if the number is negative, a blank if it is positive.

Example:

```
250 DISP USING 260 ; -47.2,-.51,
33.5,38.12
260 IMAGE mdd.dd,2x,szz.dd,2x,sz
z.dd,2x,mzz.dd
```

Executed as:

```
-47.20  -00.51  +33.50   38.12
```

The sign "floats" with the number; for example:

```
270 DISP USING 280 ; -5,6,-.07
280 IMAGE sddd.d,s3d.d,m2d.dd
```

Executed as:

```
 -5.0  +6.0  -.07
```

In the examples above, the sign appears immediately to the left of the number. If you use a z or * symbol in your format, the sign will appear to the left of any leading zeros or asterisks.

Digit Separator Symbols (c, C, p, P)

Digit separators are used to break large numbers into groups of digits (generally three digits per group) for greater readability. In the United States the comma is customarily used; in Europe, the period is commonly used.

c (or C)	Specifies a comma as a separator in the specified position.
p (or P)	Specifies a period as a separator in the specified position.

The digit separator symbol is output if a digit in that item has already been output; the separator should appear between two digits. When leading zeros are generated by the z symbol, they are considered digits and will contain separators. An IMAGE format string consisting of leading asterisks may contain separators. But if numbers are not output on both sides of the separator, the separator will be replaced with an asterisk.

Examples:

```
290 DISP USING 300 ; 25613.92,27
.96,71.5
300 IMAGE 3dc3d.dd,2x,zc3z.dd,2x
,3dc3d.dd
310 DISP USING 320 ; 9999.99
320 IMAGE dpdddrdddd
```

Executed as:

```
 25,613.92  0,027.96       71.50
9,999,9900
```

Exponent Symbol (e, E)

e (or E) Specifies that the numeric field that contains e or E is to be output in exponential notation. The symbol causes the HP-75 to express the sign of the exponent (+ or −) and a three-digit exponent. At least one digit symbol must precede the e symbol.

Examples:

```
330 DISP USING 340 ; 157.24
340 IMAGE d.ddde
350 DISP USING 360 ; 5.762
360 IMAGE ddd.dde
```

Executed as:

```
1.572E+002
576.20E-002
```

The HP-75 supplies as many trailing zeros as necessary for the number of digits specified in the mantissa and adjusts the exponent accordingly.

Example:

```
350 DISP USING 360 ; 3
360 IMAGE 25de
```

Executed as:

```
300000000000000000000000E-024
```

Note that no decimal point (.), radix (r), or sign (s,m) may be specified after the exponent symbol e; exponents will always be output with a leading plus or minus sign, three digits (with leading zeros as placeholders), and no decimal point.

Compact Field Specifiers (k, K)

A single symbol, k, is used to specify an entire field for either a number or string of characters. Items are output in compacted form, with no leading or trailing blanks.

Example:

```
370 DISP USING 380 ; 'ABC',415,'
DEF',.01
•380 IMAGE k,k,k,k
```

Specifies four compacted output fields (may *not* be abbreviated 4k).

Executed as:

```
ABC415DEF.01
```

Replication ( )

Many of the symbols used to make up image specifiers can be repeated to specify multiple symbols by placing an integer in the range 1 through 9999 in front of the symbol. You have already seen some examples; the following IMAGE statements, for instance, all specify the same image:

```
390 IMAGE ddd.dd
400 IMAGE d2d.2d
410 IMAGE 3d.dd
420 IMAGE 3d.2d
```

These symbols can be replicated: x, d, z, *, a, and /.

In addition to symbol replication, an entire specifier or group of specifiers can be replicated by enclosing it in parentheses and placing an integer in the range 1 through 9999 before the parentheses.

Examples:

```
430 IMAGE dd.d,6(ddd.dd)
440 IMAGE 4z.d,3(6x,7*.d,2(2x,d)
)
```

Specifying 3(dd) is the same as specifying dd,dd,dd, so statement 440 above is eqivalent to:

```
440 IMAGE 4z.d,6x,7*.d,2x,d,2x,d
,6x,7*.d,2x,d,2x,d,6x,7*.d,2x,d,
2x,d
```

Items in inner parentheses are replicated each time the outer expression is replicated.

In this manner, k can be repeated:

```
450 IMAGE 4(k)
```

Equivalent to specifying k,k,k,k.

You can use nested parentheses for additional levels of replication.

Reusing the Image Format String

A format string is reused from the beginning if it is exhausted before the items in the *disp using* or *print using* list have all been expressed.

Example:

```
460 DISP USING 470 ; 25.71,99.9,
14.23
•470 IMAGE ddd.dd
```

This IMAGE statement will be used three times to express the three items in the DISP USING list.

Executed as:

```
 25.71 99.90 14.23
```

Numeric Field Overflow

If a numeric item requires more digits to the *left* of the decimal point or radix than the field specifier provides, an overflow condition occurs.

Example:

```
480 DISP USING 490 ; 336.71,-14.
3
490 IMAGE 2x, dd.dd
```

Both numbers 336.71 and −14.3, with the same image specification of `dd.dd`, create an overflow condition because both numbers require three digit positions to the left of the decimal point. (Remember that a minus sign not explicitly specified with `s` or `m` requires a digit position.) With `DEFAULT ON`, warning 2—`num too large`—occurs, only the digit separators specified by `p` or `c` and signs specified by `m` or `s` or implied by `*` or `z` specifiers are output, and program execution continues. With `DEFAULT OFF`, the HP-75 reports an error and halts execution. To correct the image specification, specify more digit positions:

```
490 IMAGE 2x,ddd.dd
```

Allows for three numeric positions to the left of the decimal.

Executed as:

```
  336.71  -14.30
```

If a character string has more characters than is specified by `a`'s, the string is truncated after the specified number of characters, and no error occurs.

Formatting in `DISP USING` and `PRINT USING` Statements

There's another form that a `DISP USING` or `PRINT USING` statement may have, which enables you to specify the image string and the *disp using* or *print using* list in the same statement:

```
DISP USING 'image format string' [;disp using list]
PRINT USING 'image format string' [;print using list]
```

Instead of specifying the `IMAGE` line number, you can include the image format string, enclosed within quotation marks, in the `DISP USING` and `PRINT USING` statements before you specify the *disp/print using* list. The image format string may be a string enclosed within quote marks, a string variable, or any string expression that specifies the format.

Examples:

```
500 DISP USING '4d.dd,2x,**z.ddd
' ; 1473,25.39
510 DISP USING '3d.2d' ; 310.12,
56,42.5
•520 F$='3dc3d.2d'
530 DISP USING F$ ; 25613.92,279
6
```

Remember to dimension the string if it is longer than 32 characters.

Executed as:

```
1473.00  *25.390
310.12 56.00 42.50
 25,613.92  2,796.00
```

The image specifier stored in F$ is applied twice.

Note that an outer pair of quotation marks must be used to delimit the *entire* format string in a DISP USING or PRINT USING statement. For instance, the following DISP USING statement includes an incomplete string and is not allowed.

```
>540 disp using 'Name',2x,10a,'A
ge',3d;'Charles',43
```

Pressing [RTN] causes error 84—extra characters—to occur.

The error occurs after the second quotation mark because the HP-75 doesn't allow multiple format strings in single DISP USING statements. The same formats could be specified in a number of ways:

```
540 DISP USING "'Name',2x,10a,'A
ge',3d" ; 'Charles',43
```

Enclosing the entire format string with double quotes.

```
550 DISP USING '4a,2x,10a,3a,3d'
 ; 'Name','Charles','Age',43
```

Using character specifiers a for the three strings.

```
560 DISP USING 570 ; 'Charles',4
3
570 IMAGE 'Name',2x,10a,'Age',3d
```

Using a DISP USING-IMAGE combination.

All executed as:

```
Name  Charles   Age 43
```

If a DISP USING or PRINT USING image format string contains an inappropriate or unrecognized symbol, then error 52—invalid IMAGE—will occur when the statement is executed.

DISP USING and PRINT USING statements that contain image format strings may be executed from the keyboard; when referencing IMAGE statements, DISP USING and PRINT USING statements are programmable only.

Image Format Summary

Following is a summary of image symbols and their uses. Image specifiers may be delimited with commas or slashes.

Image Symbol	Type of Output	Comments	May be repli-cated
X, X	Blank	Specifies a blank between items.	Yes
" ", " "	Literal text	Used to delimit string items in format strings or entire specifiers in DISP USING and PRINT USING statements.	no
A, A	Character	Specifies a character position; text is left-justified.	yes
D, D	Digit	Specifies a digit position to left or right of radix symbol; leading blanks and trailing zeros.	yes
Z, Z	Digit	Specifies digit position to left of radix; leading zeros.	yes
*	Digit	Specifies digit position to left of radix; leading asterisks.	yes
S, S	Sign	Specifies sign, + or −.	no
M, M	Sign	Specifies sign, blank or −.	no
E, E	Exponential notation	Outputs numeric with exponent E, sign, and three digits.	no
.	Decimal point radix	Outputs a decimal point as radix in that position.	no
R, R	Comma radix	Outputs a comma as radix in that position.	no
C, C	Comma separator	Outputs a comma as a digit separator in the specified position.	no
P, P	Period separator	Outputs a period as a digit separator in the specified position.	no
( )	Field	Allows enclosed image specifiers to be replicated.	yes
K, K	Compacted item	Causes both strings and numerics to be output with no leading or trailing blanks.	no
/	CR/LF	Causes a carriage-return/line-feed; may also delimit items.	yes

Because the IMAGE statement is a powerful and complex tool for controlling display formats, you must use care when designing image specifications. Inappropriate specifications are not always flagged as errors and the output from incorrect or inappropriate formats may not produce the result you expected. As you have seen in this section, the IMAGE statement allows you to completely control the format of your output. Be aware, when designing image statements, of the possible values that your output data might take and prepare image specifications that will accommodate all valid outputs. The main factors that must be taken into account with formatted output are the length of the display or print line, as determined both by the DISPLAY IS or PRINTER IS device; the current WIDTH or PWIDTH setting; and the number of significant digits or characters in the output data. In general, formatting should be designed so that a line of characters does not exceed the number of characters per line.

DISP USING and PRINT USING statements will ignore format items that appear at the end of the IMAGE string if the USING list is terminated with a semicolon.

Example:

```
100 IMAGE dd.dd, ' temp.'
200 DISP USING 100 ; 32;
```

The image statement contains a numeric specifier and a literal string.

Executed as:

```
32.00
```

The literal string ' temp.' will not be displayed.

Section 17

Debugging Operations

Contents

Introduction 252
Tracing Program Execution 252
Tracing Branches (TRACE FLOW) 252
Tracing Variables (TRACE VARS) 253
Cancelling Trace Operations (TRACE OFF) 253
Using TRACE Commands 254
Single-Step Execution ([SHIFT] [RUN]) 256
Checking a Halted Program 257
Processing Run-Time Errors (ON ERROR, OFF ERROR) 258
Recovery Routines (ON ERROR GOTO, ON ERROR GOSUB) 259
Other Considerations 259
Error Numbers and Lines (ERRN, ERRL) 260

Introduction

Errors are a common occurrence in computer programming. The four types of errors are syntax (language-related) errors, initialization errors (such as missing line numbers for GOTO statements and duplicate variable declarations), run-time errors, and logical errors. The HP-75 checks for correct syntax as you enter statements, so syntax errors are immediately apparent. Initialization errors are flagged during program initialization, so they too are quickly remedied. Run-time errors and logical errors are more difficult to fix because they are often caused by flaws or oversights in program design, such as assigning a variable value at the wrong place, failing to anticipate inappropriate input values or using an incorrect problem solution method.

17

The HP-75 has a number of debugging and error-processing features for pinpointing and controlling run-time and logical errors. The TRACE FLOW and TRACE VARS commands enable you to trace program branches and variable value changes. The [SHIFT] [RUN] keystroke enables you to single-step through programs. The ON ERROR statements enables you to write your own recovery routines.

Tracing Program Execution

A useful method of locating design errors is to trace the order of statement execution (using TRACE FLOW) and variable assignments (using TRACE VARS). All trace output is directed to the HP-75 display and current DISPLAY IS devices as determined by the DELAY rate.

Tracing Branches (TRACE FLOW)

The TRACE FLOW statement is used to trace the order of statement execution in all or part of a program.

```
TRACE FLOW
```

If the order of program execution proceeds sequentially from the lowest-numbered line to the next higher-numbered line, nothing is displayed. But whenever a branch occurs in a program, both the line number where the branch occurs and the number of the line to which execution branches are displayed in the form:

TRACE *line number* TO *line number*

TRACE FLOW does not, however, trace CALL statements. If one program calls another, no trace message will be displayed, either on the call or on the return to the calling program.

Tracing Variables (TRACE VARS)

The TRACE VARS command enables you to trace the value changes of program variables.

```
TRACE VARS
```

Whenever a program variable is assigned a value, the trace output indicates the line number where the assignment took place and:

- The name and assigned value of a simple numeric variable. Example:

```
Trace line 10 B=2
```

- The name of a string variable. Example:

```
Trace line 20 S$
```

- The name, subscript(s), and assigned value of a particular array element. Example:

```
Trace line 30 C(2,3)=45
```

- The name of an array in a READ # statement, in the form A() or C(,), and the value of the array. Example:

```
Trace line 400 A()=75
```

TRACE VARS enables you to verify that variables are being properly assigned values at the right places.

Cancelling Trace Operations (TRACE OFF)

Trace operations are cancelled by executing the TRACE OFF statement:

```
TRACE OFF
```

Executing TRACE OFF once will cancel a TRACE FLOW condition, a TRACE VARS condition, or both conditions.

Using TRACE Commands

The TRACE commands can be programmed or executed directly from the keyboard. Because the commands act globally, the current trace condition remains in effect until another TRACE command is executed. In addition, TRACE VARS and TRACE FLOW act independently of each other; either or both conditions can be in effect at any time.

To trace program execution, set either a TRACE FLOW or TRACE VARS condition, or both, and then run the program.

Example: Trace the branches of the following base conversion program as it computes the octal representation (base 8) of the binary number 10101010.

```
 10 ! --Number base conversion--
 20 DIM B$[16],F$[24]
•30 B$='0123456789ABCDEF'
 40 INPUT 'Input base, output bas
 e: ';B1,B2
 50 DISP 'Number in base '&STR$(B
 1);
•60 INPUT J$
 70 N=0
 80 FOR J=1 TO LEN(J$)
 90 P=POS(B$[1,B1],UPRC$(J$[J,J])
 )
•100 IF P=0 THEN 260
•110 N=B1*N+P-1
 120 NEXT J
•130 F$=''
•140 N=N/B2
•150 P=B2*FP(N)+1
•160 F$=F$&B$[P,P]
 170 N=INT(N)
•180 IF N#0 THEN 140
•190 DISP J$;' base';B1
 200 WAIT 2
 210 FOR K=LEN(F$) TO 1 STEP -1
 220 DISP F$[K,K];
 230 NEXT K
 240 DISP ' base';B2
 250 STOP
 260 BEEP
 270 DISP J$[J,J];' not allowed i
 n base';B1
 280 WAIT 2
•290 GOTO 50
```

Assigns check string.

Inputs number in first base.

If illegal character, jumps to line 260.
Accumulates equivalent in base 10.

Initializes output string.
Shifts one digit to the right.
Gets character.
Builds string in reverse order.

Checks if done.
Displays the original input.

} Displays the answer.

For another input.

After entering the program, set a TRACE FLOW condition from the keyboard; type `trace flow` RTN. Then press RUN:

```
Input base, output base: ■
Number in base 2?■
Trace line 120 to 80
Trace line 120 to 80
Trace line 120 to 80
Trace line 120 to 80
Trace line 120 to 80
Trace line 120 to 80
Trace line 120 to 80

Trace line 180 to 140
Trace line 180 to 140

10101010 base 2
2 Trace line 230 to 210
5 Trace line 230 to 210
2 base 8
```

To convert from binary to octal, type 2,8 [RTN].
For this example, type 10101010 [RTN].

The program executes the same set of branching operations, from 120 to 80, seven times in order to convert the original number to its decimal equivalent.

Then the program executes lines 140 through 180 three times (two branches from 180 to 140) to build the character string of the octal equivalent.

Line 190 displays the original number and base.

Finally, the string 252 is displayed one character at a time with the FOR-NEXT loop, lines 210 through 230.

The program exits the FOR-NEXT loop in statements 80 through 120 when the decimal equivalent of the original value has been accumulated and stored in the variable N. Note that execution returns to the FOR statement in line 80 and in line 210 not to reexecute the FOR statement (which would reset the loop counter to its initial value) but to test for the loop exit condition.

You can use any of the TRACE commands within a program. For instance, add the following lines to the base conversion program.

```
75 TRACE VARS
125 TRACE OFF
```

Now execute TRACE OFF from the keyboard to cancel the previous TRACE FLOW condition and run the program to find the decimal equivalent of FF_{16}. (Remember that in base 16, digits range from 0 through F, corresponding to decimal values 0 through 15.)

```
Input base, output base: ■
Number in base 16?■
Trace line 80 J=1
Trace line 90 P=16
Trace line 110 N=15
Trace line 120 J=2
Trace line 90 P=16
Trace line 110 N=255
Trace line 120 J=3

ff base 16
255 base 10
```

Enter 16,10 and press [RTN].
Then type ff [RTN].

Since the values of variables F$ and B$ do not change between statements 75 and 125, no TRACE VARS output occurs for them.

Remember that a TRACE command sets a trace condition until another TRACE OFF command is executed, either from the keyboard or from within a program.

Single-Step Execution (SHIFT RUN)

The SHIFT RUN keystroke is used to single-step through initialized programs. Pressing SHIFT RUN displays the next line to be executed; releasing the keystroke executes all the statements of that line.

Example: Single-step through the number base conversion program, beginning from the first INPUT statement, as the programs converts 3_{10} to binary. First delete the TRACE VARS command in line 75 so that statements can be viewed without trace output. Then press RUN to initialize the program:

```
Input base, output base: ■
                    PRGM
```

At this point, press ATTN to interrupt the program.

Begin single-stepping: Press SHIFT RUN and *hold down the* RUN *key*:

```
40 INPUT 'Input base, output bas
                    PRGM
```

The next line to be executed is displayed for as long as the RUN key is pressed.

Then release the keystroke:

```
Input base, output base: ■
                    PRGM
```

The INPUT statement is executed. The HP-75 waits for your response. Type 10,2 RTN to convert decimal to binary.

After your response to the INPUT statement, the program stops at the line following the INPUT statement. Press SHIFT RUN to view and execute it:

```
50 DISP 'Number in base '&STR$(B
                    PRGM
```

The program line is displayed until the RUN key is released...

```
Number in base 10
```

Then the DISP statement is executed.

Press SHIFT RUN again:

```
60 INPUT J$
                    PRGM
```

To be executed next.

Release the keystroke:

```
?■
                    PRGM
```

Type 3 RTN in response to the INPUT statement.

Press SHIFT RUN once more:

```
70 N=0
                    PRGM
```

When you release the keystroke this time, the **PRGM** annunciator turns off and line 70 stays in the display.

The HP-75 continues to display program lines after you release SHIFT RUN unless program execution directs any output to the display window.

After releasing the keystroke, you can use the ← and → keys and the modifiers SHIFT and CTL to examine long program lines. Follow the number base program until its conclusion. The statements are displayed in the order of their execution, which includes a branch from line 180 to line 140. Note that the **PRGM** annunciator stays on while SHIFT RUN is being pressed as well as while the statements are being executed. For example, the WAIT statement in line 200 causes the annunciator to stay on for 2 seconds. You may type cont RTN at any time to allow program execution to continue uninterrupted.

Programs must be initialized before you press SHIFT RUN the first time; otherwise, error 31—CONT before RUN—will occur. The recommended method is to place a STOP statement where you want to begin single-stepping—as at line 0 to begin single-stepping from the beginning of a program—and then press RUN. When the STOP statement interrupts execution, press SHIFT RUN to begin single-stepping from that point.

It's possible and often useful to check and change variable values while single-stepping through program execution.

Assuming that the initialized program is also the file that you're currently editing, then pressing SHIFT RUN sets both the program pointer *and* the file pointer to the next line that will be executed (the line after the program line currently in the display). Pressing FET and then RTN will fetch this line to the display, ready for editing. Remember, however, that editing any line will deallocate the program.

When one program calls another, SHIFT RUN will single-step through the execution of the called program, beginning from the CALL statement. When this occurs, the first program line in the called program will be executed before the halt. However, assuming that the calling program is also the current BASIC file, the file pointer stays fixed at the CALL statement until execution returns to the calling program.

When a program contains an ON TIMER # statement, single-stepping the line containing the ON TIMER # will *not* restart the timer. Consequently, the timer interrupt will not occur and the timer instructions will not be executed.

SHIFT RUN may be used in combination with the TRACE commands. Pressing SHIFT RUN displays the program statement; releasing the keystroke executes the statement and displays any trace output.

Checking a Halted Program

Various debugging operations can be performed on a program that's been interrupted by ATTN, STOP, or an error condition.

- Values of variables can be checked by keying in the variable names followed by RTN.
- Values of variables can be reassigned from the keyboard, for example, A(5,2)=7 and B=0.
- Calculator expressions can be evaluated, for example, SIN(4.4) RTN.
- System commands can be executed, for example, DISPLAY IS *, PLIST, DELAY, COPY, and OPTION ANGLE DEGREES.

None of these operations affects the position of the program pointer, which is set to the statement where the interruption in execution occurred. If the interrupted program is the current BASIC file, then the file pointer is also set to the line where the interruption occurred; otherwise, the file pointer stays fixed at the last line to be edited.

Remember that variable values from the currently halted program take precedence over calculator variables. If two variables have the same name, one having been assigned in a program and one from the keyboard, you will be referencing the *program* variable by that name until the program is deallocated.

Processing Run-Time Errors (ON ERROR, OFF ERROR)

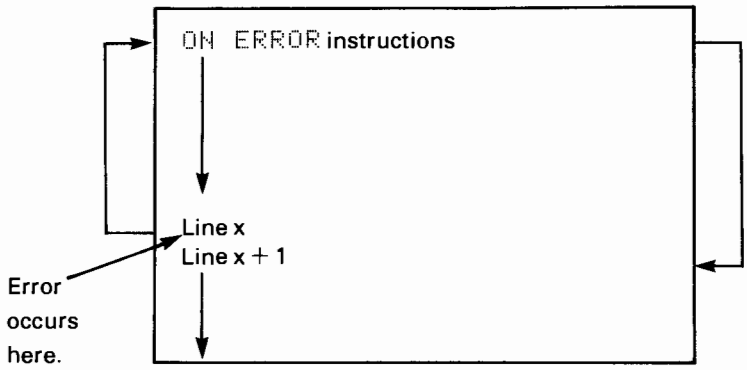
Run-time errors are those that occur after initialization, while a program is actually running. Examples are READ # statements that cause the HP-75 to read past the end of a data file and LET statements that assign more characters to a string variable than its length allows. The HP-75 may accept a faulty statement or command as you're writing a program but will catch that error when it's executed. Run-time errors normally halt execution. Although the DEFAULT ON command (page 89) normally provides default values for some error-causing operations (such as division by zero), these operations are always detected and treated like other errors when an ON ERROR has been declared.

The ON ERROR statement enables you to control the way run-time errors are processed so that programs can "trap" specific errors, display customized error messages, provide alternative parameter values, and continue execution. Program segments that handle run-time errors are called error recovery routines.

```
ON ERROR [allowable statement...] or [command...]
```

ON ERROR can include any command or any BASIC statement allowed after THEN.

After an ON ERROR declaration, if a run-time error occurs, execution transfers from the error-causing statement to the ON ERROR statement. The instruction or instructions in the ON ERROR statement are executed, then control returns to the first statement after the error-causing statement. ON ERROR overrides the assignment of default values to undefined variables. In those cases where a warning 7—no value—would occur, no default value is assigned to the variable.



After the ON ERROR instructions are performed, execution returns to the line following the error condition.

```
:
1010 ON ERROR A=0 @ RESTORE
•1020 READ A




1030 DISP @ WAIT 2
:
```

If the data pointer were past the last DATA item, this statement would cause the ON ERROR instruction in line 1010 to be executed—A would be assigned the value 0, the data pointer would be reset to the first DATA item in the file, and execution would continue at line 1030.

Note that execution returns to the *line*, rather than the statement, after the error; thus, valid statements following an error in a multistatement line will not be executed.

Only one ON ERROR declaration may be in effect at a time. An ON ERROR declaration may be replaced by another or may be disabled with an OFF ERROR statement.

```
OFF ERROR
```

Executing OFF ERROR causes subsequent errors to be handled normally by the operating system.

Example:

```
1010 ON ERROR A=0 @ RESTORE @ OF
F ERROR
```

The last instruction cancels the ON ERROR declaration.

Recovery Routines (ON ERROR GOTO, ON ERROR GOSUB)

Most ON ERROR declarations will include a GOTO or GOSUB statement to transfer execution to an error recovery routine at some other location in the program. When executed as an ON ERROR instruction, GOTO causes an unconditional branch to the specified program line.

Example:

```
:
 1000 S=0
 1010 ON ERROR GOTO 1050
•1020 READ A


 1030 S=S+A
 1040 GOTO 1020
•1050 OFF ERROR

 1060 DISP S
:
```

When the data pointer moves past the last DATA item, this statement causes an error condition, resulting in the execution of the GOTO 1050 instruction.

After the error, execution continues sequentially from here.

When a branch occurs to the line specified by ON ERROR GOTO, execution will continue from that point in the program and *not* return to the next line after the error. Use ON ERROR GOTO to completely alter the flow of execution as a result of an error condition.

If the ON ERROR instructions contain a GOSUB statement, execution will be transferred to the beginning line of the specified subroutine. When a RETURN is encountered in the error recovery routine, execution will return to the end of the ON ERROR statement to look for further instructions. Afterwards, execution will resume at the line after the one in which the error occurred. If the error occurs in the middle of a multistatement line, execution still returns to the next line—not to the next statement.

ON ERROR Is used to trap run-time errors, as opposed to syntax errors and initialization errors. In general, any condition that will cause the message ERROR to appear in the display after a program has been initialized will cause a branch to the ON ERROR instructions. Note that all mathematical errors—even those that display WARNING messages with DEFAULT ON—will cause a branch to the ON ERROR instructions.

Other Considerations

For precise control of errors, try to anticipate specific errors and then place an ON ERROR declaration in the immediate area where the error is likely to occur. As soon as the error has been "trapped," cancel the ON ERROR declaration with OFF ERROR so that you don't try to handle unexpected errors.

An ON ERROR declaration remains in effect for the program that executed it until replaced with another ON ERROR declaration or disabled with an OFF ERROR statement. ON ERROR and OFF ERROR statements act locally, affecting only the program in which they appear. When one program calls another, the ON ERROR statement of the first will be temporarily disabled until execution returns to the calling program. Attempting to execute ON ERROR from the keyboard causes error 88—bad statement—to occur.

An ON ERROR routine may not call itself. That is, if an ON ERROR routine itself contains an error, execution will halt.

An ON ERROR interrupt, like an ON TIMER # interrupt, creates a pending subroutine condition that is completed when all of the ON ERROR instructions have been executed. If the cause of the ON ERROR interrupt is error 49—GOSUB overflow—then an excess number of pending return conditions have already been created. In this one case, the ON ERROR statement will report error 29—ON ERROR overflow—and halt execution rather than execute the ON ERROR instruction.

One powerful form of the ON ERROR statement is simply ON ERROR RETURN. After this declaration, any error in the program will cause execution to continue at the next line of the program. In effect, ON ERROR RETURN declares "execute the next line of the program no matter what error occurs." However, remember to use ON ERROR declarations judiciously. On the one hand, they enable a program to recover on its own from run-time errors; on the other, they may hide many errors, making error detection difficult.

Error Numbers and Lines (ERRN, ERRL)

Two numeric functions, ERRN and ERRL, can be used with ON ERROR routines to determine the type and location of an error or warning. In turn, the type and location of an error can be used to control the direction of program execution.

Error Number Function. ERRN returns the number of the most recent error or warning.

Error Line Function. ERRL returns the line number in which the most recent error or warning occurred.

The following program contains some obvious errors for demonstration purposes.

```
 10 M=INF
 20 DISP M
•30 ON ERROR GOTO 80
 40 K=M+M
•50 OFF ERROR
 60 DISP K
 70 GOTO 120
•80 OFF ERROR

•90 IF ERRN=2 THEN 120

•100 DISP 'Error=';ERRN;'on line'
 ;ERRL
•110 STOP
•120 DISP 'Arrived at line 120.'
```

If an error occurs, jump to line 80.

Cancel the ON ERROR declaration.

Another place where the error mechanism can be disabled—the first line of the ON ERROR routine.

If error number 2—num too large—then jump to line 120.

If not an overflow, display the error and line number.

Then stop.

If an overflow condition occurs, execution will skip to this line.

Now run the program:

```
 9.9999999999E499
Arrived at line 120.
```

An overflow condition (warning 2—num too large) occurred in line 40 so that the GOTO instruction of the ON ERROR statement was executed. To see this more clearly, set a DELAY rate of 2 seconds and then set a TRACE condition. Type delay 2 [RTN] followed by trace flow [RTN]. Now run the program:

```
 9.9999999999E499
Trace line 40 to 30
Trace line 30 to 80

Trace line 90 to 120
Arrived at line 120.
```

The error occurs in line 40.
The ON ERROR statement directs execution to line 80.
ERRN is 2, so a final branch occurs.

Now change statement 40 to read:

```
40 K=M/0
```

Creates a division by zero error.

Run the program again with TRACE FLOW set:

```
 9.9999999999E499
Trace line 40 to 30
Trace line 30 to 80
Error= 8 on line 40
```

Displays error number for division by zero and the line number in which the error occurred, then stops.

Since the program remains initialized, you may perform various program checks, or change the values of variables before you continue.

ERRN and ERRL are most useful when you can anticipate the kinds of errors that are likely to occur in a given program segment.

Accessories Included With the HP-75

Your HP-75 comes with each of the following:

- *HP-75 Owner's Manual.*
- *HP-75 Reference Manual.*
- HP-75 Owner's Pac.
  Eight prerecorded program cards, one abrasive head cleaning card, and 10 blank cards.
- Service Card.
- Field Case.
- Rechargeable Nickel-Cadmium Battery Pack (model HP 82001B).
- One AC Adapter/Recharger (model HP 82059B).
- One One-Meter HP-IL Cable (model HP 82167B). (Each HP-IL device has a matching cable.)

If you are outside the U.S., please contact the Hewlett-Packard Sales Office nearest you. Availability of all accessories, standard or optional, is subject to change without notice.

Appendix B

Owner's Information

Contents

Serial Number and Operating System Version (VER$) 267
Environmental Limits 267
Operating Precautions 267
Clock Accuracy 268
Conformance of BASIC Interpreter to ANSI Standards 268
  HP-75 Extensions to Minimal BASIC 268
  HP-75 Deviations from Minimal BASIC 269
Power Supply Information 269
  Power Consumption 269
  Low-Battery Safeguards 270
  Battery Recharging 270
  Replacing the Battery Pack 271
  Rechargeable-Battery Care 272
Card Reader and Card Care 273
  Cleaning Magnetic Cards 273
  Cleaning the Card Reader Head 273
  Marking Magnetic Cards 273
General Cleaning Information 274
Plug-In Modules 274
Special Files 274
  Interchange Files (TRANSFORM, LIF1) 274
  LEX Files (COPY) 277
  ROM Files (RUN, LIST, COPY, MERGE) 277
  Other Files (?) 278
Verifying Proper Operation 278
Limited One-Year Warranty 279
  What We Will Do 279
  What Is Not Covered 279
  Warranty for Consumer Transactions in the United Kingdom 279
  Obligation to Make Changes 279
  Warranty Information 280
Service 280
  Service Centers 280
  Obtaining Repair Service in the United States 281
  Obtaining Repair Service in Europe 281
  International Service Information 281
  Service Repair Charge 282
  Service Warranty 282
  Shipping Instructions 282
  Further Information 282
Potential for Radio/Television Interference (for U.S.A. Only) 283
When You Need Help 283

B

Serial Number and Operating System Version (VER$)

Each HP-75 carries an individual serial number on the bottom of the unit. We recommend that you keep a separate record of this number. Should your unit be lost, the serial number is often necessary for tracing and recovery, as well as for insurance claims. Hewlett-Packard does not maintain records of individual owners' names and unit serial numbers.

The VER$ function returns a six-character string that indicates which version of the HP-75 operating system you are using. Type `ver$` [RTN] in EDIT mode to determine the operating system version of your unit.

Environmental Limits

Observe the following temperature and humidity limits of the HP-75.

Operating Temperature: 0° to 45°C (32° to 113°F).

Recharging Temperature: 10° to 40°C (50° to 104°F).

Storage Temperature: −40° to 55°C (−40° to 131°F).

Operating and Storage Humidity: 0 to 95 percent relative humidity.

Your computer should never be operated or stored outside of the specified range. For example, high temperatures are especially damaging to the computer and its batteries. Temperature cycling—from one extreme to another—will cause stresses in your computer that will also tend to decrease its reliability. Maximum reliability (that is, minimum failure rate) will occur at normal room temperatures.

Operating Precautions

Certain electronic circuits in the HP-75 operate continuously, and they are susceptable to disruption or damage at all times. Disruption or damage may be caused by installing or removing the battery pack or plug-in modules while the ac adapter/recharger is connected, by other electrostatic discharge to the unit, by strong magnetic fields, and by connecting plug-in equipment that is not supported by Hewlett-Packard for use with the HP-75. Observe the precautions listed below.

CAUTIONS

- Disconnect the ac adapter/recharger whenever you are going to install or remove the battery pack or a plug-in module.
- Hold or touch the computer while preparing to install the battery pack or a plug-in module. Touching the insert on the bottom of the computer is particularly effective for neutralizing an electrostatic charge.
- Do not place fingers, tools, or other foreign objects into any of the plug-in ports.
- Turn off the unit before installing or removing the battery pack or a plug-in module.

Clock Accuracy

The system clock is regulated by a quartz crystal accurate to within 3 minutes per month for worst-case operating temperatures. A more typical accuracy is 1½ minutes per month. The adjustment procedure described in section 6 (page 95) makes possible accuracies of 15 seconds per month. The clock crystal is affected by temperature, physical shock, humidity, and aging. Optimum accuracy is achieved at 25°C ±5° (77°F ±9°). When an extreme change in environment occurs, the clock may require readjustment.

Conformance of BASIC Interpreter to ANSI Standards

The HP-75 BASIC language interpreter conforms to the American National Standards Institute (ANSI) definition for Minimal BASIC except as indicated below. This language standard is listed as ANSI X3.60-1978. Conformance to the standard was verified by the application of the National Bureau of Standards (NBS) test suite to the HP-75 interpreter. This test suite is available as NBS Special Publications 500-70/1 and 500-70/2 from the National Bureau of Standards, U.S. Department of Commerce, Washington, D.C., 20234.

HP-75 Extensions to Minimal BASIC

The HP-75 extends Minimal BASIC on the following items (numbers in parentheses refer to the program number in the test suite):

- Simple numeric variables and numeric array variables may have the same name. The HP-75 distinguishes them by the presence or lack of parentheses. (#79)

  Example: Variables A and A() are seen as separate variables.

- Variables and strings are initialized to "no value"; reference to them before assignment returns warning 7—no value—and default values of 0 for numeric variables and '' (null string) for string variables are substituted in the expression.(#23)

- The character set has been expanded to include any of ASCII codes 0 through 255 (decimal) as valid character responses. (#93.1, #102, #112)

- The HP-75 accepts double and single quotes as input characters in an unquoted string if they are not the first character in the response. (#112)

- On program input, the HP-75 accepts blanks at the beginning of a line, in keywords (after the second character), and the lack of blanks between keywords. After a line has been entered, the interpreter removes extraneous blanks and inserts blanks where required for readability. (#187, #190, #191)

- In an assignment statement, the keyword LET is optional. (#185)

- The interpreter provides an invisible END statement at the physical end of a BASIC file; so the user need not supply an END statement in his program if flow will naturally go to the last line. In addition, the interpreter permits more than one END statement in a basic program. (#3, #4)

- Because the HP-75 preallocates variables and functions before a program is run, user-defined functions are not restricted to having lower line numbers than the line where they are referenced. (#157, #159, #162)

- The HP-75 permits null data items in DATA statements. (#105)

- The system will not generate an error if the program ends before a FOR statement has found a matching NEXT. (#50)

HP-75 Deviations From Minimal BASIC

The HP-75 does not comply with Minimal BASIC on the following two points:

- The HP-75 assigns the FOR variables left to right in the initial entry to the loop.

  Example: The following loop is executed once by the HP-75 (ANSI requires it to be executed six times):

```
J=-2
FOR J=9 TO J STEP J
⋮
NEXT J
```

  ANSI requires that the limit and step be evaluated once upon entering the loop. The HP-75 does this, but *after* setting the initial value. (#48)

- The HP-75 response to input errors and the ANSI requirements are tabulated below:

INPUT	HP-75		ANSI
	Default On	Default Off	
Numeric underflow	WARNING System supplies 0.	ERROR User must re-enter value.	System supplies 0.
Numeric overflow	WARNING System supplies signed INF.	ERROR User must re-enter value.	User must re-enter value.
Variable assignment	Occurs after each item verified.		Occurs after all verified.

  If a program contains the following statement:

```
INPUT I, A(I), X
```

  and the user response is:

```
1,2,'abc'
```

  then the 'abc' is invalid input and ANSI requires that I and A(I) not be assigned until all input is correct. The HP-75 will request the user to re-input his data, but I and A(I) will have the current values of 1 and 2 respectively. If the user now inputs 2,3,4 then A(1) has the value 2 and A(2) has the value 3. ANSI requires that A(1) is yet to be defined and A(2)=3. (#108.3)

Power Supply Information

Power Consumption

The HP-75 consumes least power when the display is turned off (after [SHIFT] [ATTN], BYE, or the 5-minute timeout period). More power is consumed while the computer is turned on, more yet while programs are running, and most while the beeper is sounding *or* a bar-code reading wand is operating. Power consumption is three to four times higher while the beeper is sounding than while the HP-75 is idle. In addition, executing standby on will increase idle state power consumption to approximately the level of a running program. While the HP-75 is not powered from an ac outlet, a fully charged battery pack typically allows more than 20 hours of continuous operation at room temperature (approximately 25° C).

Turn the HP-75 off (press [SHIFT] [ATTN]) before connecting the unit to a power outlet. This will prevent unexpected voltage "spikes" from disturbing the contents of memory. When connected to a power outlet, the HP-75 uses the battery pack as a backup power supply and normally doesn't draw any power from the

batteries. You won't damage the HP-75 by using the computer without a battery pack, *but* you may lose everything in memory should there be a power outage or an intermittent connection to the voltage source.

Batteries

To obtain maximum performance from the rechargeable batteries, observe these guidelines:

- Use only the recharger and batteries designated by Hewlett-Packard for use with your calculator.
- Be sure to recharge the batteries within 5 minutes to 2 hours of operation after the low battery indicator comes on. Allowing rechargeable batteries to discharge beyond their maximum limit can damage the batteries and possibly damage the product.

CAUTION

To avoid damage to your calculator, use only the batteries and recharger designated by Hewlett-Packard for your calculator. Also, do not allow the batteries to discharge beyond the recommended limit.

- Recharge the batteries only when the low battery indicator shows in the display. Recharging batteries before they are low can decrease their charging capacity for all future charges.
- Do not overcharge the batteries by allowing them to recharge for longer than 8 hours.

  Shorter charging times will reduce the operating time before recharging is required, but will not harm the batteries. Overcharging the batteries can reduce their useful life. In addition, overcharged batteries may vent a white powder. The batteries eventually reseal themselves.

WARNING

The powder vented from overcharged batteries can irritate the skin and eyes. Remove vented powder using a disposable towel and then properly dispose of the towel. Do not connect together or otherwise short-circuit the battery pack terminals—the pack may melt or cause serious burns.

- Do not leave the recharger permanently connected to the product. Doing so decreases the useful life of the batteries.
- Do not use your recharger if it appears to have loose contacts, a cracked housing, or damaged cord.
- Properly dispose of batteries when they no longer adequately hold a charge or when they appear damaged.

WARNING

To prevent injury, keep all batteries out of the reach of children and properly dispose of exhausted batteries. Do not mutilate or puncture batteries, and do not dispose of them in fire. Exposure to excessive heat can cause release of toxic fumes or explosion.

If a rechargeable battery pack seems to discharge very quickly, the problem may be that the battery pack has been charged repeatedly at shallow levels, due to the low current drain of the HP-75. Disconnect the unit from its power outlet, type `standby on` [RTN] in EDIT mode, and leave the HP-75 turned on until the battery voltage drops to the point at which the HP-75 suspends operations and turns itself off (as described in Batteries). Then reconnect the unit to a power outlet for 8 hours to recharge the battery pack fully.

The battery pack is warranted for 1 year and will be replaced if it doesn't work properly during the warranty period.

The HP-75 has two low-power safeguards to protect the contents of memory. After the first indication of low power, either recharge or replace the battery pack as soon as convenient.

When the battery voltage drops below a predetermined level, the **BATT** annunciator turns on, signifying from 5 minutes to 2 hours more of operation (run-time), depending on the condition of the batteries. If the voltage continues to drop after this time period, the HP-75 aborts its normal operations, including any currently executing program, beeper output, display window output, card reader operation, or HP-IL operation. However, the battery pack continues to power the clock and RAM circuits. If the [ATTN] key or a due appointment causes the HP-75 to turn back on, the HP-75 briefly displays the following message `WARNING: low batteries`. Immediately afterwards, the HP-75 turns off again. At this point, to prevent a complete reset, don't press the [ATTN] key before connecting the unit to an ac outlet or before replacing the battery pack. Clock settings and information in memory will be preserved for a few days so long as there are no attempts to turn the unit back on in this condition.

If the battery pack nears total discharge, the HP-75 will reset itself and lose everything in memory. A reset may also occur if an attempt is made to turn the unit on (by pressing [ATTN]) or to acknowledge an alarm on low batteries.

Replacing the Battery Pack

Whenever the battery pack is to be replaced, the ac adapter/recharger should be disconnected. When the battery pack is removed, the clock and RAM circuits are protected against loss for a minimum of 30 seconds. Because there is a limit to how long memory is protected against loss, you may want to study the following procedure before attempting to replace the battery pack. Observe the operating precautions listed earlier in this appendix (page 267). *Don't press the* [ATTN] *key* while the battery pack is out of the compartment. The HP-75 will reset itself if it tries to turn on without an adequate power source. You may wish to copy critical files to a mass storage medium before removing the battery pack to avoid the possibility of losing important information if a reset does occur.

Use the following procedure to replace the battery pack:

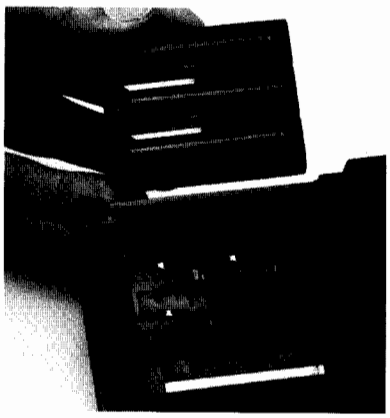
1. Type `alarm off` [RTN] in EDIT mode to prevent the arrival of future appointments while the battery pack is out of the computer.
2. *Turn off the HP-75.* Press [SHIFT] [ATTN], or type `bye` [RTN].
3. *Unplug the ac adapter/recharger from the HP-75.*
4. Turn the computer upside down and lay it on a flat, smooth surface.

5. Press down on the catch on the battery compartment door while sliding it toward the back of the computer.

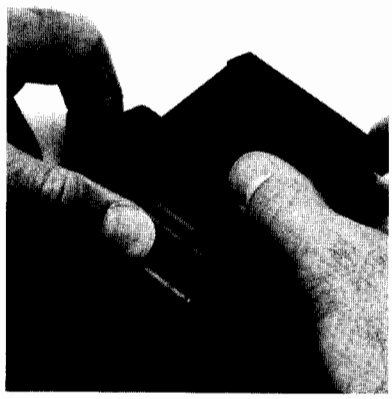
6. Remove the battery pack by lifting it out of the battery compartment.

7. Align the contacts of the replacement battery pack with the correct spring contacts and insert it into the compartment.

   Note: Be sure to use only the model HP 82001B battery pack. The earlier—model A—version of this pack will fit in the battery compartment, but will not make electrical contact with the battery compartment spring contacts.



8. Press down on the battery pack and replace the door by sliding it along the edge guides until it snaps into place.



The nickel-cadmium rechargeable battery pack uses the two parallel contacts in the battery compartment. If a battery pack is inserted backwards, no damage to the computer will occur, although the computer will reset if it's turned on in this condition.

Card Reader and Card Care

Cleaning Magnetic Cards

Card cleanliness is necessary for optimum card reader performance. Card surfaces are susceptible to dust and oil accumulation, which interferes with the transfer of information to and from the HP-75. A common source of dirt and oil is your fingers. Handle your cards by their edges only. You may clean your cards with a soft, clean, lint-free cloth moistened in isopropyl alchohol.

Place the card on a smooth, clean surface, then wipe the cleaning cloth firmly across the magnetic surface (the unlabeled side) of the card.

Creasing, bending, or scratching a card may damage the card beyond repair. You may wish to make backup copies of important files and keep the card copies protected in a card holder in case one of the original cards from the file is damaged.

Cleaning the Card Reader Head

The card reader head is similar to a high fidelity audio recording head. As such, any collection of dirt or other foreign matter on the head can interfere with contact between the head and card. Dirty cards passed through the card reader will impair the quality of its operation. You can clean the head by pulling the abrasive head cleaning card (supplied in the owner's pac) through the card reader in the direction of the arrow (as though you were reading it) one or two times if you suspect the card reader is not performing properly. (It's unnecessary to execute any command from the keyboard before inserting and pulling the head cleaning card.) Use of the abrasive card should be necessary no more than a few times during the life of the machine.

CAUTION

Frequent use of the abrasive card can cause excessive head wear.

Marking Magnetic Cards

You can label the face of a card using any writing implement that doesn't emboss the card. Permanent ink felt-tip pens (such pens usually have the words permanent, waterproof, or smearproof on them—don't use water-based overhead projector pens or ordinary felt-tip writing pens), capillary or technical drawing pens using permanent ink, and pencils work well in marking cards. Most inks must be allowed to dry for a few seconds. Pencil may smear but is erasable from the cards.

General Cleaning Information

The HP-75 can be cleaned with a soft cloth dampened either in clean water or in water containing a mild detergent. Don't use an excessively wet cloth or allow water inside the computer. Avoid abrasive cleaners, especially on the display window.

You may wish to press [SHIFT] [ATTN] to disable the keyboard before cleaning the HP-75.

Plug-In Modules

Your HP-75 has one internal port for an optional memory module and three external ports to accomodate plug-in ROM modules that extend the capabilities of your computer. Before shipping, each of the three external ports is fitted with a removable, blank module to protect the underlying circuits. These three ports should be kept covered when not in use to prevent foreign matter from entering the HP-75.

Note: Be sure to turn off the HP-75 (press [SHIFT] [ATTN]) before installing or removing any module. If you don't do this, the computer may be reset, causing all stored information to be lost.

CAUTIONS

Observe the operating precautions listed earlier in this appendix (page 267).

If a module jams when inserted in a port, it may be upside down. Attempting to force it further may result in damage to the computer or the module.

Instructions on the installation and use of the optional memory module and plug-in modules are provided in the documentation accompanying each extension.

Special Files

Three special types of files—interchange, LEX, and ROM files—are made available through the TRANSFORM command, through prerecorded magnetic cards and tapes, and through plug-in ROMs, respectively.

Interchange Files (TRANSFORM, LIF1)

Interchange files, or LIF1 (Logical Interchange Format) files, are used for interchanging information between the HP-75 and other computers. Any text or nonprivate BASIC file in memory may be transformed into an interchange file by means of the TRANSFORM command and afterwards copied to a mass storage medium for use by another computer. Similarly, an interchange file residing on a mass storage medium may be read into memory and then transformed into a BASIC or text file. Interchange files, like text files, are composed of characters.

Interchange files may be:

- Cataloged (CAT). The symbol I indicates an interchange file in the system catalog.
- Copied to and from mass storage (COPY).
- Transformed into BASIC and text files (TRANSFORM).
- Renamed (RENAME), purged (PURGE), and duplicated in memory (COPY).

Interchange files may not be:

- Executed as programs (RUN, CALL).
- Edited (EDIT).
- Listed (LIST, PLIST).

Attempting to run, edit, or list an interchange file causes error 68—`wrong file type`—to occur.

Use the `TRANSFORM` command to transform one type of file in memory—BASIC, text, or interchange—into another file type.

```
TRANSFORM ['filename'] INTO BASIC
                           TEXT
                           LIF1
```

For example, if `'MONEY'` is a BASIC file, then typing `transform 'money' into text` RTN transforms the `MONEY` file into a text file. The catalog entry for the file will show type `T` for text. Although a listing of the transformed text file will look the same as a listing of the original BASIC file, the two are not the same. Text files are stored internally as line numbers and characters, while BASIC files are stored internally as a stream of machine code instructions. The reverse operation, `transform 'money' into basic`, transforms the file into a BASIC program file, ready to run.

BASIC and text files may be transformed into interchange files, just as interchange files may be transformed into BASIC and text files.

The action of the `TRANSFORM` command is summarized below:

Source File	TRANSFORM INTO		
	BASIC	**Text**	**LIF1**
BASIC	No action.	The machine code is transformed into lines of text.	The machine code is transformed into lines of text and then transformed line by line into character information.
Text	The file is transformed line by line into machine code.	No action.	The file is transformed line by line into character information.
LIF1	The file is transformed character by character into lines of text and then transformed line by line into machine code.	The file is transformed character by character into lines of text.	No action.

It may take several seconds for a file to be transformed, depending on the length of the file. During the transformation, the entire keyboard, including the ATTN key, will be disabled to prevent a file of mixed types from occurring. Whatever is in the display when the `TRANSFORM` command is executed will remain in the display during the transformation. The prompt and the cursor will appear when the operation is completed.

The filename in the `TRANSFORM` command may be specified by a string expression, such as `A$` and `CAT$(0)`. If no filename is specified, then the HP-75 will transform the current file:

- If the current file is transformed into text or BASIC, then the text or BASIC prompt, respectively, will appear afterwards and the file pointer will be set to the first line of the file.
- If the current file is transformed into LIF1, then a new `workfile` of the current type, text or BASIC, will be created and the file pointer will be set to line 0 of the `workfile`.
- If the current file is itself a `workfile`, then `TRANSFORM INTO TEXT` and `TRANSFORM INTO BASIC` will be allowed, and the resultant file will be a text or BASIC `workfile`. However, `TRANSFORM INTO LIF1` will not be allowed and will cause error 63—`invalid filespec`—to be displayed; in this case, give the file a name and reexecute the `TRANSFORM` command.

When a text or interchange file is transformed into a BASIC file, then each line of the transformed file must make sense as a program instruction or instructions. If the HP-75 encounters an unrecognizable sequence of characters while transforming a file into BASIC, then the HP-75 transforms the entire line containing those characters into a *program remark*. The HP-75 sets off the line by placing an exclamation mark followed by a question mark (`! ?`) after the line number but before the characters in the line.

Example:

```
110 A=A MOD 256
```

This group of characters cannot be interpreted as a valid HP-75 BASIC statement...

```
110 ! ? A=A MOD 256
```

so the entire line is transformed into a program remark.

If the resulting BASIC file contains one or more program remarks of the above form, the HP-75 will report error 78—`syntax`—after the transformation is completed. When you edit the transformed BASIC file, it's possible to locate each line that was not understood by using a form of the `FETCH` command.

Example:

```
>FETCH '! ?', 0
```

Begins a search from line 0 of the file for the next occurrence of `! ?`.

```
110 ! ? A=A MOD 256
```

This line may now be revised and entered as an HP-75 BASIC statement.

On the other hand, if you transform a BASIC file into a text or LIF1 file, then any lines of the source program beginning with `! ?` (with or without embedded blanks) will be entered in the text or LIF1 file *without* the remark notation. You may choose not to use a question mark as the initial character of a program remark to avoid having the remark notation `! ?` stripped during a transformation into text or LIF1.

During a BASIC-to-text transformation, if any line exceeds 94 characters (including line number), then all characters in the source BASIC line after the first 94 are deleted, warning 67—`line too long`—is displayed, and the transformation continues. The resulting text line will consist of a question mark in the first column after the line numbers, followed by the first 94 characters of the source BASIC line. This situation will arise if, for example, you transform a BASIC file with long `DATA` statements into text.

During a text-to-BASIC transformation, it's possible that one or more of the resulting program lines will exceed 94 characters when listed, as will happen if a line in the source text file contains numerous abbreviations.

Example:

```
:10op.a.d.@op.a.d.@op.a.d.@op.a
.d.@op.a.d.
```

When transformed into BASIC, this line of text will result in five `OPTION ANGLE DEGREES` commands in one long line.

No error will be reported during the transformation, and all the instructions in the line will be stored in the BASIC file. When you attempt to edit or list the resultant long line, warning 67—`line too long`—will occur; only the first 94 characters in the line will be displayed and be available for editing. If the file is now transformed back to text, the long line will be truncated after the 94th character.

It's also possible to transform the `keys` file. First, rename the `keys` file to another filename, and then use *that* filename in the `TRANSFORM` command.

Interchange files created by computers other than the HP-75 may contain lines longer than allowed by the HP-75 or may have unnumbered lines. Such files may be copied to memory, but attempting to transform them to text or BASIC will not work. An interchange file with excessively long lines will cause error 67—`line too long`; an interchange file with missing line numbers will cause error 87—`bad expression`. In either case, no action will be taken; the LIF1 file will remain a LIF1 file.

If the currently running program transforms itself, then execution is interrupted, the program is deallocated, and the transform occurs.

The `TRANSFORM` command requires some system memory for execution; consequently some valid nested expressions may be converted into `! ?` comment lines when they are transformed from text or LIF1 files into BASIC.

LEX Files (`COPY`)

LEX files, or Language Extension files, are specially coded programs that extend the capabilities of the HP-75. A LEX file may reside on a mass storage medium or in a plug-in ROM. If on a mass storage medium, LEX files are copied to memory (RAM) the same way as other files, using a form of the `COPY` command. A LEX file in RAM is indicated by the symbol `L` in the system catalog. A LEX file in ROM will not be reported in the system catalog, but will function identically to a LEX file in RAM.

LEX files make new keywords available to the user, including new functions, statements, and commands that are not part of the HP-75 operating system. LEX files are not available for listing, editing, or running, but LEX files in RAM may be duplicated, renamed, and purged.

Specific instructions for using LEX files accompany the medium that stores the files. Note that if you write a BASIC program that uses any of the keywords provided by a LEX file, then that LEX file should be present whenever you work with the program file. If the LEX file isn't present: running the program causes error 18—`ROM missing`; if the program isn't initialized, lines containing LEX keywords can't be accessed (but other lines can); if the program is initialized, lines containing LEX keywords are replaced by lines containing `ROM missing` if the program is deallocated by the operation.

ROM Files (`RUN`, `LIST`, `COPY`, `MERGE`)

ROM files reside permanently in the plug-in ROM (read-only memory) modules available as optional accessories for the HP-75. ROM files may be of any type, including BASIC, text, and LEX. BASIC ROM files may be run and called; LEX ROM files make new keywords available to the system as soon as the ROM module is plugged into the HP-75.

ROM files may not be cataloged, edited, listed, purged, renamed, or changed in any way, nor can you use `READ #` to read a ROM file. However, individual ROM files—if nonprivate—may be copied to memory using the `COPY` command.

```
COPY 'ROM filename' TO 'RAM filename'
```

In addition, all or any part of a BASIC or text ROM file may be merged into the current file in memory using the `MERGE` command.

```
MERGE 'ROM filename' [,beginning line number [,ending line number]]
```

Once in RAM, the file may be cataloged, edited, listed, renamed, and so on. To copy a ROM file to a mass storage medium, you must first copy the file to memory. Note that if two files have identical names, one in ROM and one in RAM, then when you specify the filename in a command, the HP-75 will access the file in RAM.

For specific operating instructions, refer to the documentation for the ROM.

Other Files (?)

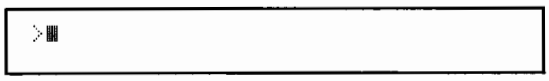
A question mark, ?, as the file type of a file on a mass storage medium indicates that the HP-75 is unable to use the information contained in that file. A file of this type may result from a copy-to-mass storage operation of another calculator or computer.

Attempting to copy a ? file to HP-75 memory will cause error 68—`wrong file type`. Unknown files may be renamed (`RENAME`), duplicated on the mass storage medium (`COPY`), and erased from mass storage (`PURGE`).

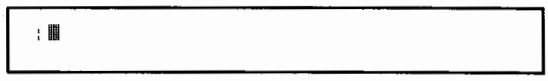
Verifying Proper Operation

If you suspect that your HP-75 is not operating properly and may require service, you can do the following:

1. Turn the unit off by pressing [SHIFT] [ATTN], and disconnect all HP-IL devices.
2. Plug the unit into a power outlet using the ac adapter/recharger.
3. Turn the unit back on by pressing [ATTN]. One of these two displays should result:

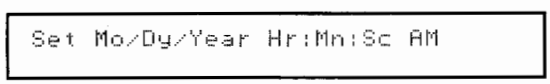


The BASIC prompt and cursor.



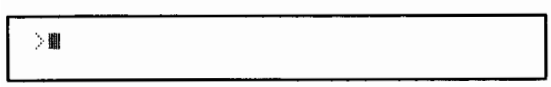
The text prompt and cursor.

4. Execute the `PI` function. Type `pi` [RTN]. The result `3.14159265359` should be displayed, which indicates that approximately 60 percent of the system circuits are performing properly.
5. If the computer repeatedly fails to perform a particular operation, such as copying a file to a magnetic card, or repeatedly displays an error message, such as `bad parameter`, then carefully reread the instructions in this manual regarding that operation; you may be specifying the operation improperly.
6. If the display remains blank when [ATTN] is pressed, then reset the machine:
   a. Unplug the ac adapter/recharger.
   b. Remove the battery pack.
   c. Press [ATTN]. The circuits will be discharged immediately.
7. After you reinstall the battery pack and reconnect the ac adapter/recharger, the set-time template should appear when [ATTN] is pressed:



Enter the values for the date and time, and press [RTN].

8. After setting the clock, press the [EDIT] key:



The BASIC prompt and the cursor should appear. Now repeat step 4.

9. If the display remains blank, or if the message `ERROR: RAM is invalid` is displayed, then remove all plug-in RAM and ROM modules, reset the machine, and repeat steps 7 and 8 to determine whether the problem is caused by a defective or improperly installed module.
10. If the HP-75 becomes inoperable with characters "frozen" in the display, then reset the computer by pressing [SHIFT] [CTL] [CLR] or removing the battery pack while the unit is not connected to a power source. After the reset, verify proper operation, beginning at step 7.
11. To verify the computer's HP-IL operation, connect a single HP-IL cable between the HP-75's IN and OUT receptacles. Then type `assignio` [RTN]. If error 56—`no loop response`—occurs, the HP-IL circuit is operating properly; other responses indicate a bad cable or improper operation.

If you cannot determine the cause of difficulty, write or telephone Hewlett-Packard at an address or phone number listed below under service.

Limited One-Year Warranty

What We Will Do

The HP-75 is warranted by Hewlett-Packard against defects in materials and workmanship for one year from the date of original purchase. If you sell your unit or give it as a gift, the warranty is transferred to the new owner and remains in effect for the original one-year period. During the warranty period, we will repair or, at our option, replace at no charge a product that proves to be defective, provided you return the product, shipping prepaid, to a Hewlett-Packard service center.

What Is Not Covered

This warranty does not apply if the product has been damaged by accident or misuse or as the result of service or modification by other than an authorized Hewlett-Packard service center.

No other express warranty is given. The repair or replacement of a product is your exclusive remedy. **ANY OTHER IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS IS LIMITED TO THE ONE-YEAR DURATION OF THIS WRITTEN WARRANTY.** Some states, provinces, or countries don't allow limitations on how long an implied warranty lasts, so the above limitation may not apply to you. **IN NO EVENT SHALL HEWLETT-PACKARD COMPANY BE LIABLE FOR CONSEQUENTIAL DAMAGES.** Some states, provinces, or countries do not allow the exclusion or limitation of incidental or consequential damages, so the above limitation or exclusion may not apply to you.

This warranty gives you specific legal rights, and you may also have other rights which vary from state to state, province to province, or country to country.

Warranty for Consumer Transactions in the United Kingdom

This warranty shall not apply to consumer transactions and shall not affect the statutory rights of a consumer. In relation to such transactions, the rights and obligations of Seller and Buyer shall be determined by statute.

Obligation to Make Changes

Products are sold on the basis of specifications applicable at the time of manufacture. Hewlett-Packard shall have no obligation to modify or update products once sold.

Warranty Information

If you have questions about this warranty, please contact Hewlett-Packard at one of the following locations.

In the United States. Call (503) 757-2002 or write to:

Hewlett-Packard Co.
Calculator Service Center
1030 N.E. Circle Blvd.
Corvallis, OR 97330

In Europe. Call (022) 83 81 11 or write to:

Hewlett-Packard S.A.
150, route du Nant-d'Avril
P.O. Box CH-1217 Meyrin 2
Geneva
Switzerland

Note: Do *not* send computers to this address for repair.

In other countries: Call (415) 857-1501 in the U.S.A. or write to:

Hewlett-Packard Intercontinental
3495 Deer Creek Road
Palo Alto, California 94304
U.S.A.

Note: Do *not* send computers to this address for repair.

Service

Service Centers

Hewlett-Packard maintains service centers in most major countries throughout the world. You may have your unit repaired at a Hewlett-Packard service center any time it needs service, whether the unit is under warranty or not. There is a charge for repairs after the one-year warranty period.

Hewlett-Packard computer products normally are repaired and reshipped within five (5) working days of receipt at any service center. This is an average time and could vary depending on the time of year and work load at the service center. The total time you are without your unit will depend largely on the shipping time.

Obtaining Service in the United States

For service in the United States, ship your HP-75 module to:

Hewlett-Packard Co.
Calculator Service Center
1030 N.E. Circle Blvd.
Corvallis, Oregon 97330

or mail it to:

Hewlett-Packard Co.
Calculator Service Center
P.O. Box 999
Corvallis, Oregon 97339

The telephone number for the Calculator Service Center is (503) 757-2002.

Obtaining Repair Service in Europe

Service centers are maintained at the following locations. For countries not listed, contact the dealer where you purchased your computer.

AUSTRIA
HEWLETT-PACKARD Ges.m.b.H.
Kleinrechner-Service
Wagramerstrasse-Lieblgasse 1
A-1220 Wien (Vienna)
Telephone: (0222) 23 65 11

BELGIUM
HEWLETT-PACKARD BELGIUM SA/NV
Woluwedal 100
B-1200 Brussels
Telephone: (02) 762 32 00

DENMARK
HEWLETT-PACKARD A/S
Datavej 52
DK-3460 Birkerod (Copenhagen)
Telephone: (02) 81 66 40

EASTERN EUROPE
Refer to the address listed under Austria.

FINLAND
HEWLETT-PACKARD OY
Revontulentie 7
SF-02100 Espoo 10 (Helsinki)
Telephone: (90) 455 02 11

FRANCE
HEWLETT-PACKARD FRANCE
Division Informatique Personnelle
S.A.V. Calculateurs de Poche
F-91947 Les Ulis Cedex
Telephone: (6) 907 78 25

GERMANY
HEWLETT-PACKARD GmbH
Kleinrechner-Service
Vertriebszentrale
Berner Strasse 117
Postfach 560 140
D-6000 Frankfurt 56
Telephone: (611) 50041

ITALY
HEWLETT-PACKARD ITALIANA S.P.A.
Casella postale 3645 (Milano)
Via G. Di Vittorio, 9
I-20063 Cernusco Sul Naviglio (Milan)
Telephone: (2) 90 36 91

NETHERLANDS
HEWLETT-PACKARD NEDERLAND B.V.
Van Heuven Goedhartlaan 121
N-1181 KK Amstelveen (Amsterdam)
P.O. Box 667
Telephone: (020) 472021

NORWAY
HEWLETT-PACKARD NORGE A/S
P.O. Box 34
Oesterndalen 18
N-1345 Oesteraas (Oslo)
Telephone: (2) 17 11 80

SPAIN
HEWLETT-PACKARD ESPANOLA S.A.
Calle Jerez 3
E-Madrid 16
Telephone: (1) 458 2600

SWEDEN
HEWLETT-PACKARD SVERIGE AB
Skalholtsgatan 9, Kista
Box 19
S-163 93 Spanga (Stockholm)
Telephone: (08) 750 20 00

SWITZERLAND
HEWLETT-PACKARD (SCHWEIZ) AG
Kleinrechner-Service
Allmend 2
CH-8967 Widen
Telephone: (057) 31 21 11

UNITED KINGDOM
HEWLETT-PACKARD Ltd
King Street Lane
GB-Winnersh, Wokingham
Berkshire RG11 5AR
Telephone: (0734) 784 774

International Service Information

Not all Hewlett-Packard service centers offer service for all models of HP computer products. However, if you bought your product from an authorized Hewlett-Packard dealer, you can be sure that service is available in the country where you bought it.

If you happen to be outside of the country where you bought your unit, you can contact the local Hewlett-Packard service center to see if service is available for it. If service is unavailable, please ship to the address listed under Obtaining Repair Service in the United States. A list of service centers for other countries can be obtained by writing to that address.

All shipping, reimporation arrangements, and customs costs are your responsibility.

Service Repair Charge

There is a standard repair charge for out-of-warranty repairs. The repair charges include all labor and materials. In the United States, the full charge is subject to the customer's local sales tax. In European countries, the full charge is subject to Value Added Tax (VAT) and similar taxes wherever applicable. All such taxes will appear as separate items on invoiced amounts.

Computer products damaged by accident or misuse are not covered by the fixed repair charges. In these situations, repair charges will be individually determined based on time and materials.

Service Warranty

Any out-of-warranty repairs are warranted against defects in materials and workmanship for a period of 90 days from date of service.

Shipping Instructions

Should your unit require service, return it with the following items:

- A completed Service Card, including a description of the problem.
- A sales receipt or other documentary proof of purchase date if the one-year warranty has not expired.

The product, the Service Card, a brief description of the problem, and (if required) the proof of purchase date should be packaged in adequate protective packaging to prevent in-transit damage. Such damage is not covered by the one-year limited warranty; Hewlett-Packard suggests that you insure the shipment to the service center. The packaged unit should be shipped to the nearest Hewlett-Packard designated collection point or service center. Contact your dealer for assistance. (If you are not in the country where you originally purchased the unit, refer to International Service Information above.)

Whether the unit is under warranty or not, it is your responsibility to pay shipping charges for delivery to the Hewlett-Packard service center.

After warranty repairs are completed, the service center returns the unit with postage prepaid. On out-of-warranty repairs in the United States and some other countries, the unit is returned C.O.D. (covering shipping costs and the service charge).

Further Information

Service contracts are available. For information about service contracts, please contact the Calculator Service Center in Corvallis, Oregon.

Calculator product circuitry and design are proprietary to Hewlett-Packard. Service manuals are not available to customers.

Potential for Radio/Televison Interference (for U.S.A. Only)

The HP-75 generates and uses radio frequency energy and if not installed and used properly—that is, in strict accordance with the instructions in this manual—may cause interference to radio and television reception. It has been tested and found to comply with the limits for a Class B computing device in accordance with the specifications in Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference in a residential installation. However, there is no guarantee that interference will not occur in a particular installation. In the unlikely event that your HP-75 does cause interference to radio or television reception (which can be determined by removing all power to the HP-75 and then resetting it), you are encouraged to try to correct the interference by one or more of the following measures:

- Reorient the receiving antenna.
- Relocate the computer with respect to the receiver.
- Move the computer away from the receiver.
- Plug the computer and ac adapter/recharger into a different ac outlet so that the computer and the receiver are on different branch circuits.

If necessary, you should consult your dealer or an experienced radio/television technician for additional suggestions. You may find the following booklet, prepared by the Federal Communications Commission, helpful: *How to Identify and Resolve Radio-TV Interference Problems*. This booklet is available from the U.S. Government Printing Office, Washington, D.C. 20402, Stock Number 004-000-00345-4.

When You Need Help

Technical Assistance. For technical assistance with this product,

call:	(503) 757-2004 8 a.m. to 3 p.m. Pacific time	or write to:	Hewlett-Packard Co. Portable Computer Division Calculator Technical Support 1000 N.E. Circle Blvd. Corvallis, OR 97330

Product Information. For information about Hewlett-Packard products and prices, contact your local Hewlett-Packard dealer. For the name of the dealer nearest you, or to order free literature about Hewlett-Packard products,

call toll-free:	(800) FOR-HPPC (800) 367-4772	or write to:	Hewlett-Packard Co. Personal Computer Group PCG Telemarketing 10520 Ridgeview Court Cupertino, CA 95014

Appendix C

Keyboard Operations

Contents

Key Functions 284
Typewriter Keys 284
Editing Keys 285
System Keys 285
Keystroke Combinations 286
System Keystrokes 286
Editing Keystrokes 286
Display Character Keystroke 287
Escape Keystroke 287
Display Device Keystrokes 287

Key Functions

Typewriter Keys

[A] through [Z] — Letters.

[0] through [9] — Digits.

[ ] — Space.

[.] — Period. Used as a decimal point in numbers and as the final character in abbreviations.

[+] [-] [*] [/] [^] [\] ([CTL] [/]) — Arithmetic symbols: addition (plus), subtraction (minus), multiplication (asterisk), division (slash), exponentiation (circumflex), and integer division (back slash).

C

[,] — Comma. Used to separate items in commands, statements, and functions.

[(] [)] — Parentheses. Used to key in numeric expressions.

[!] — Exclamation mark. Used for end-of-line comments in program statements and for appointment notes.

["] ['] — Double and single quotation marks. Used to enclose filenames and other literal strings.

[#] — Number sign. Used to specify file numbers of BASIC files in `ASSIGN #`, `PRINT #`, `RESTORE #`, and `READ #` statements and to assign timer numbers in `ON TIMER #` and `OFF TIMER #` statements. Also used for inequality in relational tests.

[$] — Dollar sign. Used to specify string variables and string functions.

[&] — Ampersand. Used to concatenate, or join, string expressions.

[[] []] — Opening and closing brackets. Used to dimension string variables and to specify substrings.

[@] Commercial at. Used to form multistatement program lines.

[;] Semicolon. Used to separate items in PRINT, DISP, INPUT, PRINT #, and READ # statements.

[=] Equals. Used to assign variable values and to test for equality.

[<] Less-than. Used in relational tests.

[>] Greater-than. The BASIC prompt for programs and command appointments. Also used in relational tests.

[:] Colon. The text-editing prompt. Also used to delimit device codes.

[%] Percent sign.

[?] Question mark. The default prompt for the INPUT statement.

Editing Keys

[SHIFT] Shift. Causes keys to perform their shifted functions. In uppercase mode, causes the letter keys to display lowercase letters.

[CTL] Control. Used to generate special display characters and to form a variety of keystroke combinations.

[LOCK] Lock. Used when locking the keyboard in uppercase letters or when enabling the numeric keypad.

[←][→] Left-arrow and right-arrow. Move the cursor across the display. If necessary, cause the display to scroll.

[↑][↓] Up-arrow and down-arrow. Move the display up and down through BASIC, text, and appointment files and through the system and mass storage catalogs.

[I/R] Insert/Replace. Exchanges the replace cursor (▮) with the insert cursor (◢).

[CLR] Clear. Clears the display.

[BACK] Backspace. Backspaces the cursor and erases the character there.

[DEL] Delete. Deletes a character and left-shifts the trailing characters in the display line.

[TAB] Tab. Moves the cursor across TIME and APPT mode display fields.

System Keys

The system keys and the [↑] and [↓] keys send a carriage-return/line-feed to the display and DISPLAY IS devices before performing their designated functions.

[ATTN] Attention.Turns on the HP-75; interrupts programs, listings, auto line-numbering, card reader operations, and HP-IL operations. Acknowledges due appointments.

[TIME] Time. Switches the HP-75 to TIME mode.

[APPT] Appointment. Switches the HP-75 to APPT mode. Causes unacknowledged appointments to be displayed.

[EDIT] Edit. Switches the HP-75 to EDIT mode.

[FET] Fetch. In EDIT mode, functions as a typing aid for FETCH command.

[RTN] Return. Causes an expression, statement, or command in the display to be evaluated, stored, or executed.

[RUN] Run. Runs the current BASIC program in EDIT mode. Processes due appointments in APPT mode.

Keystroke Combinations

System Keystrokes

[SHIFT] [CTL] [CLR] Causes the HP-75 to perform a system reset (refer to section 1, page 13).*

[SHIFT] [ATTN] Causes the HP-75 to turn off the display and to disable all keys except [ATTN].

[SHIFT] [LOCK] Locks the HP-75 in uppercase. Pressing [LOCK] again restores the unshifted keys.

[CTL] [LOCK] Enables the numeric keypad. Pressing [LOCK] again restores the normal keyboard.

[SHIFT] [FET] Displays the most recent ERROR, WARNING, card reader, trace, or HP-IL message for as long as the [FET] key is pressed. The message is cleared after [CLR], [ATTN], [TIME], [APPT], [EDIT], [FET], [↑], [↓], [RTN], or [RUN] is pressed.

[SHIFT] [RUN] Single-steps through a program, beginning from the current line.

Editing Keystrokes

[SHIFT] [↑] Moves the file pointer to the first line of the current file, to the first entry in a CAT ALL or mass storage catalog listing, or to the first appointment in APPT mode.

[SHIFT] [↓] Moves the file pointer to the last line of the current file, to the last entry in a CAT ALL or mass storage catalog listing, or to the last appointment in APPT mode.

[CTL] [FET] In EDIT mode, brings the last entry to the display and makes it available for editing. The entry is displayed intact, so long as no other keys have been pressed since the last [RTN]. The following commands use the input buffer, causing your last entry to be lost: FETCH, FETCH KEY, LIST, LOCK, PLIST, and TRANSFORM.

[SHIFT] [DEL] In EDIT and TIME modes, deletes from the cursor to the end-of-line. In APPT mode, deletes the currently displayed appointment or replaces it with an edited appointment.

[SHIFT] [TAB] In APPT and TIME modes, shifts the cursor leftwards across the display fields, the opposite direction of [TAB] alone.

[SHIFT] [←] Shifts the cursor to the beginning of the display line.

---

* [SHIFT] [CTL] [M], [I], [J], [8], and [I/R] also cause a system reset if held for one second or longer.

[SHIFT] [→] Shifts the cursor to the end of the display line.

[CTL] [←] Left-shifts the cursor 32 positions or to the beginning of the display line, whichever distance is shorter.

[CTL] [→] Right-shifts the cursor 32 positions or to the end of the display line, whichever distance is shorter.

Display Character Keystroke

[SHIFT] [I/R] Causes the next key or keystroke combination to display the character associated with it regardless of key redefinitions.

Escape Keystroke

[CTL] [BACK] Generates decimal code 27 (ESC) as the initial character of escape code sequences.

Display Device Keystrokes

[CTL] [↑] Sends an ESC T (roll up one line) to DISPLAY IS devices (ignored by the HP-75 display).

[CTL] [↓] Sends an ESC S (roll down one line) to DISPLAY IS devices (ignored by the HP-75 display).

Appendix D

Reference Tables

Contents

Character Set 288
System Memory Requirements 292
Global and Local Declarations 293
Display Escape Codes 294
Machine Defaults 295
Abbreviations 296

Character Set

The CHR$ function returns the display character of a given decimal code, 0 through 255. Arguments are rounded to integer values and converted to the proper range (modulo 256). An underlined display character has a decimal code 128 larger than its non-underlined equivalent.

The NUM function returns the decimal code of the first character of a given character string.

Examples: Press [RTN] after each.

```
>chr$(65), chr$(65+128)
```

```
A                A
```

```
>num('a'), chr$(num('a')+128)
```

```
97               a
```

Characters whose decimal codes are 32 through 126 are standard printable characters as defined by the American Standard Code for Information Interchange (ASCII). Refer to the HP-IL peripheral owner's manuals to determine the response of individual peripherals to decimal codes 0 through 31 and 127 through 255. To avoid unexpected device behavior during program listings, use the CHR$() function to represent special characters (e.g., the form feed, or FF, character) rather than the display character itself (e.g., use CHR$(12) instead of μ).

D

An asterisk (*) indicates that [SHIFT] [I/R] must first be pressed in order to display the character associated with the key or keystroke combination. Where no keystroke sequence is shown, the display character can only be displayed by using the CHR$ function.

Decimal Code	Display Character	Keystrokes
0	△	[CTL] [space bar]
1	▫	[CTL] [A]
2	⊠	[CTL] [B]
3	←	[CTL] [C]
4	α	[CTL] [D]

Decimal Code	Display Character	Keystrokes
128	a	* [ATTN]
129	b	* [TIME]
130	c	* [APPT]
131	d	* [EDIT]
132	e	* [↑]

Decimal Code	Display Character	Keystrokes
5	⊞	CTL E
6	Γ	CTL F
7	∧	CTL G
8	BS	CTL H
9	σ	CTL I
10	LF	CTL J
11	∩	CTL K
12	μ	CTL L
13	CR	CTL M
14	⊤	CTL N
15	⋕	CTL O
16	Θ	CTL P
17	Ω	CTL Q
18	δ	CTL R
19	∈	CTL S
20	π	CTL T
21	Ä	CTL U
22	ä	CTL V
23	Ö	CTL W
24	ö	CTL X
25	Ü	CTL Y
26	ü	CTL Z
27	ESC	CTL BACK
28	Σ	CTL +
29	≠	CTL =
30	£	CTL ;
31	▒	CTL 8
32		space bar
33	!	SHIFT 1
34	"	SHIFT 2
35	#	SHIFT 3
36	⋕	SHIFT 4
37	%	SHIFT 5
38	&	SHIFT 6
39	'	SHIFT 7
40	(	SHIFT 8
41	)	SHIFT 9
42	*	*
43	+	+
44	,	,
45	–	-
46	.	.

Decimal Code	Display Character	Keystrokes
133	⊞	* ↓
134	Γ	* ←
135	∧	* →
136	∼	* I/R
137	σ	* FET
138	⊢	* DEL
139	∩	* CLR
140	μ	* LOCK
141	▒	* RUN
142	⊤	* TAB
143	⋕	
144	Θ	
145	Ω	Ω
146	δ	
147	∈	
148	π	
149	Ä	
150	ä	
151	Ö	
152	ö	
153	Ü	
154	ü	
155	⊧	
156	Σ	
157	≠	
158	£	
159	▒	
160		* SHIFT ATTN
161	!	* SHIFT TIME
162	"	* SHIFT APPT
163	#	* SHIFT EDIT
164	⋕	* SHIFT ↑
165	%	* SHIFT ↓
166	&	* SHIFT ←
167	'	* SHIFT →
168	(	* SHIFT I/R
169	)	* SHIFT FET
170	*	* SHIFT DEL
171	+	* SHIFT CLR
172	,	* SHIFT LOCK
173	=	* SHIFT RUN
174	⊥	* SHIFT TAB

Decimal Code	Display Character	Keystrokes
47	/	/
48	0	0
49	1	1
50	2	2
51	3	3
52	4	4
53	5	5
54	6	6
55	7	7
56	8	8
57	9	9
58	:	SHIFT ;
59	;	;
60	<	SHIFT ,
61	=	=
62	>	SHIFT .
63	?	SHIFT /
64	@	SHIFT +
65	A	SHIFT A
66	B	SHIFT B
67	C	SHIFT C
68	D	SHIFT D
69	E	SHIFT E
70	F	SHIFT F
71	G	SHIFT G
72	H	SHIFT H
73	I	SHIFT I
74	J	SHIFT J
75	K	SHIFT K
76	L	SHIFT L
77	M	SHIFT M
78	N	SHIFT N
79	O	SHIFT O
80	P	SHIFT P
81	Q	SHIFT Q
82	R	SHIFT R
83	S	SHIFT S
84	T	SHIFT T
85	U	SHIFT U
86	V	SHIFT V
87	W	SHIFT W
88	X	SHIFT X

Decimal Code	Display Character	Keystrokes
175	_/	
176	_0	* CTL 0
177	_1	* CTL 1
178	_2	* CTL 2
179	_3	* CTL 3
180	_4	* CTL 4
181	_5	* CTL 5
182	_6	* CTL 6
183	_7	
184	_8	
185	_9	
186	_:	
187	_;	
188	_<	
189	_=	
190	_>	
191	_?	
192	_@	* CTL ATTN
193	_A	* CTL TIME
194	_B	* CTL APPT
195	_C	* CTL EDIT
196	_D	* CTL ↑
197	_E	* CTL ↓
198	_F	* CTL ←
199	_G	* CTL →
200	_H	* CTL I/R
201	_I	* CTL FET
202	_J	* CTL DEL
203	_K	* CTL CLR
204	_L	* CTL LOCK
205	_M	* CTL RUN
206	_N	* CTL TAB
207	_O	
208	_P	
209	_Q	
210	_R	
211	_S	
212	_T	
213	_U	
214	_V	
215	_W	
216	_X	

Decimal Code	Display Character	Keystrokes
89	Y	SHIFT Y
90	Z	SHIFT Z
91	[	SHIFT 0
92	\	CTL /
93	]	SHIFT -
94	^	SHIFT *
95	_	CTL -
96	`	CTL 7
97	a	A
98	b	B
99	c	C
100	d	D
101	e	E
102	f	F
103	g	G
104	h	H
105	i	I
106	j	J
107	k	K
108	l	L
109	m	M
110	n	N
111	o	O
112	p	P
113	q	Q
114	r	R
115	s	S
116	t	T
117	u	U
118	v	V
119	w	W
120	x	X
121	y	Y
122	z	Z
123	{	CTL ,
124	|	SHIFT =
125	}	CTL .
126	~	CTL *
127	⊢	CTL 9

Decimal Code	Display Character	Keystrokes
217	Y	
218	Z	
219	[	
220	\	
221	]	
222	^	
223	_	
224	`	* SHIFT CTL ATTN
225	a	* SHIFT CTL TIME
226	b	* SHIFT CTL APPT
227	c	* SHIFT CTL EDIT
228	d	* SHIFT CTL ↑
229	e	* SHIFT CTL ↓
230	f	* SHIFT CTL ←
231	g	* SHIFT CTL →
232	h	* SHIFT CTL I/R
233	i	* SHIFT CTL FET
234	j	* SHIFT CTL DEL
235	k	
236	l	* SHIFT CTL LOCK
237	m	* SHIFT CTL RUN
238	n	* SHIFT CTL TAB
239	o	
240	p	
241	q	
242	r	
243	s	
244	t	
245	u	
246	v	
247	w	
248	x	
249	y	
250	z	
251	{	
252	|	
253	}	
254	~	BYE†
255	⊢	* SHIFT CTL CLR

† When decimal code 254 is assigned to a key, pressing the redefined key will execute the BYE command. Using CHR$ to display character 254 will cause the ~ character to be displayed.

System Memory Requirements

To determine the size of an initialized program, first interrupt execution (press [ATTN]) and determine available memory (type `mem` [RTN]). Then deallocate the program by performing a line edit. The amount of additional memory that becomes available plus the size of the deallocated file (as indicated in the catalog entry) is the total size.

Item	Memory Required
Appointments	15 bytes for the appointment file *plus* 7 bytes/appointment *plus* 1 byte/character in the `Note` field *plus* 5 bytes/repeating appointment.
Data items	5 bytes/`DATA` statement *plug* 4 bytes/ integer *plus* 5 bytes/short *plus* 9 bytes/real *plus* 2 bytes/item *plus* 1 byte/character in strings.
Data pointers	15 bytes/data pointer created by an `ASSIGN #` statement.
Files (BASIC and text)	15 bytes/file *plus* 3 bytes/line *plus* 1-3 bytes/keyword (BASIC) or 1 byte/character (text).
Files (LEX and interchange)	Refer to the catalog entry for the file.
HP-IL assignments	7 bytes/device declared in an `ASSIGN IO` command.
Key redefinitions	3 bytes/redefinition *plus* 1 byte/character in the the key definition string *plus* 1 byte for final semicolon.
Mass storage commands (`CAT`, `COPY`, `PURGE`, `RENAME`)	43 bytes/command.
Mass storage devices	105 bytes/device.
`PACK` command	256 bytes *plus* 6 bytes/file.
Plug-in ROMs	Refer to the ROM owner's manual for the number of bytes of system RAM used by the ROM.
Program calls	30 bytes/`CALL` statement *plus* 2 bytes/calling program variable *plus* the number of bytes required by the variables' sizes or precisions.
Timers	63 bytes/timer set by an `ON TIMER` statement *plus* one or more bytes per timer instruction.
`TRANSFORM` command	A maximum of 255 bytes while the transformation is occurring.
Variables (both calculator variables and initialized program variables)	
Simple numeric variables	
`REAL`	12 bytes/variable
`SHORT`	8 bytes/variable
`INTEGER`	7 bytes/variable
Numeric array variables	10 bytes/array variable *plus*
`REAL`	8 bytes/element
`SHORT`	4 bytes/element
`INTEGER`	3 bytes/element
String variables	8 bytes/variable *plus* dim length.

Global and Local Declarations

A *global* declaration is an HP-75 setting that remains in effect until:

- The setting is changed by a new declaration, executed either from the keyboard or from a running program.
- The HP-75 is reset.

A *local* declaration affects only keyboard calculations or the program in which the declaration occurs.

Local	Global
DATA	ALARM ON, ALARM OFF
DEF FN, END DEF	ASSIGN IO, OFF IO, RESTORE IO
DIM	ASSIGN #
IMAGE	BEEP OFF, BEEP ON
LET	DEF KEY
ON ERROR, OFF ERROR	DEFAULT OFF, DEFAULT ON
ON TIMER #	DELAY
OPTION BASE	DISPLAY IS, PRINTER IS
REAL, SHORT, INTEGER	ENDLINE
	LOCK
	MARGIN
	OFF TIMER #
	OPTION ANGLE RADIANS
	OPTION ANGLE DEGREES
	STANDBY OFF, STANDBY ON
	TRACE FLOW, TRACE VARS, TRACE OFF
	WIDTH, PWIDTH

Display Escape Codes

ESC represents the escape character, decimal code 27. Displaying [ESC]—that is, [CTL][BACK]—or displaying `CHR$(27)` sends an escape character to the HP-75 display and other `DISPLAY IS` devices. Printing `CHR$` sends an escape character to `PRINTER IS` devices.

The HP-75 display and most HP-IL devices have special responses to predefined *escape codes*, that is, to display or print instructions which contain sequences formed by the escape character and the characters immediately following. A given escape code may be ignored by one device while interpreted by another device as a machine instruction. For example, an ESC `C` is ignored by the HP 82162A Thermal Printer but causes the HP 82163 Video Interface to move the cursor one position to the right. An ESC `C` may be sent to the HP-75 display and other display devices using `DISP CHR$(27)&'C` , for example.

The HP-75 display responds to 12 escape codes. Other escape sequences used in `DISP` or `PRINT` statements are ignored by the display.

The 32 display window positions are identified as 0 through 31.

Escape Code	Instruction	Description
ESC `C`	Cursor Right*	Moves the cursor one position to the right.
ESC `D`	Cursor Left*	Moves the cursor one position to the left.
ESC `E`	Clear All*	Moves the cursor to column 0 and clears the display.
ESC `G`	Cursor Return	Moves the cursor to column 0.
ESC `H`	Cursor Home*	Moves the cursor to column 0.
ESC `J`	Clear From Cursor*	Clears the display from cursor location.
ESC `K`	Clear to End of Line	Clears the display from the cursor location.
ESC `O`	Delete Character With Wraparound	Deletes the character at the cursor location and left-shifts all trailing characters with wraparound.
ESC `P`	Delete Character	Deletes the character at the cursor location and left-shifts all trailing characters.
ESC `<`	Cursor Off*	Turns off the cursor.
ESC `>`	Cursor On*	Turns on the cursor.
ESC `%` *c r*	Cursor to Address*	Moves the cursor to the display position (0 through 31) specified by the decimal code of the first display character modulo 32. The decimal code of the second character is used by the video interface as a row parameter, but is ignored by the HP-75. For example, `DISP CHR$(27);'%';CHR$(16);CHR$(8);'here'` positions the cursor to column 16 (and row 8 of the video interface) and displays `here`, beginning from that location.

An asterisk (*) indicates that the HP 82163 Video Interface responds similarly when declared a `DISPLAY IS` device. Refer also to the owner's manual for the video interface.

Type a semicolon after an escape code to suppress the carriage-return/line-feed that normally terminates a DISP or PRINT statement.

In addition, pressing [CTL][↑] sends an ESC T (roll up one line) to DISPLAY IS devices and pressing [CTL][↓] sends an ESC S (roll down one line) to DISPLAY IS devices. The HP-75 display window ignores these characters.

Machine Defaults

Condition	After a machine reset or after the battery pack is first installed.	After turning on following a timeout period.
ALARM	ON	Previous setting.
ASSIGN IO	No HP-IL assignments.	Previous HP-IL assignments.
ASSIGN #	No file number assignments.	Previous file number assignments.
BEEP	ON	Previous setting.
[CTL][FET]	Set-time template.	Blank line.
DEFAULT	ON	Previous setting.
DEF KEY	Identity assignments	Previous key definitions.
DELAY	1 second.	Previous setting.
DISPLAY IS	* (That is, the HP-75 display.)	Previous display devices.
ENDLINE	CR/LF	Previous setting.
ERRL	0	Previous error line.
ERRN	0	Previous error number.
EXACT	EXACT marks cleared.	EXACT marks unchanged.
FETCH	Line 0 of workfile.	Previously pending line.
Files	BASIC workfile (0 bytes, time and date of clock setting.)	Previous BASIC files, text files, appointment files, keys files, LEX files, LIF1 files.
Keyboard	Unshifted.	Previous keyboard mode.
LOCK	No password.	Previous password.
MARGIN	91	Previous setting.
Mode	TIME	EDIT
PRINTER IS	* (The HP-75 display.)	Previous printer devices.
PWIDTH	32 columns.	Previous setting.
RES	0	Last numeric result.
RND	.529199358633	Next number in sequence.
[SHIFT][FET]	Blank display.	Blank display.
STANDBY	OFF	Previous setting.
STATS template	MDY, ~, AM, YEAR	Previous settings.
TIME display	Set-time values.	Current time.
Trigonometric mode.	OPTION ANGLE RADIANS	Previous setting.
Variables:		
calculator	None.	Previous assignments.
program	None.	Previous assignments if allocated program.
WIDTH	32 columns.	Previous setting.

Abbreviations

The following are the shortest distinct abbreviations recognized by the HP-75. Longer abbreviations are allowed, providing that the period doesn't match the final character of the keyword. Abbreviations may not contain embedded blanks.

The HP-75 supplies the complete spelling of keywords when listing and fetching lines.

Keyword	Abbreviation
ACOS()	ac.()
ALARM OFF	al.off
ALARM ON	al.on
ANGLE()	ang.()
ASSIGN #	as.#
ASSIGN IO	as..
AUTO	a.
BEEP	be.
BEEP OFF	be.off
BEEP ON	be.on
BYE	b.
CAT$()	c.()
CAT ALL	c..
CEIL()	ce.()
CHR$()	ch.()
CLEAR *device code*	cl.':*dc*'
CLEAR LOOP	cl.l.
CLEAR VARS	cl..
COPY	co.
DATA	da.
DEFAULT OFF	defa.off
DEFAULT ON	defa.on
DEF KEY	d..
DELAY	d.
DELETE	dele.
DISP	di.
DISPLAY IS	di..
DISP USING	di.us.
EDIT	e.
EDIT BASIC	e.ba.
EDIT TEXT	e.t.
ELSE	el.
END DEF	end d.
ENDLINE	en.
ERRN	er.
FETCH	f.
FETCH KEY	f..
FLOOR()	fl.()
GOSUB	gos.
GOTO	g.

Keyword	Abbreviation
IMAGE	im.
INITIALIZE	ini.
INPUT	i.
INTEGER	int.
KEY$	k.
LIST	l.
LIST IO	l..
LET FN	le..
LOG10()	lo.()
MARGIN	ma.
MERGE	m.
NAME	n.
NEXT	ne.
OFF ERROR	of.e.
OFF TIMER #	of.t.#
ON ERROR	o..
ON TIMER #	o.t.#
OPTION ANGLE DEGREES	op.a.d.
OPTION ANGLE RADIANS	op.a.r.
OPTION BASE	op..
PACK	pa.
PLIST	pl.
POP	p.
PRINT	pri.
PRINT #	pri.#
PRINTER IS	pri..
PRINT USING	pri.us.
PROTECT	pr.
PURGE	pu.
PWIDTH	pw.
RANDOMIZE	ra.
READ #	r.#
REAL	re.
RENAME	ren.
RENUMBER	renu.
RESTORE	res.
RESTORE #	res.#
RESTORE IO	res..
RETURN	ret.
RUN	r.

Keyword	Abbreviation
SHORT	sh.
STANDBY OFF	s.off
STANDBY ON	s.on
THEN	th.
TIME$	ti.
TRACE FLOW	t.
TRACE OFF	tr.o.
TRACE VARS	tr.v.
TRANSFORM	tr.

Keyword	Abbreviation
UNPROTECT	u.
UPRC$()	up.()
VER$	v.
WAIT	wa.
WIDTH	w.

Appendix E

Error Conditions

Contents

Occurrences 298
Mathematical Warnings and Errors (1–13) 299
System Errors (14–18) 300
Card Reader Warnings (19–26) 300
Program Errors (27–54) 301
HP-IL Errors (55–61) 303
File and Device Errors (62–69) 303
TIME Mode Warning (70) 304
APPT Mode Errors (71–77) 305
Syntax Errors (78–91) 305
Mass Memory Errors (92–97) 306
Alphabetical Listing 307

E

Occurrences

When an error occurs, the HP-75 sounds the beep, lights the **ERROR** annunciator, and displays an ERROR or WARNING message according to the current DELAY rate. If the error occurs during program execution, the line number of the statement that caused the error is also displayed. Pressing [SHIFT] [FET] displays the message again for as long as the [FET] key is held down. Typing errn [RTN] returns the identification number of the error or warning. Pressing [CLR], [ATTN], [TIME], [APPT], [EDIT], [FET], [↑], [↓], [RTN], or [RUN] turns off the **ERROR** annunciator and clears the message.

A WARNING condition causes the HP-75 to supply default values and allows execution to continue (unless ON ERROR has been declared). An ERROR condition interrupts program execution at the statement that caused the error. If the interrupted program is also the current file, then the file pointer will be set to the line in which the error occurred; pressing [FET] and then [RTN] fetches the line to the display. The program remains initialized until you edit any line or run another program.

Mathematical Warnings and Errors		
Number	**Message and Condition**	**Default Values (with DEFAULT ON)**
1	num too small Number lies between ±EPS.	0
2	num too large • Larger than ±INF. • Larger than SHORT precision allows. • Larger than INTEGER precision allows.	±9.9999999999E499 ±9.9999E99 ±99999
3	COT or CSC inf COT or CSC of $n \times 180°$; n = integer.	9.9999999999E499
4	TAN or SEC inf TAN or SEC of $n \times 90°$; n = odd integer.	9.9999999999E499
5	0^neg Zero raised to a negative power.	9.9999999999E499
6	0^0 Zero raised to zero power.	1
7	no value • The value of a string variable is referenced before the variable has been assigned a value. • The value of a simple numeric variable or a numeric array element is referenced before the variable has been assigned a value. Although a default value will be supplied to the program with DEFAULT ON, the variable value will remain undefined.	'' 0
8	/zero Division by zero.	±9.9999999999E499
(The remaining mathematical errors do not default.)		
9	neg^non-integer Negative value raised to a non-integer power.	
10	SQR(neg number) Square root of a negative number.	
11	arg out of range Argument too large or too small.	
12	LOG(0) Logarithm of zero.	
13	LOG(neg number) Logarithm of a negative number.	

System Errors (14 through 18)	
Number	**Message and Condition**
14	`Low batteries` Replace battery pack or plug in the ac adapter/recharger.
15	`system error` The HP-75 may require a reset.
16	`not enough memory` • Not enough available memory exists to copy a card or tape file to memory. • A file or program requires too much memory. Possible solutions: • Delete one or more lines from the current file. • Execute `CLEAR VARS` to clear calculator variables. • Lessen the precisions or reduce the dimensions of program variables. • Purge one or more files from memory. • Execute `ASSIGN # TO *`, specifying any unused data file pointers. • Execute `CATALL` then press ↑ or ↓ until you see a file that can be purged, then press EDIT and execute `PURGE .` • Execute `ASSIGN IO *` to clear HP-IL loop.
17	`RAM is invalid` • Indicates a defective circuit. Unit may require service. • Indicates memory lost.
18	`ROM missing` Missing a necessary plug-in ROM or LEX file.

Card Reader Warnings (19 through 26)	
Number	**Message and Condition**
19	`write protected` Attempting to copy to a write-protected card. Execute `UNPROTECT` first.
20	`not this file` A track does not belong to the same file that is being copied to memory.
21	`verify failed` The same track requires two more passes through the card reader, once to write to the track and once more to verify. Clean the card first.
22	`unknown card` The information on the track is not recognized by the HP-75.
23	`bad read/write` A failure to read a track or write to the track properly. Clean the card and try again.
24	`pulled too fast` Pull the card through the card reader more slowly.
25	`pulled too slow` Pull the card through the card reader faster.
26	`wrong name` The file specifier in a `COPY` command doesn't match the name of the card file.

Program Errors (27 through 54)	
Number	**Message and Condition**
27	`invalid subscript` • An array subscript is out of bounds. • A string variable substring is out-of-bounds.
28	`record overflow` A random `PRINT #` statement tried to force the data pointer past the end of line.
29	`ON ERROR overflow` An `ON ERROR` routine was requested to handle error 49—`GOSUB overflow`. Rather than create another subroutine and add to the problem, the HP-75 halts execution and displays this message.
30	`OPTION BASE` • Out of range `OPTION BASE` specified. • `OPTION BASE` statement appears after an array variable reference. • Multiple `OPTION BASE` statements in one program.
31	`CONT before RUN` `CONT` is executed or [SHIFT] [RUN] is pressed before a program has been initialized or after a program has been deallocated.
32	`missing line` • Missing a line that is referenced in a `GOTO` or `GOSUB` statement. • Missing a line that is referenced is a `DISP USING` or `PRINT USING` statement.
33	`data type` A `READ` or `READ #` statement tried to read string information into a numeric variable.
34	`no data` • A `READ` or serial `READ #` statement tried to read past the end of file. • A `RESTORE` or `RESTORE #` statement was executed when there was no data in the file. • A `RESTORE`, `RESTORE #`, random `READ #` statement, or random `PRINT #` statement specified an existing but non-`DATA` line in a BASIC file. • A `RESTORE #` or `READ #` statement specified a nonexistent line in a text file.
35	`DIM exist var` Declaring the precision or dimension of a simple numeric variable, string variable, or numeric array variable that has appeared previously in the program.
36	`Invalid DIM` Dimensioning an array improperly after an `OPTION BASE` declaration.
37	`duplicate FN` The same function name appears in two or more `DEF FN` statements.
38	`no END DEF` A multiple-line user-defined function is missing the `END DEF` statement.
39	`FN missing` Attempting to branch into the middle of a multiple-line user-defined function.

Program Errors (27 through 54) (*continued*)

Number	Message and Condition
40	FN parameter • Mismatch occurred between the formal parameter list in the function definition and the actual parameter list in the main program. • Attempting to use a user-defined function that isn't defined.
41	FN calls itself A user-defined function is defined in terms of itself.
42	string too long • Attempting to assign too many characters to a string variable. • Printing a string longer than 251 characters to a BASIC file or longer than 255 characters to a text file. Causes a warning and truncation of the string.
43*	numeric input • Attempting to supply characters in response to an INPUT statement. • Pressing [RTN] before enough values have been supplied for all the numeric variables in the input List.
44*	too many inputs Too many items entered in response to an INPUT statement.
45	missing ASSIGN# Executing a PRINT #, READ #, or RESTORE # statement without first assigning a file number.
46	missing NEXT Missing the NEXT statement of a FOR-NEXT loop.
47	no matching FOR A NEXT statement is encountered with no corresponding FOR statement. May be caused by an incorrectly nested loop.
48	FOR overflow FOR nesting exceeds 255 levels.
49	GOSUB overflow GOSUB nesting exceeds 255 levels.
50	RETURN w/o GOSUB A RETURN statement was encountered with no pending subroutine condition.
51	PRINT# to runfile A program referenced itself in a PRINT # statement.
52	invalid IMAGE At least one of the specifiers in an IMAGE, DISP USING, or PRINT USING statement is improper. May be caused by an invalid character in the image string.
53	invalid USING One of the specifiers in an IMAGE, DISP USING, or PRINT USING statement cannot represent an item to be displayed or printed.
54	invalid TAB TAB argument rounds to a value less than 1. A warning condition occurs and a value of 1 is supplied.

* An INPUT error will cause the HP-75 to prompt again for input. If an ON ERROR declaration is in effect at the time of an INPUT error, then as many values as possible will be assigned to the INPUT variables and the ON ERROR instruction will be performed.

HP-IL Errors (55 through 61)	
Number	**Message and Condition**
55	`ASSIGN IO needed` Executing `DISPLAY IS`, `PRINTER IS`, `LIST IO`, `OFF IO`, or `RESTORE IO` without first assigning the devices on the loop.
56	`no loop response` The HP-IL loop is connected, but no devices responded to the `ASSIGN IO` command.
57	`bad transmission` • Hardware error. • Pressing [ATTN] during a loop transmission.
58	`loop timeout` • An intentional or accidental disconnection occurred or power was lost to a peripheral during a loop transmission. • During `STANDBY OFF`, a device took longer than 10 seconds to process a single HP-IL instruction. Make sure the devices are properly connected and powered and then execute `RESTORE IO`. If the disconnection was intentional, connect an HP-IL cable between in and out connectors on the HP-75 and execute `OFF IO`. Execute `STANDBY ON` if a device needs longer than 10 seconds to respond to a command.
59	`too many names` • More names than devices in an `ASSIGN IO` command. Causes a warning; as many devices as exist will be assigned. • More device names in memory than connected devices in the loop were found after a `RESTORE IO` command. Causes an error; no devices are assigned. Connect the original devices and reexecute `RESTORE IO`, or execute a new `ASSIGN IO`.
60	`RESTORE IO needed` The HP-75 tried to issue an HP-IL instruction after an `OFF IO` command.
61	`>31 devices` More than 31 devices, including the HP-75, have been connected in the HP-IL loop.

File and Device Errors (62 through 69)	
Number	**Message and Condition**
62	`file not found` The specified file doesn't exist in memory or on mass storage medium.
63	`invalid filespec` • Invalid name for a file in memory. Restricted to one to eight characters—first character a letter or period, remaining characters letters or digits.. • Invalid name for a mass storage or card file. Restricted to one to eight letters or digits; the first character must be a letter (periods aren't allowed). • Invalid HP-IL device code. Restricted to one or two letters or one letter and one digit. • A `TRANSFORM INTO LIF1` command was executed from a `workfile`. • A display device code is used in a mass storage command.
64	`duplicate name` • Duplicate filename. A file already exists in memory or on a mass storage medium by that name. • Duplicate device code in an `ASSIGN IO` declaration. If `ASSIGN IO` devices are being assigned one by one from the keyboard, causes a warning; the HP-75 prompts for another device code.

File and device Errors (62 through 69) (*continued*)	
Number	**Message and Condition**
65	`access restricted` • Attempting to run an appointment, text, LEX, or LIF1 file. • Attempting to EDIT, LIST, COPY, or RENAME a private BASIC file. • Attempting to PRINT # to or READ # from a private BASIC file, a LEX file, or a LIF1 file. • Attempting to PRINT # a numeric value to a text file.
66	`invalid password` • The password for a mass storage or card file must be correctly specified in order to copy the file. • A password is specified when copying a LIF1 file or a file not created by the HP-75 to or from a mass storage medium. Causes a warning; the copy is allowed, but no password is attached to the destination file.
67	`line too long` • A line listed or fetched from a file contains more than three display windows of information. Causes a warning, and the first 94 characters of the line are displayed. If [RTN] is pressed, the fetched line will be truncated at the 94th character, if possible; otherwise, a syntax error will be reported. • Fetching a key definition that exceeds 80 characters. Only the first 80 characters of the definition will be displayed. • Attempting to transform a LIF1 file with excessively long lines into text or BASIC—produces this as a warning and inserts ! ? immediately after the line number.
68	`wrong file type` • Attempting to rename a non-text file to keys. • Attempting to rename a non-appointment file to appt. • Attempting to merge one file into a file of a different type. • Attempting to copy an unknown (?) mass storage file to memory. • Attempting to copy a text, appointment, LEX, or LIF1 file to a private file; causes a warning and allows a non-private copy to be made.
69	`workfile name?` Attempting to edit another file while editing a non-empty, unnamed workfile. NAME, RENAME, or PURGE the workfile before executing the EDIT command or pressing the [EDIT] key during a CAT ALL.

Time Mode Warning (70)	
Number	**Message and Condition**
70	`time adjust bad` Attempting to adjust the clock speed by more than ±10%. Causes a warning; the adjustment factor is left at its current setting, and a new adjustment period is begun.

APPT Mode Errors (71 through 77)	
Number	**Message and Condition**
71	`duplicate APPT` Attempting to enter a duplicate appointment. Appointments must differ by at least one character.
72	`day/date mismatch` The day-of-week and the date don't agree.
73	`bad day field` Misspelling the day-of-week.
74	`bad date field` Out-of-range parameter specified for the month, day, or year. May be caused by an illegal character in the `Mo`, `Dy`, or `Yr` fields.
75	`bad time field` Out-of-range parameter specified for the hour or minute. May be caused by an illegal character in the `Hr`, `Mn`, or `AM` fields.
76	`bad rep field` • Invalid Rept (repeating) template caused by an out-of-range parameter or an illegal character. • Pressing a system key while the Rept template is displayed.
77	`bad alarm spec` • Invalid alarm type. Must be a digit, 0 through 9. • Invalid appointment type. Must be `N`, `A`, or `R`.

Syntax Errors (78 through 91)	
Number	**Message and Condition**
78	`syntax` • Incorrect spacing or characters in line. Cursor is positioned to character where error was detected. • A line not understood during a `TRANSFORM INTO BASIC` operation. The error is reported after the entire file has been transformed; unrecognized lines are transformed into program remarks beginning with `! ?`.
79	`; expected` Missing a semicolon between parameters.
80	`) expected` Missing a right parenthesis in an expression.
81	`comma expected` Missing a comma between parameters.
82	`string expected` Unsuccessful attempt to evaluate the expression as a character string.
83	`missing TO` The keyword `TO` with no embedded blanks must be typed in the command.
84	`extra characters` Extra characters appear at the end of the line. May be caused by mistyping an instruction if it is interpreted as a different instruction.

Syntax Errors (78 through 91) (*continued*)	
Number	**Message and Condition**
85	`expr too big` Expression is too big for the HP-75 to evaluate. May be caused by too many nested pairs of parentheses or too many operations in one expression.
86	`illegal context` Statement is not allowed after `THEN`, `ELSE`, `ON TIMER #`, or `ON ERROR`.
87	`bad expression` • A syntax error in an expression; e.g., too many operators between operands. • Attempting to transform a LIF1 file with missing line numbers into text or BASIC. The file remains a LIF1 file.
88	`bad statement` • An incorrect abbreviation. • An incomplete statement. • Attempting to execute a program only statement (e.g., `GOTO`) from the keyboard.
89	`bad parameter` • Using the wrong type of parameter; e.g., using a string argument for a function that takes a numeric argument. • Entering out-of-range parameter, illegal characters, or too many parameters.
90	`bad line number` • Attempting an improper renumbering operation. Causes a warning; default line numbering occurs for the specified portion of the file. • A `PRINT #` statement attempts to create a line number greater than 9999. Causes an error; no printing is done and the data pointer is left at the end of file. • Attempting to specify a starting line number greater than 9999 in an `AUTO` command.
91	`missing parameter` Omitting a necessary parameter from a function, statement, or command.

Mass Memory Errors (92 through 97)	
Number	**Message and Condition**
92	`dev not mass mem` A non-mass storage device was requested to perform a mass storage operation.
93	`mass mem error` The mass storage device is experiencing difficulty, perhaps due to low batteries.
94	`no medium` Load a medium into the mass storage device.
95	`medium full` The medium in the mass storage device has no room for other files. Purge one or more files, pack the medium, or load another medium.
96	`invalid medium` The mass storage device cannot read from or write to the medium. May be caused by an unformatted medium; execute an `INITIALIZE` command for the device. May be caused by a physically damaged or defective medium. May also be caused by dirty recording heads; clean the heads as directed in the peripheral manual.
97	`invalid pack` Interrupting a `PACK` operation. The medium may need to be reformatted with an `INITIALIZE` command.

Alphabetical Listing

Message	Number
access restricted	65
arg out of range	11
ASSIGN IO needed	55
bad alarm spec	77
bad date field	74
bad day field	73
bad expression	87
bad line number	90
bad parameter	89
bad read/write	23
bad rep field	76
bad statement	88
bad time field	75
bad transmission	57
comma expected	81
CONT before RUN	31
COT or CSC inf	3
data type	33
day/date mismatch	72
dev not mass mem	92
DIM exist var	35
duplicate APPT	71
duplicate FN	37
duplicate name	64
expr too big	85
extra characters	84
file not found	62
FN calls itself	41
FN missing	39
FN parameter	40
FOR overflow	48
GOSUB overflow	49
illegal context	86
invalid DIM	36
invalid filespec	63
invalid IMAGE	52
invalid medium	96
invalid pack	97
invalid password	66
invalid subscript	27
invalid TAB	54
invalid USING	53
line too long	67
LOG(neg number)	13
LOG(0)	12
loop timeout	58
low batteries	14
mass mem error	93
medium full	95
missing ASSIGN#	45
missing line	32
missing NEXT	46
missing parameter	91
missing TO	83
neg^non-integer	9
no data	34
no END DEF	38
no loop response	56
no matching FOR	47
no medium	94
no value	7
not enough memory	16
not this file	20
num too large	2
num too small	1
numeric input	43
ON ERROR overflow	29
OPTION BASE	30
PRINT# to runfile	51
pulled too fast	24
pulled too slow	25
RAM is invalid	17
record overflow	28
RESTORE IO needed	60
RETURN w/o GOSUB	50
ROM missing	18
SQR(neg number)	10
string expected	82
string too long	42
syntax	78
system error	15
TAN or SEC inf	4
time adjust bad	70
too many inputs	44
too many names	59
unknown card	22
verify failed	21
workfile name?	69
write protected	19
wrong file type	68
wrong name	26
/zero	8
0^neg	5
0^0	6
>31 devices	61
) expected	80
; expected	79

Appendix F

Owner's Pac Program Listings

Contents

The MONEY Program 308
The PAYATTN Program 310
The NAMELIST Program 315

Listings follow for the three prerecorded programs shipped with your HP-75—the MONEY, PAYATTN, and NAMELIST programs. The programs are referenced in the main manual to demonstrate programming designs and applications.

The MONEY Program

The MONEY program is discussed in section 1 on page 21.

Specifications:

Input: Borrowing or savings option.

Borrowing inputs: Initial amount, number of periods, interest rate per period.

Borrowing output: Periodic payment.

Savings inputs: Initial amount, number of periods, interest rate per period, periodic deposit.

Savings output: Future amount.

Size:

Recorded on four card tracks, 2205 bytes.

Requires approximately 2947 bytes when initialized.

```
1000 ! Money - 04/22/82
1010 DELAY 0 @ WIDTH INF
1020 E$=CHR$(27) @ H$=E$&"E" @ C0$=E$&"<" @ U$=E$&"A" @ O$="+-*/^" @ T=1
```

Program introduction and menu.

```
1030 DISP H$;C0$;E$&"%IC~ M O N E Y ~" @ WAIT T
1040 DISP @ DISP "Choose "&CHR$(226)&"orrow or "&CHR$(243)&"ave money";
1050 INPUT ": "; R$ @ IF NOT LEN(R$) THEN R=0 @ GOTO 1080
1060 R=POS("BS",UPRC$(R$[1,1]))
1070 IF R#0 THEN 1100
1080 DISP @ DISP "Please enter 'b' or 's'." @ WAIT T
1090 DISP FNR$(4); @ GOTO 1040
1100 FOR I=1 TO 5 @ V(I)=0 @ NEXT I @ F=0
1110 OFF ERROR @ M=1 @ N=3 @ RESTORE 1160 @ DISP H$;C0$;
1120 IF R=1 THEN DISP "Borrow money ..." ELSE DISP "Save money ..."
1130 WAIT .5
1140 ON ERROR DISP FNR$(2*I-1); @ GOTO 1230
```

Get user inputs.

```
1150 FOR I=M TO N @ READ P$
1160 DATA 'Initial amount: $','Number of periods: ','Int.rate/period: '
1170 DATA 'Periodic deposit: $'
1180 IF F#0 THEN 1190 ELSE V$="" @ GOTO 1230
1190 ON I GOTO 1220,1210,1200,1220
1200 V$=STR$(V(3)) @ V$=V$&" %" @ GOTO 1230
1210 V$=STR$(V(2)) @ GOTO 1230
1220 V$=STR$(V(I)) @ GOSUB 1490
1230 DISP @ DISP P$;
1240 IF F=0 AND I=3 THEN 1250 ELSE 1260
1250 PUT CHR$(136) @ V$=" %"
1260 INPUT "",V$;V$
1270 GOSUB 1550
1280 V(I)=ABS(V(I))
1290 NEXT I @ IF R=2 AND I=4 THEN N,M=4 @ GOTO 1150
1300 F=1 @ ON ERROR GOTO 1440
1310 R1=V(3)/100 @ R2=R1+1 @ R3=R2^(-V(2))
1320 IF R=1 THEN 1360
```

Save money calculations.

```
1330 IF V(3)=0 THEN V(5)=V(1)+V(2)*V(4) ELSE V(5)=(V(1)+V(4)*R2/R1*(1-R3))/R3
1340 I=5 @ GOSUB 1490
1350 L$="Future amount: $" @ GOSUB 1450 @ GOTO 1390
```

Borrow money calculations.

```
1360 IF V(3)=0 THEN V(4)=V(1)/V(2) ELSE V(4)=V(1)*R1/(1-R3)
1370 I=4 @ GOSUB 1490
1380 L$="Periodic payment: $" @ GOSUB 1450
```

Automatic program end.

```
1390 OFF ERROR
1400 ON TIMER # 1,5 GOSUB 1670
1410 ON TIMER # 2,120 DISP FNR$(NUM("I")+2*R);"End of 'Money' program" @ STOP
1420 IF KEY$="" THEN 1420
1430 OFF TIMER # 1 @ OFF TIMER # 2 @ GOTO 1110
```

Invalid entries message.

```
1440 DISP @ DISP "Invalid entries; try again." @ WAIT T @ GOTO 1100
```

Display the output.

```
1450 DISP
1460 DISP @ DISP L$;
1470 IF LEN(V$)>12 THEN DISP USING "d.dddddde" ; V(I) ELSE DISP V$
1480 RETURN
```

Round value for output.

```
1490 V(I)=INT(V(I))+INT(FP(V(I))*100+.5)/100 @ V$=STR$(V(I))
1500 IF V(I)>=1.E12 THEN RETURN
1510 IF V(I)<.005 THEN V(I)=0
1520 P=POS(V$,".") @ IF P=LEN(V$)-2 AND P#0 THEN 1540
1530 IF P=0 THEN V$=V$&".00" ELSE V$=V$&"0"
1540 RETURN
```

Evaluate numeric expression.

```
1550 Y=2
1560 FOR J=1 TO 5
1570 X=POS(V$[Y,LEN(V$)],O$[J,J])+1 @ IF X#1 THEN 1590
1580 NEXT J @ V(I)=VAL(V$) @ RETURN
1590 IF UPRC$(V$[X-1,X-1])="E" THEN Y=X+1 @ GOTO 1560
```

Strobe keyboard for key to continue.

```
1600 U=VAL(V$[1,X-1]) @ V=VAL(V$[X+1,LEN(V$)])
1610 ON J GOTO 1620,1630,1640,1650,1660
1620 V(I)=U+V @ RETURN
1630 V(I)=U-V @ RETURN
1640 V(I)=U*V @ RETURN
1650 V(I)=U/V @ RETURN
1660 V(I)=U^V @ RETURN
1670 DISP @ DISP "(Press any key to continue)";U$;U$;U$;
1680 FOR W=1 TO 300
1690 IF KEY$#"" THEN POP @ GOTO 1430
1700 NEXT W
1710 GOSUB 1460
1720 RETURN
```

Select a row for video display.

```
1730 DEF FNR$(R) = CHR$(27)&"%@"&CHR$(R)&CHR$(27)&"J"
```

The PAYATTN Program

The PAYATTN program is discussed in section 11 on page 170.

Specifications:

Inputs: Number of players, level of difficulty, user's initials.

Outputs: Scrolling introduction, audible tones, cursor positioning, current score, bonus indicator, final score, highest score.

Special Keys: ←, →, space bar.

Size:

Recorded on eight card tracks, 4708 bytes.

Requires approximately 5777 bytes when initialized.

```
1000 ! Pay Attention - 04/22/82
1010 DELAY 0 @ WIDTH INF
1020 INTEGER L9,A9,H,N9,B1,I1,D1,E1,B9,C5,X5,Y5,Z,U,A,L,V,E,R7,R1,L1
1030 INTEGER A(12),B(12),C(2),S(2),W(2)
1040 DIM A$[26],B$[12],T$[26],Z$[26],C$[1],E1$[1],H1$[2],C1$[2],W5$[4]
1050 DIM E5$[4],P1$[1],P2$[1],R1$[2],E$[5],Y$[1]
1060 E1$=CHR$(27) @ H1$=E1$&"E" @ C1$=E1$&"<" @ W5$="["&E1$&"C"&"]"
1070 E5$=" "&E1$&"C"&" " @ P1$,P2$="" @ I$="     "
1080 T2=1.5 @ T3=.2 @ T5=.6 @ T6=0
1090 N9=6 @ B1=100 @ I1=20 @ D1=25 @ E1=40
1100 N=2^N9 @ L9,A9,H=0
```

Program introduction.

```
1110 DISP H1$;C1$ @ A$="Pay" @ Y5=0
1120 FOR I=1 TO 3
1130 FOR X5=29 TO I+9 STEP -1
1140 GOSUB 2430 @ DISP A$[I,I];" ";
1150 NEXT X5
1160 NEXT I @ WAIT .1 @ X5=15
1170 FOR I=1 TO 5
1180 GOSUB 2430 @ DISP "          "; @ GOSUB 2430 @ DISP "Attention";
1190 NEXT I @ WAIT 1.5 @ DISP
1200 IF Y5#0 THEN DISP H1$;C1$ @ DISP
1210 FOR I=1 TO N @ B(I),A(I)=0 @ NEXT I @ W(1),W(2)=0 @ RANDOMIZE
1220 DISP "Play - "&CHR$(177)&" person or "&CHR$(178)&" people";
1230 INPUT ": ",P1$; R$ @ R1$="12" @ GOSUB 2360 @ A=R1
1240 IF A=0 THEN 1220 ELSE P1$=R$
1250 DISP "Choose - "&CHR$(229)&"asy or "&CHR$(232)&"ard";
1260 INPUT ": ",P2$; R$ @ R1$="EH" @ GOSUB 2360 @ L=R1
1270 IF L=0 THEN 1250 ELSE P2$=CHR$(NUM(R$)+32)
1280 IF L#L9 OR A#A9 THEN H=0
1290 IF A#A9 THEN I$="     "
1300 L9=L @ A9=A
1310 IF L=1 THEN A$="ABCDEFGHIJKLMNOPQRSTUVWXYZ" @ T8=T5 @ GOTO 1330
1320 A$="<>/\{}()" @ T8=T6
1330 B$="ABCDEFGHIJKL" @ B$=B$[1,N]
1340 ON A GOSUB 1850,1900
1350 DISP M$ @ WAIT .3 @ GOSUB 2100
```

Display hidden characters.

```
1360 Y5=8 @ X5=0 @ GOSUB 2430 @ D$=""
1370 FOR I=1 TO N @ D$=D$&" #" @ NEXT I @ D$=D$&" Start" @ DISP D$; @ WAIT T2
1380 FOR I=1 TO N @ X5=2^I-1
1390 GOSUB 2430 @ GOSUB 2420 @ DISP CHR$(A(I)); @ WAIT T3
1400 GOSUB 2430 @ GOSUB 2410 @ DISP "#"; @ WAIT .2
1410 NEXT I
```

Begin playing game.

```
1420 F=IP(2*RND+1) @ V=0
1430 IF A=1 THEN F=1
1440 E$=STR$(S(F))&"  " @ E$=E$[1,3] @ IF V=N9-1 THEN W(F)=0
1450 IF W(F)=1 THEN E=E1 ELSE E=I1
1460 X5=2*N+1 @ GOSUB 2430 @ WAIT .5 @ DISP I$[2*F-1,2*F]&":"&E$;
1470 BEEP 220+(F-1)*109,.1 @ IF W(F)=1 THEN DISP "*";ELSE DISP " ";
1480 X5=0 @ GOSUB 2430 @ DISP W5$; @ C(1)=1 @ C(2)=0 @ R$=KEY$
1490 FOR J=1 TO 2
```

Strobe keyboard for key input.

```
1500 R$=KEY$ @ IF R$="" THEN 1500 ELSE R7=NUM(R$)
1510 IF R7=32 THEN 1540
1520 IF R7<134 OR R7>135 THEN 1500
1530 ON R7-133 GOSUB 2450,2490 @ GOTO 1500
```

A selection is made.

```
1540 C(J)=(X5+1)/2
1550 IF B(C(J))=1 THEN GOSUB 2400 @ GOTO 1500
1560 IF C(1)=C(2) THEN GOSUB 2400 @ GOTO 1500
1570 B(C(J))=1 @ X5=X5+1 @ GOSUB 2430 @ GOSUB 2420
1580 DISP CHR$(A(C(J))+128); @ X5=X5-1
1590 NEXT J
1600 IF A(C(1))#A(C(2)) THEN 1640
```

Got a match?

```
1610 WAIT .2 @ BEEP 261,.05 @ BEEP 1046,.08
1620 S(F)=S(F)+E @ V=V+1 @ W(F)=1 @ GOSUB 1970
1630 IF V=N9 THEN GOSUB 2010 @ GOTO 1770 ELSE Q=1 @ GOTO 1670
```

No match.

```
1640 S(F)=S(F)-D1 @ W(F)=0 @ GOSUB 1970
1650 B(C(1)),B(C(2)),Q=0
1660 IF S(F)=0 AND A=1 THEN 1750
```

Reset the display.

```
1670 FOR I=1 TO 2
1680 X5=2*C(I)-2 @ GOSUB 2430
1690 IF Q=0 THEN GOSUB 2410 @ DISP " # ";ELSE DISP " ";CHR$(A(C(I)));" ";
1700 NEXT I
```

Determine game results.

```
1710 IF A=1 THEN 1440
1720 IF W(F)=1 THEN 1440
1730 IF F=1 THEN F=2 ELSE F=1
1740 GOTO 1440
1750 WAIT .3 @ BEEP 41,1 @ DISP @ DISP
1760 DISP "You lose, ";I$[2*F-1,2*F];" ..." @ WAIT T2 @ H$="H"
1770 DISP "Score is   ";I$[1,2];": ";STR$(S(1));
1780 IF A=2 THEN DISP "  ";I$[3,4];": ";STR$(S(2)) ELSE DISP
1790 WAIT T2 @ DISP H$;"igh score is ";STR$(H);"." @ WAIT T2
1800 DISP "Play again? (";CHR$(249);"es or ";CHR$(238)&"o)";
1810 INPUT ": ","y"; R$ @ R1$="YN" @ GOSUB 2360
1820 ON R1+1 GOTO 1800,1200,1830
1830 DISP "Bye for now ..."
1840 END
```

One person setup.

```
1850 S(1)=B1 @ S(2)=0
1860 INPUT "What are your two initials? ",I$[1,2];I$
1870 IF LEN(I$)<>2 THEN GOSUB 2320 @ GOTO 1860
1880 I$=UPRC$(I$) @ M$="Here we go, "&I$&".  Pay attention!"
1890 RETURN
```

Two person setup.

```
1900 FOR I=1 TO 2 @ N$=I$[2*I-1,2*I]
1910 DISP "Player number ";STR$(I);"'s initials"; @ INPUT ": ",N$;N$
1920 IF LEN(N$)<>2 THEN N$="" @ GOSUB 2320 @ GOTO 1910
1930 I$[2*I-1,2*I]=N$ @ S(I)=B1
1940 NEXT I @ I$=UPRC$(I$)
1950 M$="Pay attention now, "&I$[1,2]&" and "&I$[3,4]&"!"
1960 RETURN
```

Display the score.

```
1970 IF S(F)<0 THEN S(F)=0
1980 X5=2*N+4 @ GOSUB 2430 @ DISP "    ";
1990 GOSUB 2430 @ DISP STR$(S(F)); @ WAIT T8
2000 RETURN
```

Determine the winner.

```
2010 WAIT T2 @ BEEP 1046,.05 @ BEEP 261,.08 @ DISP
2020 H$="H" @ Z=0 @ U=1 @ DISP
2030 IF S(1)>S(2) THEN 2060
2040 IF S(2)>S(1) THEN U=2 @ GOTO 2060
2050 IF A=2 THEN DISP "It's a tie game." @ WAIT T2 @ Z=1
2060 IF S(U)>H THEN H=S(U) @ H$="New h"
2070 IF Z=1 THEN RETURN
2080 DISP "Congratulations to ";I$[2*U-1,2*U];"." @ WAIT T2
2090 RETURN
```

Character pair selection.

```
2100 Z$=A$
2110 FOR I=1 TO N/2
2120 GOSUB 2210 @ GOSUB 2240
2130 A$=Z$ @ Y$=C$ @ Z$=B$
2140 FOR K=1 TO 2
2150 GOSUB 2210 @ GOSUB 2240
2160 A(NUM(C$)-64)=NUM(Y$)
2170 NEXT K
2180 B$=Z$ @ Z$=A$
2190 NEXT I
2200 RETURN
```

Random character selection.

```
2210 L1=LEN(Z$) @ X=IP(L1*RND+1)
2220 C$=Z$[X,X]
2230 RETURN
```

Shorten character selection string.

```
2240 T$=""
2250 FOR J=1 TO L1 @ IF Z$[J,J]=C$ THEN 2270
2260 T$=T$&Z$[J,J]
2270 NEXT J @ Z$=T$
2280 RETURN
```

Response to improper input.

```
2290 GOSUB 2400 @ GOSUB 2530
2300 DISP "Please try again when prompted." @ WAIT T2 @ GOSUB 2530
2310 RETURN
```

Response to improper initials.

```
2320 GOSUB 2400 @ GOSUB 2530
2330 DISP "Please type two initials." @ I$[2*A-1,2*A]="  "
2340 WAIT T2 @ GOSUB 2530
2350 RETURN
```

Check for correct input.

```
2360 IF NOT LEN(R$) THEN R1=0 @ GOTO 2380
2370 R$=UPRC$(R$[1,1]) @ R1=POS(R1$,R$)
2380 IF R1<1 THEN GOSUB 2290
2390 RETURN
```

Sounds.

```
2400 BEEP 110,.05 @ BEEP 110,.05 @ RETURN
2410 BEEP 3000,.002 @ RETURN
2420 BEEP 6000,.002 @ RETURN
```

Position cursor.

```
2430 DISP E1$;"%";CHR$(X5);CHR$(Y5);
2440 RETURN
```

Move cursor left.

```
2450 IF X5=0 THEN GOSUB 2430 @ DISP E5$; @ X5=2*N+2 @ GOTO 2470
2460 X5=X5+2 @ GOSUB 2430 @ DISP " ";
2470 X5=X5-4 @ GOSUB 2430 @ DISP W5$;
2480 RETURN
```

Move cursor right.

```
2490 IF X5=2*N-2 THEN GOSUB 2430 @ DISP E5$; @ X5=-2 @ GOTO 2510
2500 GOSUB 2430 @ DISP " ";
2510 X5=X5+2 @ GOSUB 2430 @ DISP W5$;
2520 RETURN
```

Erase a displayed line.

```
2530 DISP E1$;"A";CHR$(13);E1$;"J";
2540 RETURN
2550 ! change lines 1080 & 1090 to vary game
```

The NAMELIST Program

The NAMELIST program is discussed in section 14 on page 213.

Specifications:

Inputs: Filename of text data file and file record entries, which consist of three fields: last name, first name, and note.

Output: Alphabetized list in data file.

Special Keys: [+], [↑], [↓], [SHIFT][↑], [SHIFT][↓], [←], [→], [SHIFT][←], [SHIFT][→], [-], [=], [/], [.], [A]-[Z].

Size:

Recorded on four card tracks, 2481 bytes.

Requires approximately 2995 bytes when initialized.

```
1000 ! Name List - 04/22/82
1010 DIM L$[110]
1020 PWIDTH INF @ WIDTH INF
1030 INPUT "Enter filename: ";A$
1040 ASSIGN # 1 TO A$
1050 N=0
```

Compute numbers of lines.

```
1060 ON ERROR GOTO 1080
1070 READ # 1 ; LO$,FO$,NO$ @ N=N+1 @ GOTO 1070
1080 OFF ERROR
```

Initialize parameters.

```
1090 M,P=0 @ L1$,F1$=""
```

Start of command input loop.

```
1100 L$="Name List" @ IF P=0 THEN 1140
1110 READ # 1,P ; LO$,FO$,NO$
1120 GOSUB 1870
1130 IF LEN(L$)>32 THEN DISP L$[1,32] @ GOTO 1150
1140 DISP L$
```

Get command key.

```
1150 K$=KEY$ @ IF K$="" THEN 1150
1160 K$=UPRC$(K$) @ K=NUM(K$)
1170 IF K$="+" THEN 1360
1180 IF K$="=" THEN 1550
1190 IF K=132 THEN P=P-(P>0) @ GOTO 1100
1200 IF K=133 THEN P=P+(P<N) @ GOTO 1100
1210 IF K=134 OR K=166 THEN 1130
1220 IF K=164 THEN P=P>0 @ GOTO 1100
1230 IF K=165 THEN P=N @ GOTO 1100
1240 IF K$>="A" AND K$<="Z" THEN 1610
1250 IF K$>="0" AND K$<="9" THEN 1610
1260 IF K$="*" THEN 1670
1270 IF K$="¦" THEN 1580
1280 IF K$="-" THEN 1780
1290 IF K$=" " THEN P=N>0 @ GOTO 1100
1300 IF K$="/" THEN M=ABS(M-1) @ GOTO 1100
1310 IF K$="." THEN ASSIGN # 1 TO * @ END
1320 IF K<>135 AND K<>167 THEN 1150
1330 I=LEN(L$) @ IF I<33 THEN 1150
1340 IF M=1 THEN DISP L$[33] @ GOTO 1150
1350 DISP L$[I-31] @ GOTO 1150
```

Add an entry.

```
1360 GOSUB 2000
1370 IF LEN(L0$)+LEN(F0$)>0 THEN 1390
1380 BEEP @ DISP "Error - Missing Name" @ GOTO 1360
1390 INPUT "Note: ";L$
1400 GOSUB 1940
1410 IF LEN(L$)>32 THEN L$=L$[1,32]
1420 N1$=L$
1430 IF LEN(L0$)=0 THEN L0$=" "
1440 GOSUB 2070
1450 IF L1$=" " THEN L1$=""
1460 IF F=1 THEN GOSUB 2100 @ GOTO 1460
1470 IF P>N THEN 1520
1480 FOR I=N TO P STEP -1
1490 READ # 1,I ; L0$,F0$,N0$
1500 PRINT # 1,I+1 ; L0$,F0$,N0$
1510 NEXT I
1520 PRINT # 1,P ; L1$,F1$,N1$
1530 N=N+1
1540 GOTO 1100
```

Search for an entry.

```
1550 GOSUB 2000
1560 GOSUB 2070
1570 GOTO 1590
```

Continue search.

```
1580 GOSUB 2100
1590 IF F=0 THEN P=0
1600 GOTO 1100
```

Alpha-numeric key.

```
1610 P=0
1620 P=P+1
1630 IF P>N THEN P=N @ GOTO 1100
1640 READ # 1,P ; LO$,FO$,NO$
1650 IF UPRC$(LO$)<K$ THEN 1620
1660 GOTO 1100
```

Print entries.

```
1670 IF N=0 THEN 1100
1680 FOR I=1 TO N
1690 READ # 1,I ; LO$,FO$,NO$
1700 GOSUB 1870 @ IF M=0 THEN 1740
1710 K=POS(L$,"   ") @ IF K<>0 THEN L$[K,K+2]=" . " @ GOTO 1710
1720 IF L$[30,30]<>"." THEN 1740
1730 K=POS(L$,".") @ L$=L$[1,K-1]&" "&L$[K,30]&L$[32]
1740 PRINT L$
1750 NEXT I
1760 PRINT ""
1770 GOTO 1100
```

Delete an entry.

```
1780 IF P=0 THEN 1100
1790 PRINT # 1,P ; ""
1800 IF P=N THEN P=N-1 @ GOTO 1840
1810 FOR I=P+1 TO N
1820 READ # 1,I ; LO$,FO$,NO$
1830 PRINT # 1,I-1 ; LO$,FO$,NO$ @ NEXT I
1840 PRINT # 1,N ; ""
1850 N=N-1
1860 GOTO 1100
```

Create display line.

```
1870 L$=LO$
1880 FO=LEN(FO$)
1890 IF LEN(L$)>0 AND FO>0 THEN L$=L$&","
1900 IF FO>0 THEN L$=L$&" "&FO$
1910 L$=L$&" "
1920 L$=L$&" " @ IF M=1 AND LEN(L$)<32 THEN 1920
1930 L$=L$&NO$ @ RETURN
```

Remove end blanks.

```
1940 IF LEN(L$)=0 THEN RETURN
1950 IF L$[1,1]=" " THEN L$=L$[2] @ GOTO 1940
1960 I=LEN(L$)
1970 IF L$[I,I]=" " THEN I=I-1 @ GOTO 1970
1980 L$=L$[1,I]
1990 RETURN
```

Input names.

```
2000 INPUT "Last Name: ";L$
2010 GOSUB 1940 @ IF LEN(L$)>32 THEN L$=L$[1,32]
2020 LO$=L$
2030 INPUT "First Name: ";L$
2040 GOSUB 1940 @ IF LEN(L$)>32 THEN L$=L$[1,32]
2050 FO$=L$
2060 RETURN
```

Search routine.

```
2070 P=0
2080 L1$=L0$
2090 F1$=F0$
```

Continue search routine.

```
2100 F=0
2110 P=P+1
2120 IF P>N THEN RETURN
2130 READ # 1,P ; L0$,F0$,N0$
2140 L0$=UPRC$(L0$)
2150 F0$=UPRC$(F0$)
2160 IF L1$=" " THEN 2240
2170 IF LEN(L1$)=0 THEN 2210
2180 IF UPRC$(L1$)>L0$ THEN 2110
2190 IF UPRC$(L1$)<L0$ THEN RETURN
2200 IF LEN(F1$)>0 AND UPRC$(F1$)<F0$ THEN RETURN
2210 IF LEN(F1$)=0 THEN 2260
2220 IF UPRC$(F1$)<>F0$ THEN 2110
2230 GOTO 2260
2240 IF LEN(L0$)>0 OR UPRC$(F1$)<F0$ THEN RETURN
2250 IF UPRC$(F1$)>F0$ THEN 2110
2260 F=1
2270 RETURN
```

Appendix G

Glossary

Contents

Special Symbols 320
Terms 321

Special Symbols

@	Enables multistatement program lines, key redefinitions, and command appointments.
!	Allows end-of-line comments in program lines.
?	Two special uses: • The default INPUT prompt. • Indicates an unknown file in a catalog entry.
**	In TIME mode, specifies 24-hour notation. In APPT mode, specifies either 24-hour notation or an appointment outside a 100-year range from the current date.
A	Three special uses: • Indicates an appointment file in a catalog entry. • Specifies an appointment that reschedules after *acknowledgement*. • Specifies an *absolute* adjustment in the ADJST template.
B	Indicates a BASIC file in a catalog entry.
E	Exponent—represents a power of 10.
I	Indicates an interchange (LIF1) file in a catalog entry.
L	Specifies a Language Extension (LEX) file in a catalog entry.
N	Two special uses: • Specifies a *normal*, one-time appointment in the APPT template. • Specifies a *normal* clock adjustment in the ADJST template.
P	Indicates a private file in a catalog entry.
R	Specifies a *repeating* appointment that reschedules itself immediately after coming due.
T	Indicates a text file in a catalog entry.

Terms

A

acknowledge an appointment	To press [ATTN] or [SHIFT] [DEL] when a due appointment is displayed.
allocate	To reserve memory for program variables prior to running the program. Programs are allocated by RUN and CALL commands.
allowable statement	A BASIC statement that may follow a THEN, ELSE, ON ERROR, or ON TIMER keyword. All HP-75 statements are allowable except DATA, DEF FN, DIM, END DEF, FOR, IF, IMAGE, INTEGER, NEXT, OPTION BASE, REAL, ON ERROR, ON TIMER, and SHORT statements.
annunciator	One of four areas on the display window that indicates a certain machine condition—**BATT**, **ERROR**, **APPT**, **PRGM**.
appointment	A time and date stored in the special file named appt in memory or on a mass storage medium. May include a message, command, or alarm information.
appt	The name of the file that consists of appointments. The filename appears unquoted in all commands.
APPT mode	The operating state of the HP-75 that enables you to schedule appointments and check calendar dates.
argument	The numeric or string information acted on by a function.
array element	One of the values in a numeric array; may be referenced by its *subscript*.
ASCII	The American Standard Code for Information Interchange, the standard used by the HP-75 to represent its 256 character set internally. Each HP-75 character correspondes to a decimal code of 0 through 255.

B

BASIC	Beginner's All-purpose Symbolic Instruction Code, The programming language of the HP-75. The keyword BASIC is used in EDIT commands to create new program files.
Boolean value	One of two values, 1 or 0, that indicates a true of false condition.
branch	To transfer program execution to a specified program statement.
byte	A standard unit of memory, equivalent to eight bits (1's and 0's) of data or to one character of information.

C

calculator variable	A numeric or string variable that is not created in a program but from the keyboard. Calculator variables are not accessible to programs. See also: *program variable*.
calling program	A program that begins the execution of another by means of the CALL statement.
CARD	Used in a COPY or CAT command, CARD specifies a file read or written by the card reader. The keyword CARD is not quoted.

card file	A BASIC, text, keys, appointment, LEX, or interchange file that resides on a magnetic card or cards.
carriage control	Certain escape code sequences used to control the position of the paper and print head in PRINTER IS devices and the position of the cursor in DISPLAY IS devices. See: *escape code.*
carriage-return	A control character (decimal code 13) that causes the cursor to return to the left edge of the display.
carriage-return/line-feed	A sequence of two characters that is normally generated by the termination of keyboard entries and PRINT and DISP statements. The seven system keys and the ↑ and ↓ keys send carriage-return/line-feeds to display devices.
catalog entry	One display line of information showing the name, type, length, time, and date of a file in memory or on a mass storage medium.
character	Any of the 256 different letters, digits, or symbols that can be displayed by means of the CHR$ function. A subset of these may be displayed directly from the keyboard.
character string	A group of characters that can be manipulated as a single unit. See also: *string expression.*
command	Commonly defined as instructions which, when entered from the keyboard, operate on programs or affect the machine directly. The difference between commands and statements in the HP-75 is indistinct because most commands and statements can either be entered from the keyboard or used in programs. The use of separate terms for different types of instructions in machines like the HP-75 is largely a matter of tradition.
command appointment	An appointment whose Note field specifies one or more commands or statements.
concatenate	To join string expressions (with &) or to join commands and statements (with @).
conditional branch	A program branch that depends on the outcome of a specific test.
control character	A special display character whose decimal code is 0 through 31 or 127.
controller	The computer that controls peripherals—your HP-75.
current file	The file in which the file pointer is currently located. The file directly accessible for editing, running, etc.
current line number	The line number of the pending line. See: *line number, pending line.*
cursor	A blinking symbol to show your location on the display line. The *replace* cursor (▮) lets you type over characters; the *insert* cursor (◆) inserts characters as you type.

D

data pointer	The mechanism used to keep track of the next DATA item to be read.

deallocate — The process of returning to free memory, all of the memory needed to initialize program variables. Various system commands deallocate programs. See also: *allocate*.

decimal code — The numeric equivalent (base 10) of a display character; may range from 0 through 255.

default value — A value supplied by the HP-75 when a parameter is not specified in a command, statement, or template, or a value supplied when an improper operation occurs to enable execution to continue.

delimiter — A character (for example, " and ,) used to set off other characters or to separate parameters.

device code — There are two types of device codes:

- One of two words that may appear after the colon (:) in a file specifier, CARD or PCRD, to specify a card file.
- A one- or two-character abbreviation that specifies one of the devices in an HP-IL system. May be one or two letters, a letter and digit, or a digit and letter.

dimension — To declare the maximum length of a string variable, the number of elements in a numeric array, or the precision of a numeric variable. Variables are dimensioned by DIM, REAL, SHORT, and INTEGER statements.

display character — Any character that can be displayed directly from the keyboard, following a [SHIFT] [I/R] keystroke.

display device — An HP-IL output device whose device code has appeared in a DISPLAY IS declaration.

display fields — The areas of TIME and APPT displays that are reserved for specific types of information. In the APPT template, for example, the N display field is reserved for the type of appointment you wish to schedule, N, A, or R.

display format — The way that information (like the date and time) appears in the display window.

display line — The line currently in the HP-75 display, up to 96 characters in length, minus one character position for the cursor and one or more character positions for the prompt. The display *window* displays up to 32 characters at a time.

due appointment — An appointment whose time has come.

E

editing key — A key or keystroke combination that controls the input buffer.

end-of-file mark — A mark, invisible to the user, placed by the operating system at the end of every file to separate files.

escape character — The control character (decimal code 27) generated by [CTL] [BACK] and CHR$(27).

escape code — A sequence of characters, the first being the escape character, that may be interpreted as an HP-IL or display instruction.

escape keystroke — [CTL][BACK].

evaluate — To compute the value of an expression. The result is always a string or numeric constant.

EXACT mark — A timing mark that is set for the system clock when you type `exact` [RTN] in TIME mode.

execute — To perform any HP-75 operation.

F-G

file — A set of lines in memory or on a mass storage medium that can be manipulated as a single unit and which has a unique name.

file number — An integer from 1 through 9999 that specifies a BASIC or text file in an ASSIGN #, PRINT #, or READ # statement.

file pointer — A mechanism in memory that keeps track of the current line of the current BASIC, text, or appointment file.

file specifier — A string of characters in the form '*filename* [:*device code* [/*password*]]' that names a BASIC, text, keys, appointment, LEX, or interchange file residing on a mass storage medium. A file specifier may consist of a filename with a device code, a filename with a device code and a password, or just the word CARD. All file specifiers except CARD may be specified by string expressions. File specifiers may be terminated by blanks.

filename — A string of one to eight characters that specifies a file in memory. The first character must be a letter or period. Remaining characters may be any combination of letters and digits. May also be APPT or KEYS. All filenames except APPT and KEYS may be specified by string expressions and may be terminated by blanks.

function — A built-in routine that operates on zero, or more arguments to produce a single string or numeric value. A *user-defined* function declared in a program may operate on an arbitrary number of arguments.

H

hierarchy — The prescribed order in which the HP-75 performs operations in an expression.

HP-IL — The Hewlett-Packard Interface Loop. A system of peripherals that's controlled by the HP-75.

I-J

image format string — The characters in an IMAGE, DISP USING, or PRINT USING statement that specify the formatting of information sent to display and printer devices.

immediate-execute key — A key or keystroke combination that displays and then executes a command, statement, or expression when pressed.

initialize a program	To prepare the machine for program execution by allocating memory for variable values, initializing variables to undefined values, checking for errors, and setting up other run-time conditions. Initializing occurs when [RUN] is pressed in EDIT mode and when RUN and CALL commands are executed.
initialize a variable	To assign an initial value to a program variable.
input buffer	A 95-character location in memory that contains keyboard input. May be viewed by pressing [CTL][FET].
interchange file	A Logical Interchange Format (LIF1) file used to interchange information between the HP-75 and other computers.
interface	The circuitry that connects a controller and peripherals and enables them to function together.
I/O	Stands for *Input/Output*, an operation that involves receiving input from devices or sending output to them. The keyboard is the normal HP-75 input device, and the display is the normal HP-75 output device. The card reader is also a built-in I/O device.

K

key definition	The current function of a key or keystroke combination, which may be to display a string of characters or to execute one or more instructions.
KEYS	Specifies the keys file. The name appears unquoted in all commands.
keys file	The special text file that stores all key redefinitions.
keyword	A word that has special meaning for the HP-75 (like ON, TO, DIM, and GOTO). Keywords form the kernel of program lines.

L

Language Extension file	A special program file available on mass storage media and in plug-in ROMs that may define new keywords and extend computer capabilities.
left justified	To begin an output string in the leftmost column of a display or print field.
line editing	Adding to, modifying, or deleting characters in the input buffer.
line-feed	A control character (decimal code 10) that causes the HP-75 to advance to a new display line. Generated by [CTL][J] and CHR$(10).
line number	An unsigned integer from 0 through 9999 that determines the order of statements in a BASIC file or the order of lines in a text file.
listener	An HP-IL peripheral that has been declared a DISPLAY IS or PRINTER IS device.
loop counter	The simple numeric variable in a FOR-NEXT loop that controls the number of loop iterations.
looping	Repeatedly executing a series of statements, usually until a specified condition is satisfied.

M

mass storage medium	A magnetic card, cassette, or disc that may store files from the computer externally.
mass storage device	An I/O device like the card reader or HP 82161A Digital Cassette Drive which may be used to copy files from memory to mass storage media and from mass storage media to memory.
modifier keys	The [SHIFT] and [CTL] keys, used to modify the functions of typewriter, editing, and system keys.
multistatement line	A program line that contains two or more statements or commands, concatenated with the @ symbol.

N

nested loop	One FOR-NEXT loop contained within another.
nested subroutine	One subroutine that is executed as part of another subroutine.
null string	Character string of length zero, specified by '' or "".
numeric array variable	An ordered collection of numbers, called *elements*, that are specified by their row or their row and column subscripts.
numeric constants	A fixed numeric value within the range of the HP-75, for example, 3.14 and -5E99.
numeric expression	A valid combination of constants, numeric variables, and numeric functions that are joined by arithmetic, relational, and logical operators. When evaluated, a numeric expression is reduced to a single numeric constant.
numeric function	An operation that—given the appropriate type, number, and range of arguments—computes a single numeric value.

O

operator	A symbol that combines or compares the values of two expressions. *Arithmetic*, *relational*, and *logical* operators result in a numeric quantity; the *string* operator, &, results in a string quantity.

P-Q

parameter	A general term referring to any string or numeric value that is used in a keyboard or program instruction.
password	Two types are available: • The *system* password is any string expression specified in a LOCK command. The first eight characters must be entered exactly as specified in order to regain control of the HP-75 after it turns off. • A *mass storage* password may be one to four letters or digits that appear after the slash (/) in a mass storage file specifier. The mass storage password limits access to the file. Uppercase and lowercase letters are treated identically in mass storage passwords, but not in system passwords.

pending line	The line of the current BASIC or text file available for editing—that is, the line displayed when you press [FET] and then [RTN] in EDIT mode.
pending return	A pending transfer of program execution that awaits the completion of a subroutine or a called program.
peripheral	An external HP-IL device that's controlled by the HP-75.
precision	A number of significant digits with which the HP-75 stores a numeric value.
private file	A card file or plug-in ROM file whose lines can be copied to memory and run but can't be examined, changed, or recopied.
process a due appointment	To cause the HP-75 to display the message or execute the contents of a Note field. Processing occurs when you press [RUN] in APPT mode, for example.
program	A set of instructions which performs some computing task and which controls the input, processing, and output of data. Programs are stored in BASIC files, LEX files, and ROMs.
program line	One line of a BASIC file, containing a line number and one or more complete instructions.
program pointer	The mechanism used to keep track of the next command or statement to be executed in a BASIC program.
program variable	A variable defined in and only accessible to, a program. See also: *calculator variable.*
prompt	The symbol that appears at the left edge of the display to indicate readiness for user input. The HP-75 has three prompts: • The BASIC prompt (>) is in the display while you edit BASIC files. • The text prompt (:) is in the display while you edit text files. • The INPUT prompt appears during the execution of an INPUT statement. If no prompt is specified in the INPUT statement, the prompt is a ?.

R

random access memory (RAM)	The circuits that store user created programs, data, text, appointments, variable values, and related information.
read-only memory (ROM)	Permanent memory that can not be changed or erased; available through the HP-75 operating system and through plug-in ROM modules.
recursion	See: *recursion.* A recursive process or procedure is one which is defined in terms of itself.
routine	A program, program segment, or subroutine that supports the execution of a larger program.
run-time error	An error that occurs during program execution after the program has been initialized.

S

schedule an appointment	To enter an appointment in the appt file.

simple numeric variable — A name (a single letter or a letter-digit combination) representing a location in memory that stores a numeric value. A simple numeric variable can be used interchangeably with the number it references.

statement — Commonly defined as instructions, within a program, which control the flow of program execution; usually entered as lines in a program. The difference between statements and commands in the HP-75 is indistinct because most statements and commands can either be entered from the keyboard or used in programs. The use of separate terms for different types of instructions in machines like the HP-75 is largely a matter of tradition.

string constant — An arbitrary collection of characters delimited by quotation marks (' ' or " "). Also called a *literal* or *quoted* string.

string expression — Any string constant, string variable, substring, or string function, or any combination of string expressions concatenated with the & operator. Also called a *character string* or *string*.

string function — A predefined operation that evaluates numeric or string arguments and returns a single string value. Some string functions, like TIME$, require no argument.

string variable — A name (either a letter or a letter-digit combination), followed by $, that represents a location in memory where character information may be stored. A string variable may be used interchangeably with the character information it references.

subroutine — A program segment that begins execution as a result of a GOSUB statement and that returns control when it executes a RETURN statement.

substring — A portion of a string variable made up of zero or more contiguous characters. A substring is specified by a subscript or subscripts enclosed within brackets ([ ]), following the name of a string variable.

subscript — Also called an *index*.

- A number that specifies the row or column location of a numeric array element.
- A number that specifies the beginning or ending character positions of a substring.

syntax — The rules governing the spelling of keywords, variable names, operators, filenames, etc., and the construction of commands and statements.

system key — One of seven keys that control the operation of the computer: [ATTN], [TIME], [EDIT], [APPT], [FET], [RTN], and [RUN].

T

text — An arbitrary collection of characters.

text editing — The process of composing text files.

text file — A file of numbered lines of characters.

TIME command field — The five spaces at the end of the TIME display that are reserved for any of five TIME mode commands.

TIME display	The clock reading that appears when you press [TIME].
timer number	An integer from 0 through 1000 that is used in an ON TIMER # statement to specify a system timer.
track	One of two surfaces of a magnetic card that may record up to 650 bytes of a single file, not including the catalog information.
typewriter key	A key that normally displays a letter, digit, or other symbol.
typing aid	A key or keystroke combination that enters a string of characters when pressed, as if they had been typed in.

U

user-defined function	A numeric or string function that's defined in a program by means of DEF FN, LET FN, and END DEF statements.
unconditional branch	A branch that occurs every time the statement is executed.

V

value	Value is defined for numeric or string variables: • The actual number that is represented by a symbol (5), numeric variable (A), function (sqr(2)), or expression (2+2). • The actual characters that are represented by a string expression.
variable	A name that represents numeric or string information. May be a simple numeric variable (A), a numeric array variable (A(,)), or a string variable (A$).
volatile file	A file whose filename begins with a period.It is purged from memory the next time the HP-75 is turned off.

W-X-Y-Z

workfile	A temporary BASIC or text file.

Appendix H

Syntax Summary

Contents

Operational Precedence 330
String Operators 330
Numeric Precision 331
Variables 331
- Simple Numeric Variables 331
- Numeric Array Variables 331
- String Variables 331
- Substrings 331

BASIC Predefined Functions 331
- Numeric Functions 332
- String Functions 333
- Print Function 334

TIME Mode Commands 334
Syntax Guidelines for BASIC Statements and System Commands 334

Operational Precedence

The HP-75 arithmetic, relational, and logical operators are listed according to their operational precedence:

Performed first.

( )	Nested parentheses are evaluated from the inside out.
Functions.	
^	
NOT	
*, /, DIV or \ (CTL /)	Expressions are evaluated from left to right for operators at the same level.
+, -	
=, >, >=, <, <=, <> or #	
AND	
OR, EXOR	

Performed last.

String Operators

The string concatenator (&) is used to join string expressions.

The relational operators (=, >, >=, <, <=, <>, and #) may be used to compare string expressions; they return a 1 if the comparison is true and a 0 if the comparison is false. Two strings are considered equal if they contain exactly the same characters in the same order. For inequalities, two strings are compared character

by character, from left to right, until a difference is found. If one string ends before a difference is found, the shorter string is considered the lesser. If a difference is found, the decimal code of the two differing characters are used to determine which string is the lesser.

Numeric Precision

Type	Precision	Range
REAL	12 digits	$\pm 9.99999999999 \times 10^{\pm 499}$
SHORT	5 digits	$\pm 9.9999 \times 10^{\pm 99}$
INTEGER	5 digits	±99999

Variables

Simple Numeric Variables (Examples: A1, B, C3)

The name consists of a letter or a letter and one digit. REAL precision is assumed unless SHORT or INTEGER type is declared.

Numeric Array Variables (Examples: A1(50,5), B(20,20), C3(10))

The name consists of a letter or a letter and one digit. An array name can be the same as a simple variable name used elsewhere in the program, but a one-dimensional array cannot have the same name as a two-dimensional array. Subscripts dimension the row or row and column in DIM, REAL, INTEGER, and SHORT declarations. The lower bound of an array subscript is 0 unless OPTION BASE 1 is specified before all array references. The default upper bound for row and column subscripts is 10.

Entire arrays may be referenced in PRINT # and READ # statements by specifying just the array name with parentheses (for example, C()). A comma between the parentheses specifies a two-dimensional array (for example, C9(,)).

String Variables (Examples: A1$, B$, C3$)

The name consists of a letter or a letter and one digit followed by a dollar sign. The default length is 32 characters unless otherwise specified in a DIM statement. Strings may be of any length from zero characters to a maximum limited by available memory. A string variable is dimensioned by specifying the variable name followed by its length between brackets: DIM A1$[25]. The *length* must be a nonnegative integer.

Substrings (Examples: A1$[2,25], B$[5], C$[I,I])

Substrings are specified by one or two numbers or numeric expressions enclosed within brackets. Two subscripts separated by a comma specify beginning and ending character positions, respectively. A single subscript specifies a beginning character—the substring extends to the end of the string.

BASIC Predefined Functions

In the following tables, X and Y specify any two numeric expressions, and S$ and T$ specify any two string expressions.

Numeric Functions

Given the appropriate number of arguments and type of arguments (numeric or string expressions), each numeric function returns a single numeric constant.

ABS(X)	Absolute value of X.
ACOS(X)	Arccosine of X, in Quadrant I or II.
ANGLE(X,Y)	Arctangent of Y/X, in proper quadrant. That is, returns the angle θ formed between the x-axis and the point (x,y), such that $-\pi < \theta \leq \pi$.
ASIN(X)	Arcsine of X, in Quadrant I or IV.
ATN(X)	Arctangent of X, in Quadrant I or IV.
CEIL(X)	Smallest integer $\geq$ X.
COS(X)	Cosine of X.
COT(X)	Cotangent of X.
CSC(X)	Cosecant of X.
DATE	The date in *yyddd* format based on the clock setting.
DEG(X)	Radian-to-degree conversion of X.
EPS	Smallest machine number (1.E-499).
ERRL	The line number of the most recent error or warning.
ERRN	The identification number of the most recent error or warning.
EXP(X)	e^X.
FLOOR(X)	Same as INT(X).
FP(X)	Fractional part of X.
INF	Largest machine number (9.9999999999E499).
INT(X)	Largest integer $\leq$ X.
IP(X)	Integer part of X.
LEN(S$)	The length of string S$.
LOG(X)	Natural logarithm of X, X > 0.
LOG10(X)	Log to the base 10 of X, X > 0.
MAX(X,Y)	If X > Y then X, else Y.
MEM	The number of bytes of available memory.
MIN(X,Y)	If X < Y then X, else Y.

MOD(X,Y)	X modulo Y: X-Y*INT(X/Y).
NUM(S$)	The decimal code of the first character of S$.
PI	3.14159265359.
POS(S$,T$)	Searches string S$ for the first occurrence of T$. Returns the starting position if found; 0 if not.
RAD(X)	Degree-to-radian conversion of X.
RES	The last numeric result to be displayed or printed.
RMD(X,Y)	Remainder of X/Y: X-Y*IP(X/Y).
RND	Next number, R, in sequence of pseudo-random numbers, $0 \leq R < 1$.
SEC(X)	Secant of X.
SGN(X)	The sign of X: -1 if $X < 0$, 0 if $X = 0$, 1 if $X > 0$.
SIN(X)	Sine of X.
SQR(X)	Positive square root of X.
TAN(X)	Tangent of X.
TIME	The number of seconds since midnight.
VAL(S$)	Returns the numeric value of a string composed of digits, decimal point, and/or exponent.

String Functions

String functions return zero or more characters of information.

CAT$(X)	The catalog entry of the specified file, 32 characters in length. Files are numbered in order of their appearance in the system catalog. CAT$(0) returns the catalog of the current EDIT file. For $X < 0$, CAT$(X) returns the catalog of the currently initialized BASIC file, if any.
CHR$(X)	The character whose decimal code is MOD(X,256).
DATE$	The date in a *yy/mm/dd* format.
KEY$	The display character of the currently depressed key or keystroke combination. Returns the null string if no key is depressed.
STR$(X)	The string information contained in the digits, decimal point, sign, and exponent of X.
TIME$	The time in a *hh:mm:ss* format, using 24-hour notation.
UPRC$(S$)	Converts S$ to all uppercase letters.
VER$	A six-character string indicating the operating system version.

Print Function

TAB(X)	Causes the following DISP or PRINT item to be output beginning at column X, where column 1 is the left margin.

TIME Mode Commands

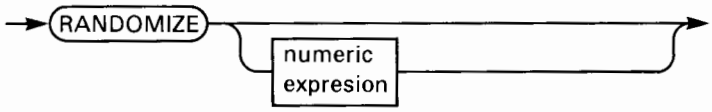
The following commands are nonprogrammable and executable in TIME mode only.

ADJST	Displays the ADJST template that's used to adjust the clock setting.
EXACT	Sets a timing mark for clock calibration.
RESET	Clears previous EXACT marks and the current speed adjustment factor.
SET	Displays the set-time template that's used to set the system clock.
STATS	Displays the STATS template that's used to specify month/day and hour display formats and to specify the YEAR or the EXTD (extended) calendar in APPT mode.

Syntax Guidelines for BASIC Statements and System Commands

Items enclosed in ovals and circles may be typed in uppercase or lowercase letters. Items in boxes are the parameters that can (or must) be used in the command or statement.

Example:



The lines connecting the boxes can only be followed in one direction when constructing the command or statement. An item is optional if there is a valid path around it. Thus, any combination of elements can be used as long as the arrows are followed in the proper direction. This means there are two acceptable forms of the RANDOMIZE command above, RANDOMIZE by itself and RANDOMIZE *numeric expression*.

When they enclose a parameter, quotation marks may be single (' ') or double (" "), although they must be paired. Any quoted parameter, like '*filename*' and ':*device code*', may be specified by a string expression.

Except for the first two letters of the keyword, which may have no spaces between them, a command or statement may be entered with an arbitrary number of spaces.

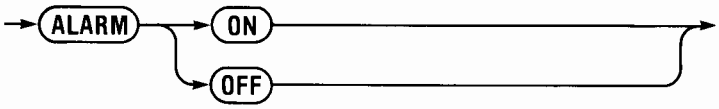
A full arrow (⟶) heading into the keyword (as in the RANDOMIZE example above) indicates:

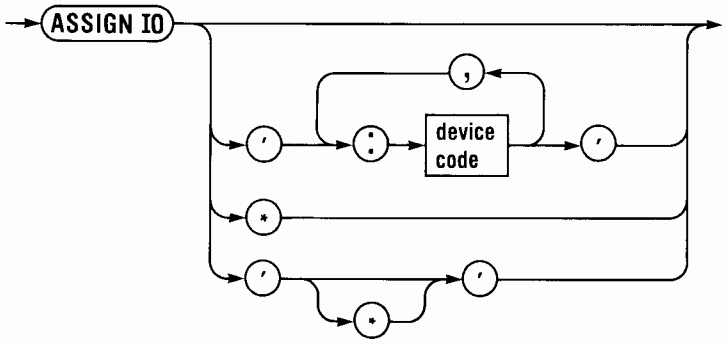
- The command or statement may follow another instruction in the same program line, joined by the ⓔ symbol.
- The command or statement may appear in an IF...THEN...ELSE statement.

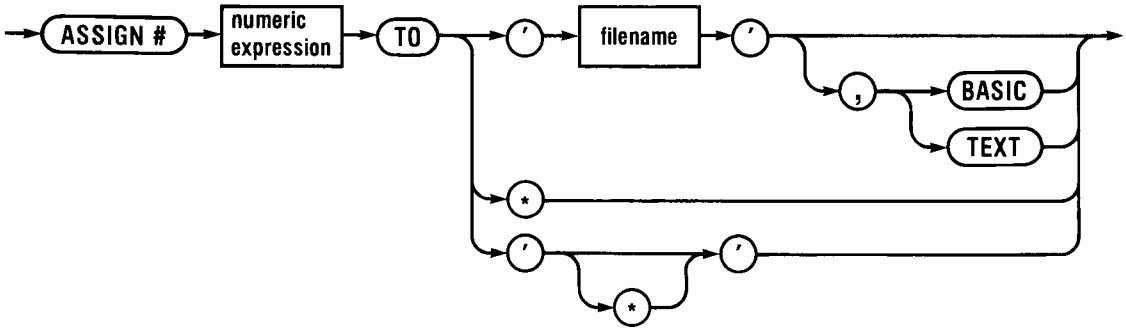
A half-arrow (⟶) heading into the keyword signifies that the statement is not allowed after THEN, ELSE, ON ERROR, or ON TIMER. The absence of an arrow indicates that the statement must be entered as the first instruction of a program line.

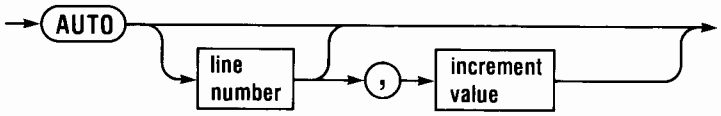
A full arrow leading out of the instruction indicates that it may be followed by another instruction in the same line. A —→X indicates that no other instruction may follow. A —→| indicates that another instruction may follow but that that instruction will never be executed, as after an END statement.

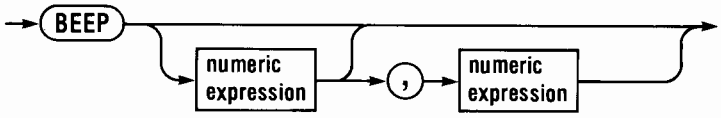
Refer to appendix D, Reference Tables, for a listing of acceptable keyword abbreviations.

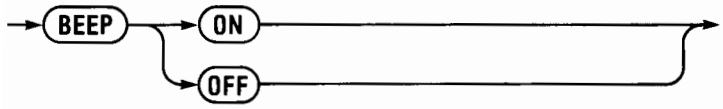


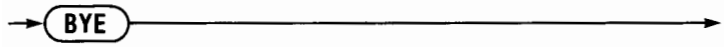


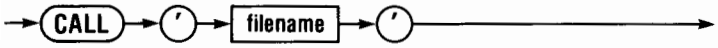


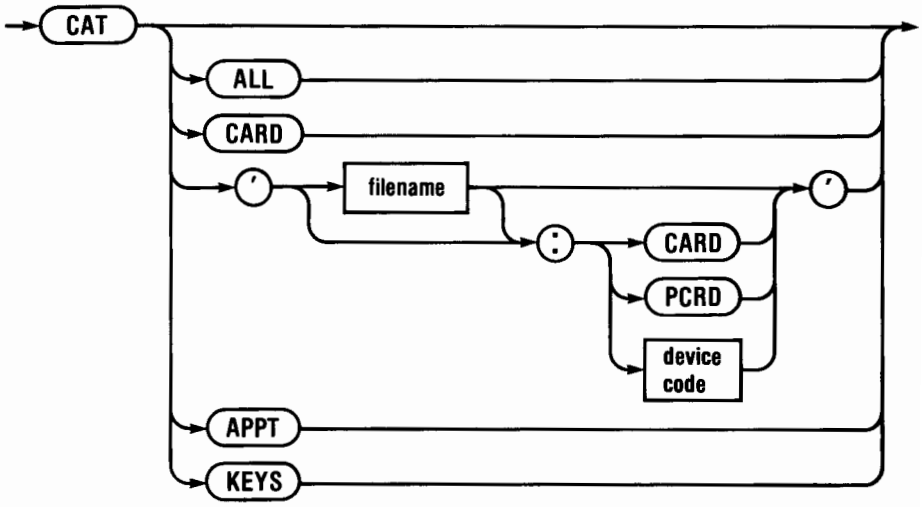




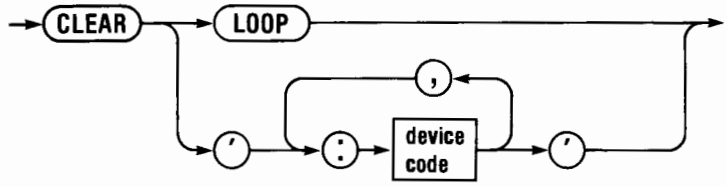

BEEP
ON
OFF

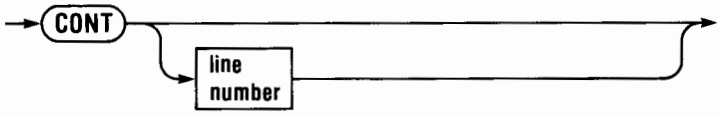

BYE

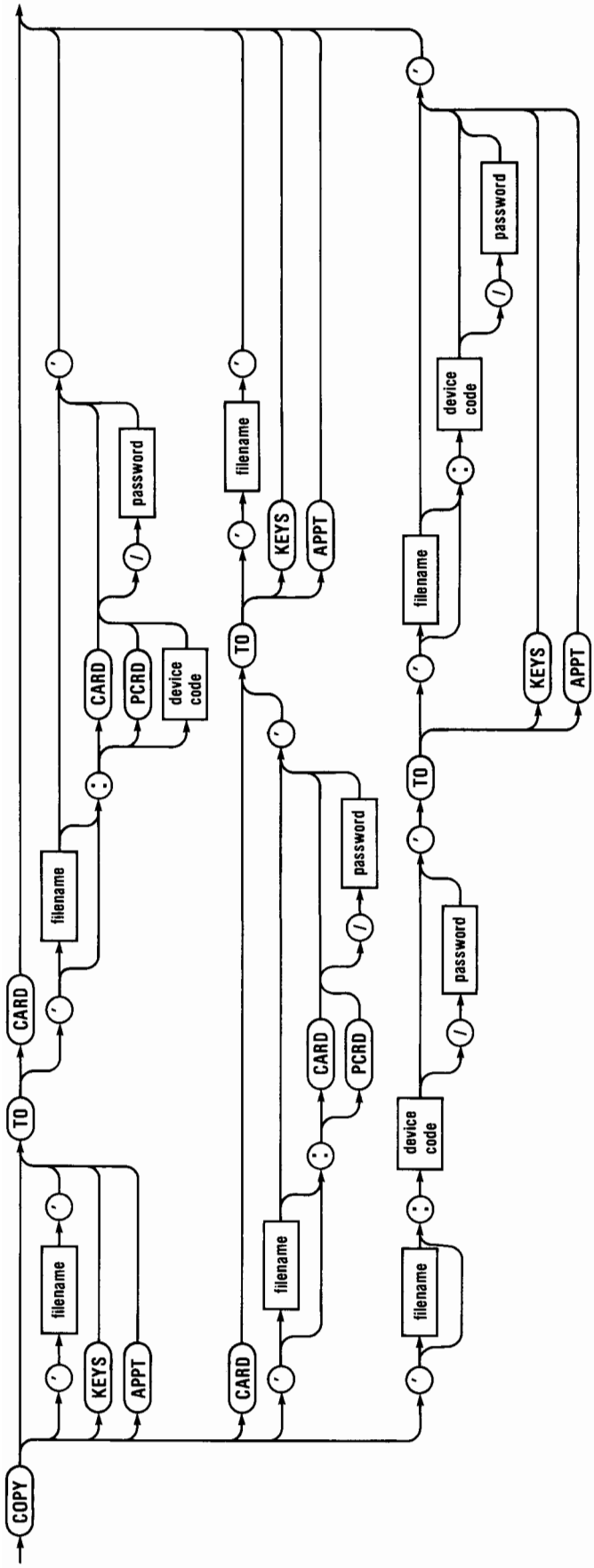

CALL
'
filename
'

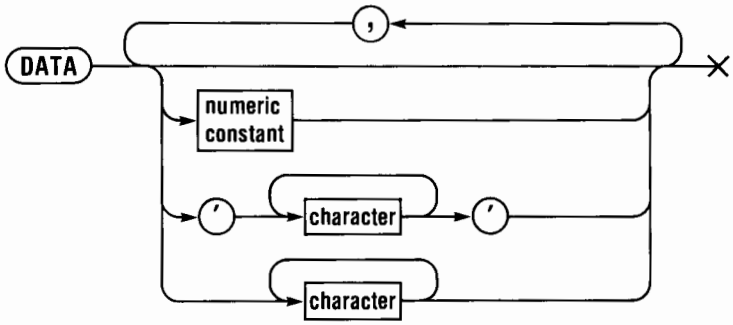

CAT
ALL
CARD
'
filename
:
CARD
PCRD
device code
'
APPT
KEYS

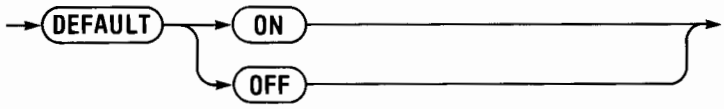
CLEAR
VARS

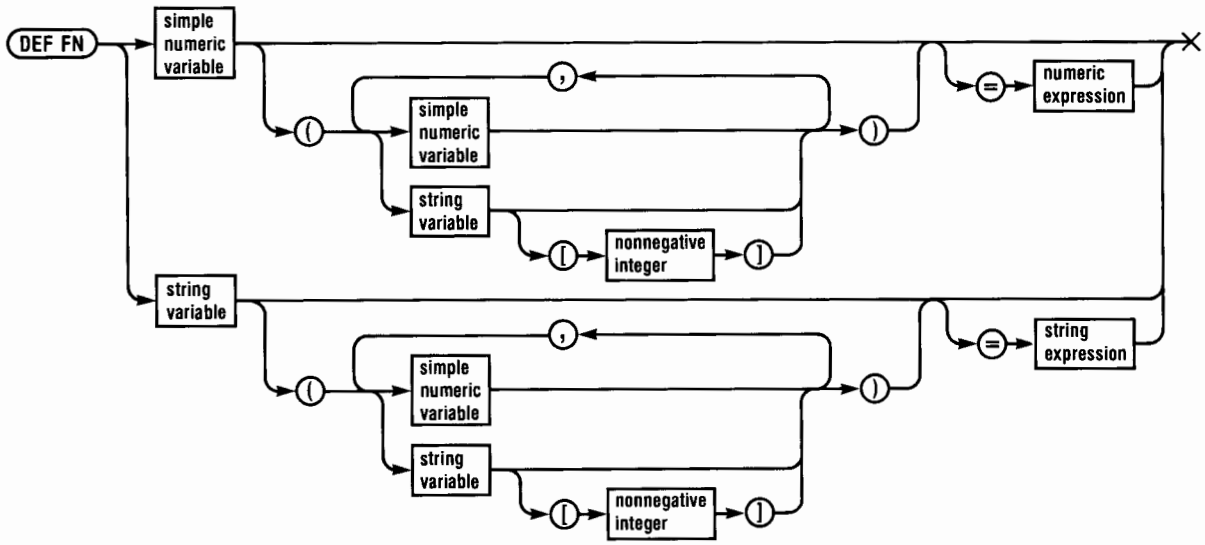

CLEAR
LOOP
,
'
:
device code
'

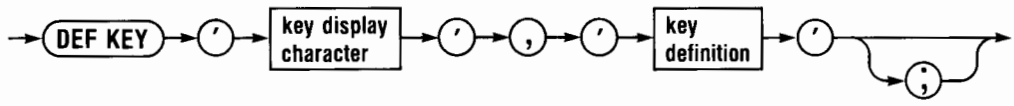

CONT
line number


COPY
KEYS
APPT
filename
TO
CARD
PCRD
device code
password

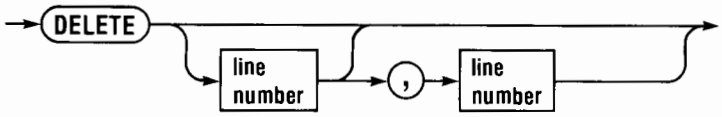

DATA
,
numeric constant
'
character
'
character

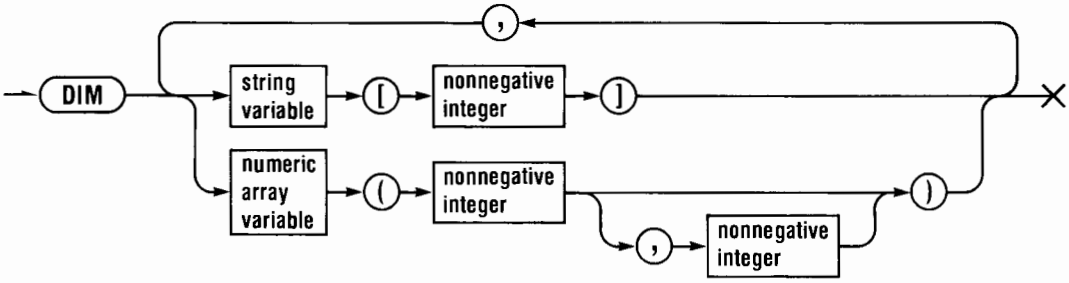

DEFAULT
ON
OFF

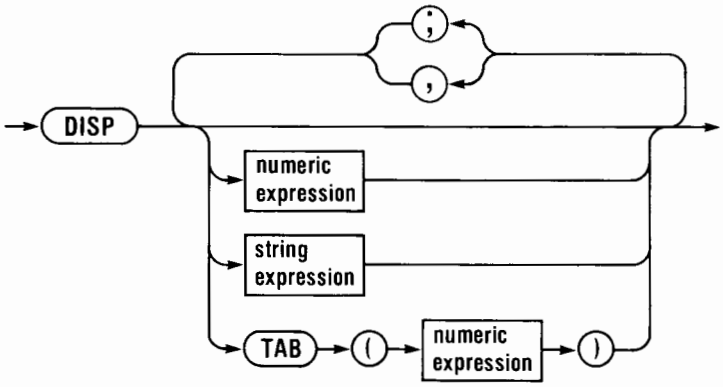

DEF FN
simple numeric variable
(
,
simple numeric variable
string variable
[
nonnegative integer
]
)
=
numeric expression
string variable
(
,
simple numeric variable
string variable
[
nonnegative integer
]
)
=
string expression

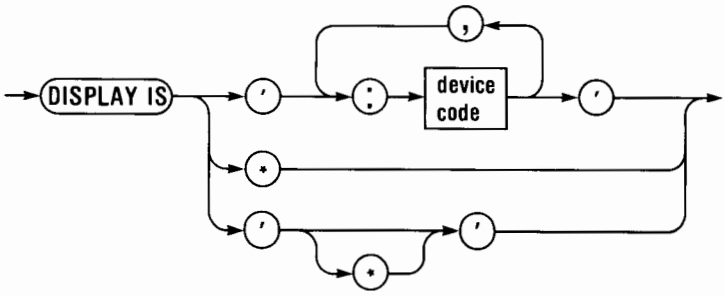

DEF KEY
'
key display character
'
,
'
key definition
'
;

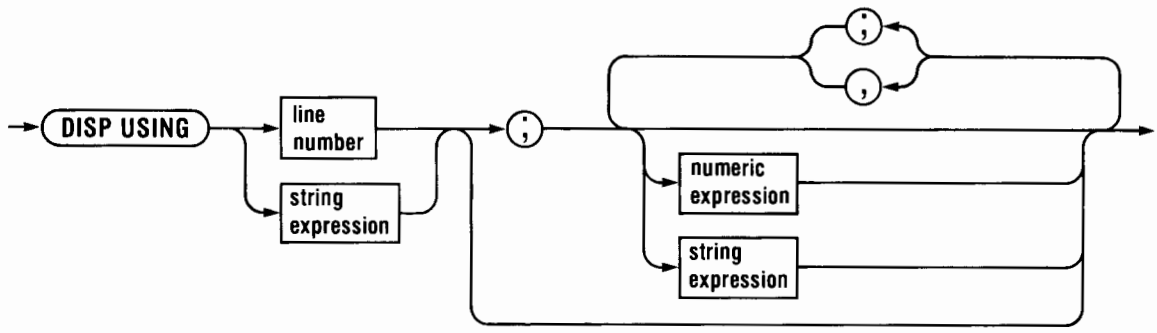
DELAY
numeric expression

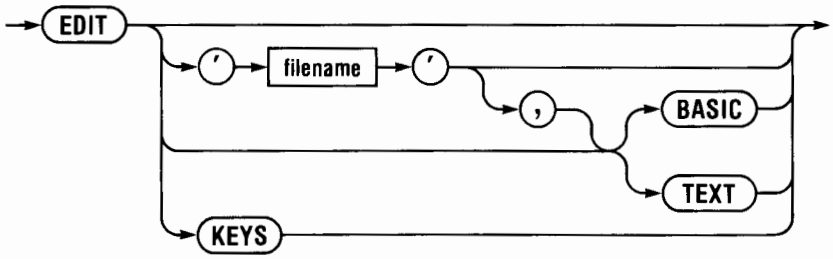

DELETE
line number
,
line number


DIM
,
string variable
[
nonnegative integer
]
numeric array variable
(
nonnegative integer
,
nonnegative integer
)


DISP
;
,
numeric expression
string expression
TAB
(
numeric expression
)

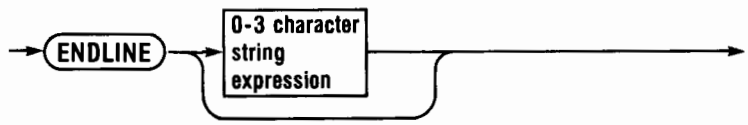

DISPLAY IS
'
,
:
device code
'
*
'
'
*

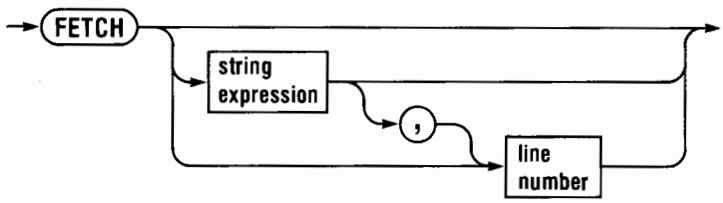

DISP USING
line number
string expression
;
;
,
numeric expression
string expression

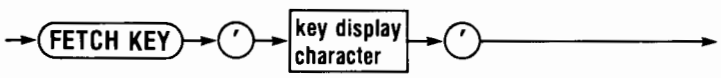

EDIT
'
filename
'
,
BASIC
TEXT
KEYS

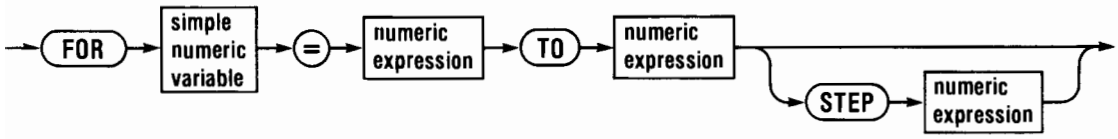
END

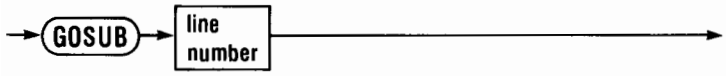
END DEF


ENDLINE
0-3 character string expression

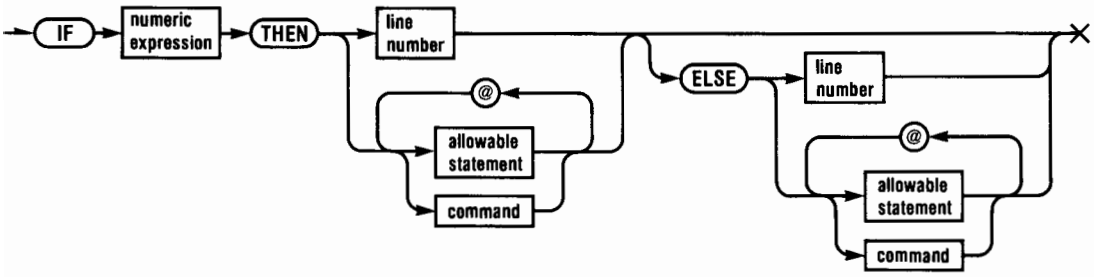

FETCH
string expression
,
line number


FETCH KEY
'
key display character
'


FOR
simple numeric variable
=
numeric expression
TO
numeric expression
STEP
numeric expression


GOSUB
line number

GOTO
line number


IF
numeric expression
THEN
line number
@
allowable statement
command
ELSE
line number
@
allowable statement
command

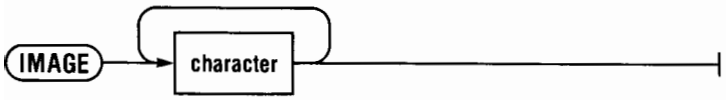
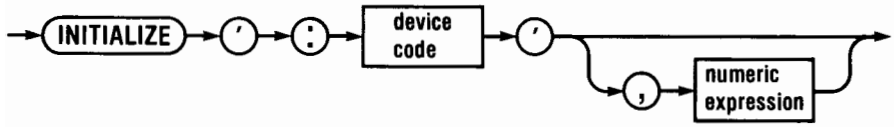
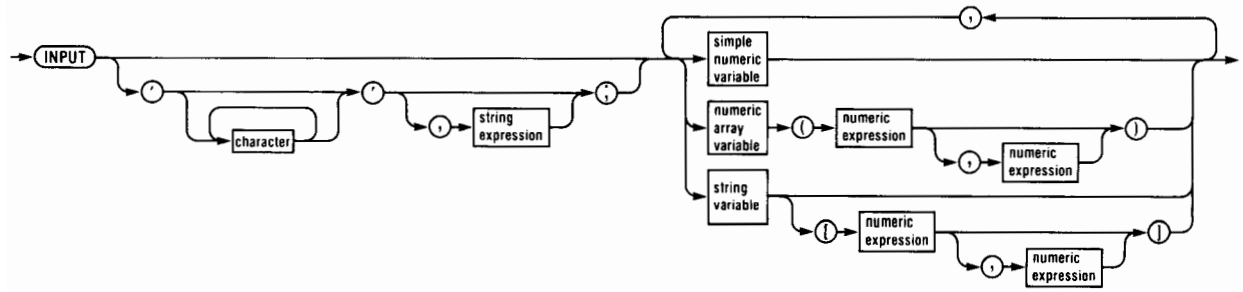
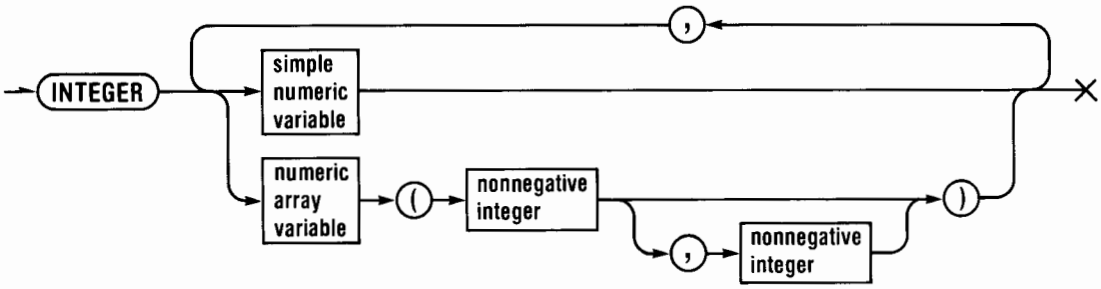
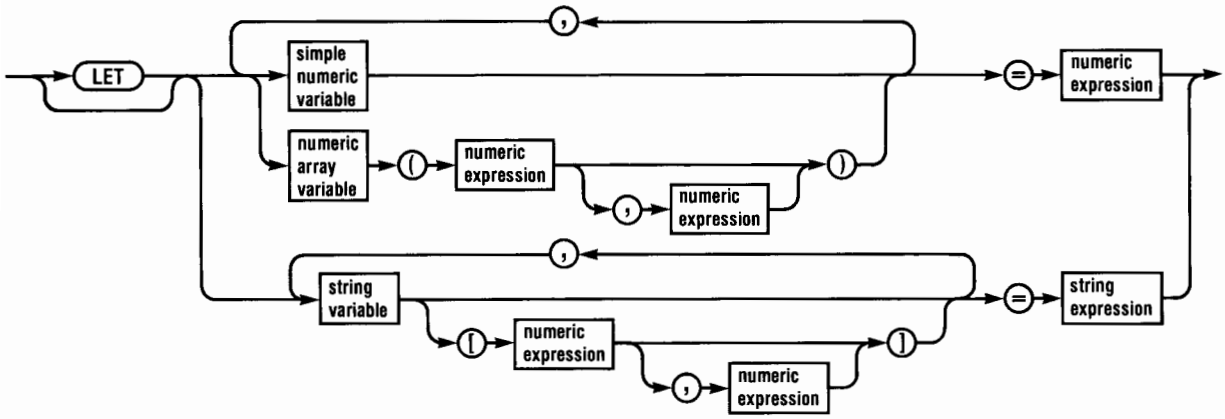


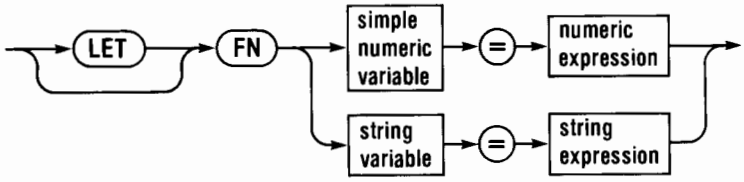
IMAGE
character

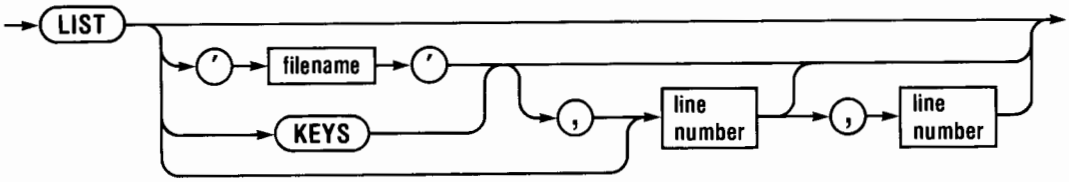

INITIALIZE
'
:
device code
'
,
numeric expression

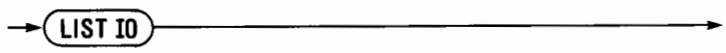

INPUT
'
character
'
,
string expression
;
simple numeric variable
numeric array variable
(
numeric expression
,
numeric expression
)
string variable
(
numeric expression
,
numeric expression
)
,


INTEGER
simple numeric variable
numeric array variable
(
nonnegative integer
,
nonnegative integer
)
,

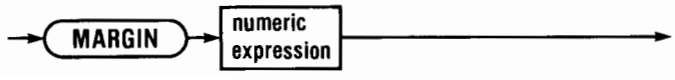

LET
simple numeric variable
numeric array variable
(
numeric expression
,
numeric expression
)
=
numeric expression
string variable
(
numeric expression
,
numeric expression
)
=
string expression

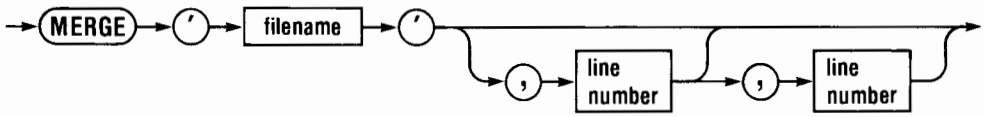

LET
FN
simple numeric variable
=
numeric expression
string variable
=
string expression

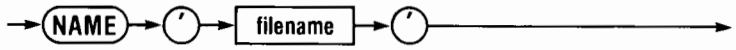

LIST
'
filename
'
KEYS
,
line number
,
line number


LIST IO

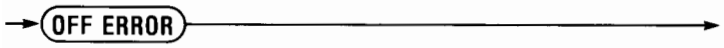
LOCK
string expression


MARGIN
numeric expression

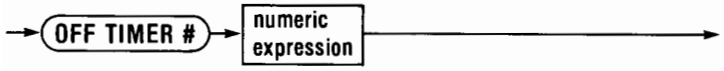

MERGE
'
filename
'
,
line number
,
line number

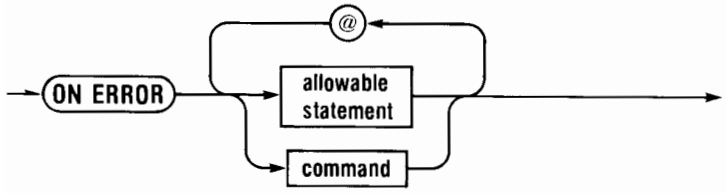

NAME
'
filename
'

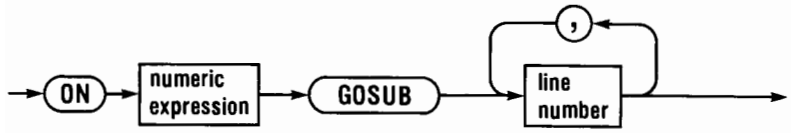
NEXT
simple numeric variable


OFF ERROR

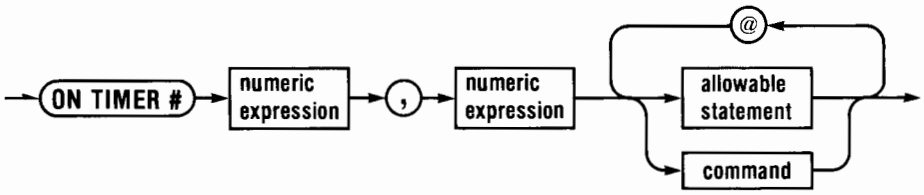
OFF IO


OFF TIMER #
numeric expression

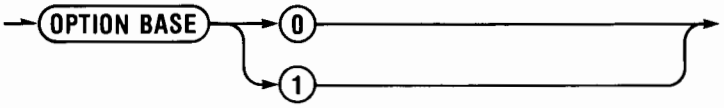

ON ERROR
@
allowable statement
command

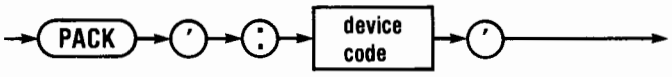

ON
numeric expression
GOSUB
,
line number

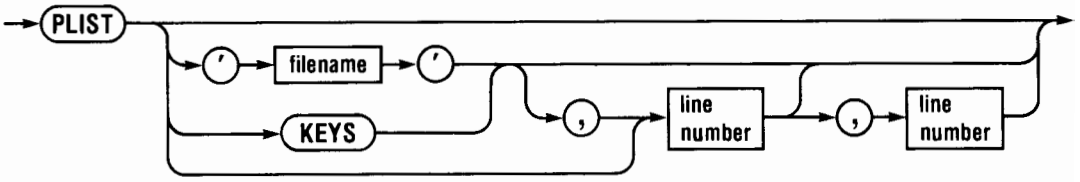
ON
numeric expression
GOTO
,
line number


ON TIMER #
numeric expression
,
numeric expression
@
allowable statement
command

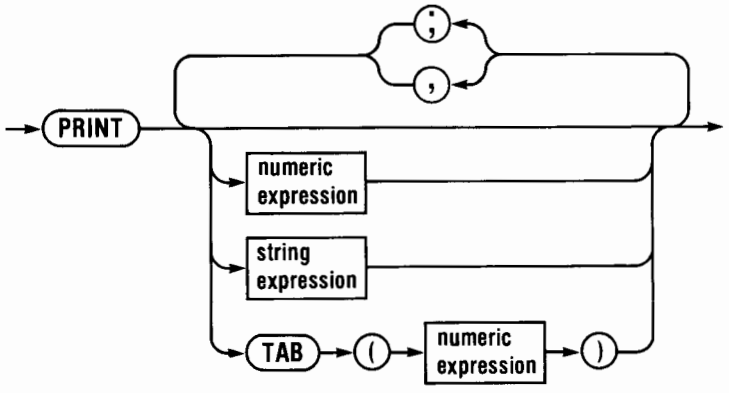
OPTION ANGLE
RADIANS
DEGREES

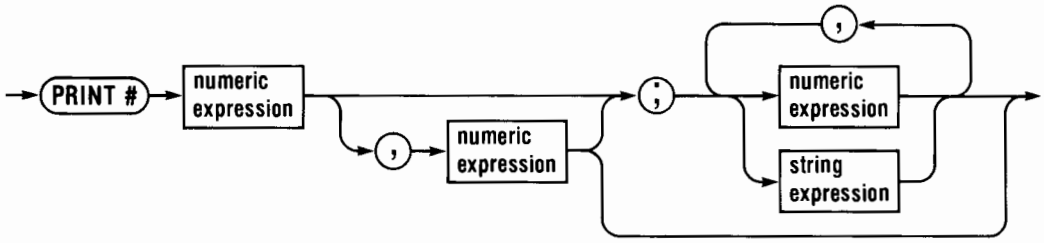

OPTION BASE
0
1

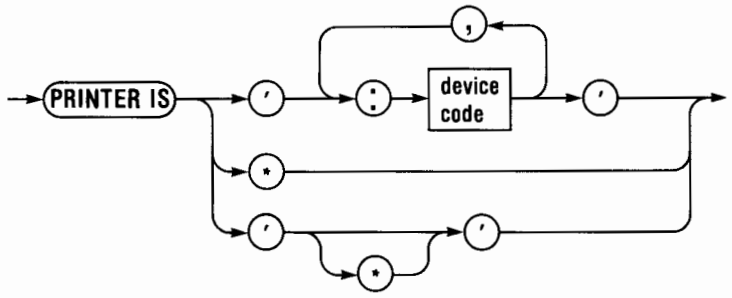

PACK
'
:
device code
'

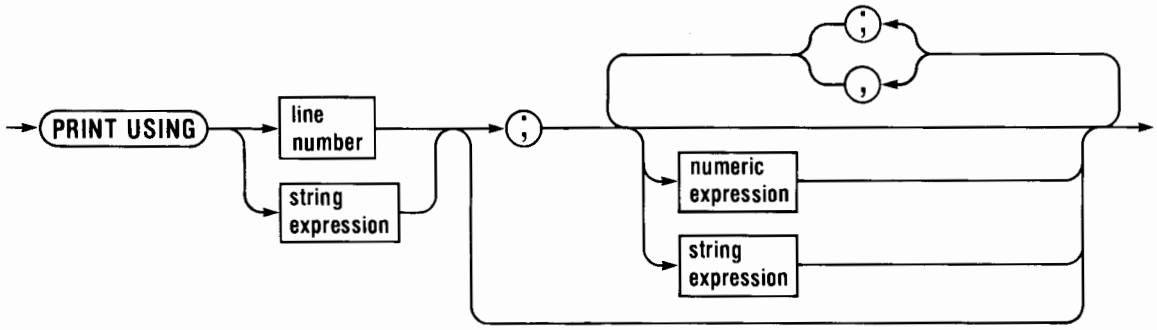

PLIST
'
filename
'
KEYS
,
line number
,
line number

POP

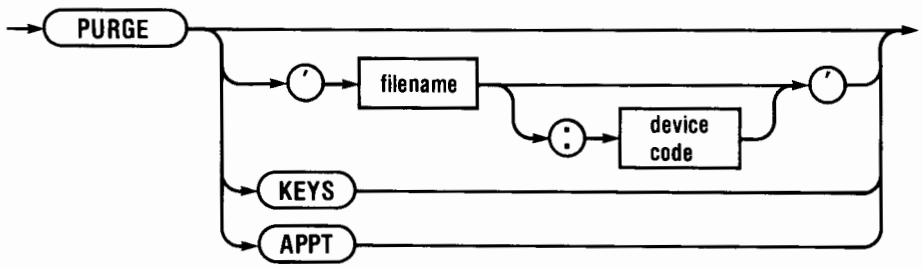

PRINT
;
,
numeric expression
string expression
TAB
(
numeric expression
)


PRINT #
numeric expression
,
numeric expression
;
,
numeric expression
string expression


PRINTER IS
'
:
device code
,
'
*
'
'
*

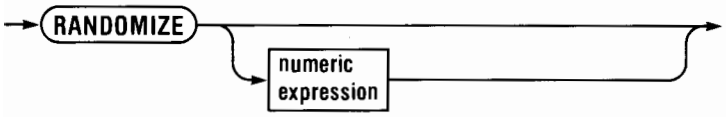

PRINT USING
line number
string expression
;
;
,
numeric expression
string expression

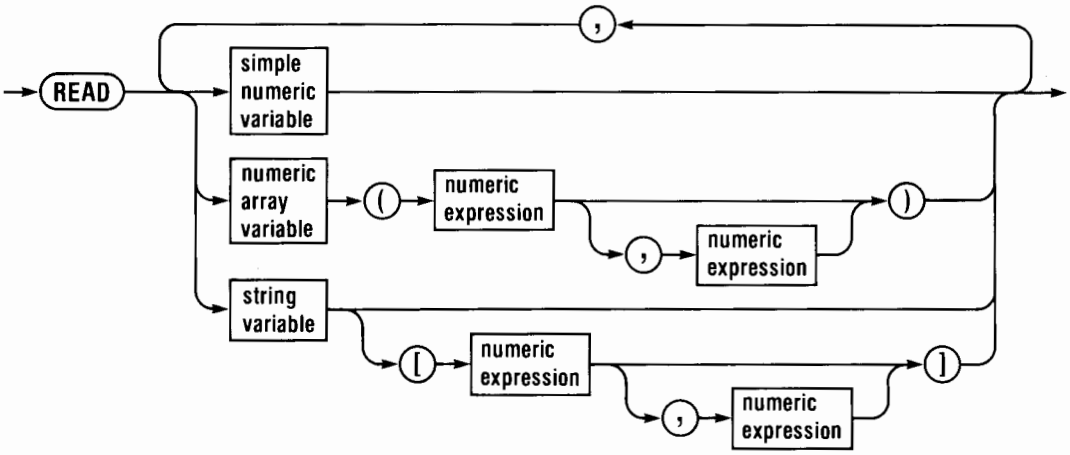
PROTECT

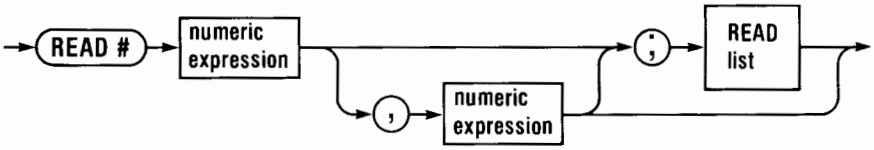

PURGE
'
filename
:
device code
'
KEYS
APPT

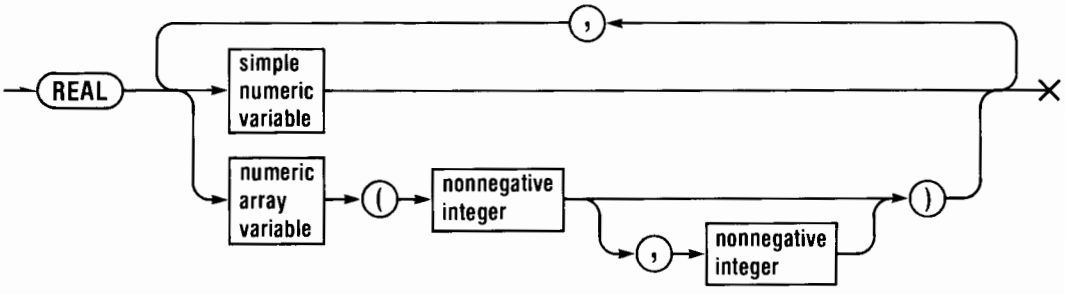
PUT
1 character string expression

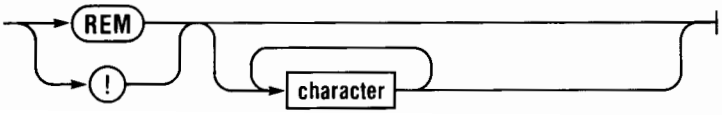
PWIDTH
numeric expression

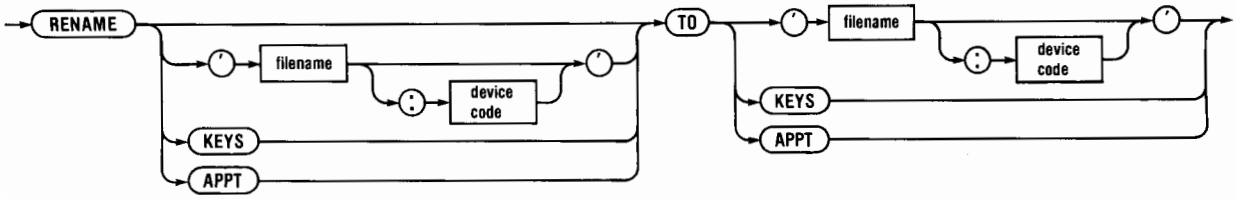

RANDOMIZE
numeric expression

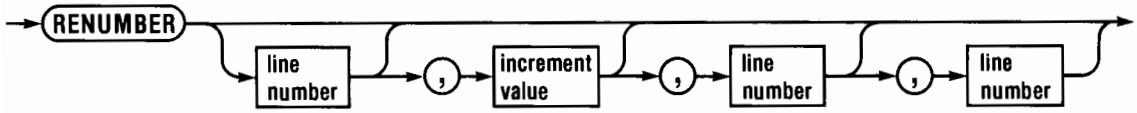

READ
,
simple numeric variable
numeric array variable
(
numeric expression
,
numeric expression
)
string variable
[
numeric expression
,
numeric expression
]

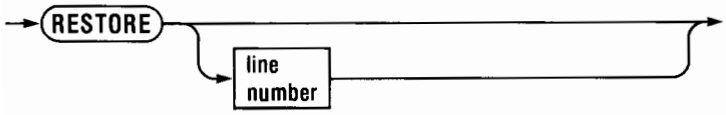

READ #
numeric expression
,
numeric expression
;
READ list


REAL
,
simple numeric variable
numeric array variable
(
nonnegative integer
,
nonnegative integer
)

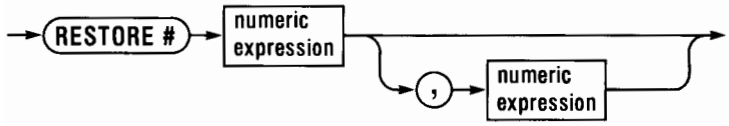

REM
!
character

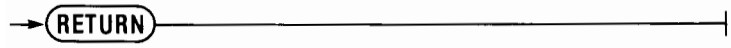

RENAME
'
filename
:
device code
'
KEYS
APPT
TO
'
filename
:
device code
'
KEYS
APPT

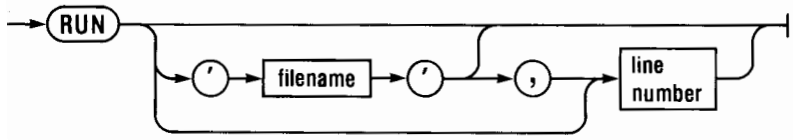

RENUMBER
line number
,
increment value
,
line number
,
line number

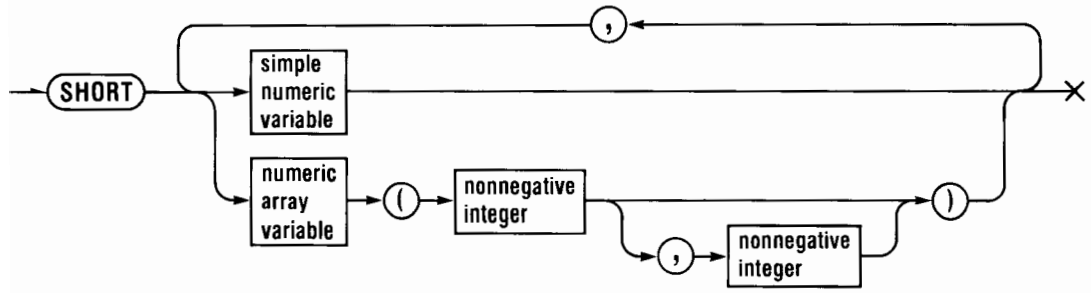

RESTORE
line number

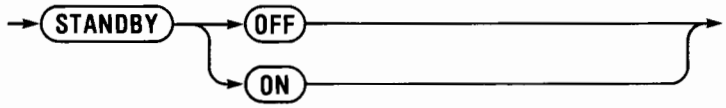
RESTORE IO

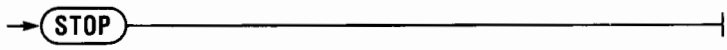

RESTORE #
numeric expression
,
numeric expression

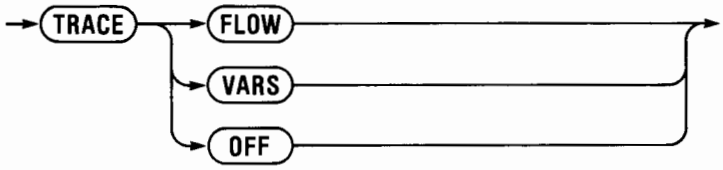

RETURN

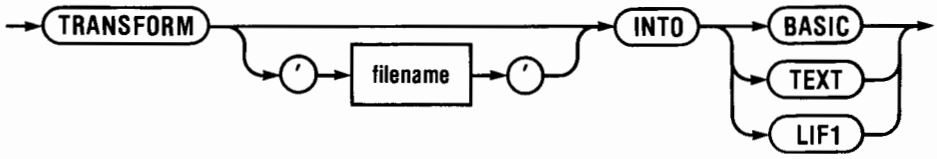

RUN
'
filename
'
,
line number

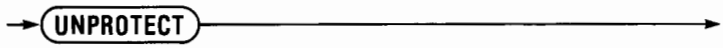

SHORT
,
simple numeric variable
numeric array variable
(
nonnegative integer
,
nonnegative integer
)


STANDBY
OFF
ON


STOP


TRACE
FLOW
VARS
OFF


TRANSFORM
'
filename
'
INTO
BASIC
TEXT
LIF1


UNPROTECT

WAIT
numeric expression

WIDTH
numeric expression

Subject Index

Page numbers in bold type indicate primary references; page numbers in standard type indicate secondary references. In addition to the references in this subject index, a complete index to the HP-75 instruction set—operators, functions, commands, and statements—is located inside the back cover of the owner's manual.

A

Abbreviations, keyword, **31**, **296**
ABS (*absolute value*) function, **82**, **332**
Absolute adjustment of clock, **97**
AC adapter/recharger, using, **11**
Accessing text files from programs, **224-228**
Accessories, standard, **264**
Acknowledging appointments, 17, **105**
ACOS (*arccosine*) function, **86**, **332**
Addition operator (+), **69**, 72
ADJST (TIME *adjust*) command, 93, **95-97**
Adjusting the clock, **95-97**
Alarm types, **101**
ALARM OFF/ON command, **106**
Altering flow of program control, **176**
Ampersand operator (&), **164-165**
AND operator, **88**, 330
ANGLE function, **86**, **332**
Annunciators, **14**
ANSI standards, conformance of interpreter to, **268-269**
Antilog
  Common, **85**
  Natural, **84**
Appending strings. See *concatenating strings*
Appointment
  commands, **108-109**
  commands in key definitions, **147**
  file, cataloging, **49**
  files, **46**
  files, purging, **104**
  memory requirements, **292**
  message/command indicator, **102**
  mode errors, **305**
  repeat interval, **107**
  Rep t template, **106**
  types, 102, **106**, 109
Appointments,
  Acknowledging, 17, **105**
  Copying, to and from mass storage, **112**
  Deleting individual, **104**
  Editing, **104**
  Repeating, **106-108**
  Scheduling, 16, **100-102**
  Self-scheduling, **106-108**
Appointment template, 16, 36, **100-102**, 107, 108
Appointment template, alternate, **109-110**
Appointment template fields, 16, **101**
  Alarm Field, **101**
  Command Field, 101, **108-109**
  Day-of-Week Field, **101**, 102, 107
  Note Field, 36, **101**
APPT (*appointment*) annunciator, 14, 100, **105**
appt file, 46, **102-104**
APPT (*appointment*) Key, 14, 16, 17, **100**, **285**
APPT mode, 14, 16, **100-112**
  Turning off, **106**
APPT template. See *appointment template*
Arccosine function (ACOS), **86**, **332**
Arcsine function (ASIN), **86**, **332**
Arctangent function (ATAN), **86**, **332**
Arguments, function, **82**, **332**
Arithmetic
  expressions, **68**
  hierarchy, **72**
  keyboard, 18, **68**
  operators, **69-193**
Array, **192**
  subscripts, **192-193**
  upper bound, **194**
Arrays,
  One-dimensional, **192-193**
  Two dimensional, **192-193**
Array variables, 78, **192-196**
  Assigning values to, **195-196**
  Declaring, **193-195**
  Dimensioning, **194**
  Initializing, **195-196**
  Naming, **193**
  Retrieving from and storing in data files, **223-224**
  Setting the lower bound of, **193**
ASCII, 41, **288**
ASIN (*arcsine*) function, **86**, **332**
ASSIGN # statement, **216-217**, 222-223, 224-225, 227, 232
Assigning device codes on the HP-IL loop, **126-127**
Assigning file numbers to data files, **126-127**
Assigning mass storage device codes, **132**
Assigning values to variables
  in calculator mode, **79**
  from the keyboard in programs, **168-169**
  in programs, **165**, **195-196**
ASSIGN IO command, **126-127**, **132**
Assignment statement (LET), **79**, **165**
ATN (*arctangent*) function, **86**, **332**
At (@) symbol, **147**, **163**, 179, **320**
ATTN (*attention*) key, 12, 13, **14**, 15, 159, 160-161
  Interrupting programs with, **159**
AUTO command, 25, **51-52**
Automatic line numbering, 25, **51-52**
Available memory function (MEM), **47**

B

BACK (*backspace*) key, 16, **35**, 36, **285**
Backspace character (CTL H), **43**, **289**
Base conversion program, **254-255**
BASIC
  files, 46, **159-160**
  interpreter, conformance to ANSI standards, **268-269**
  keywords, **162**
  predefined functions, **331-334**
  programming, 21, **156-157**, 162
  prompt (>), **17**

statements, **162**
statements, concatenating, **163-164**
syntax diagrams, **335-348**
syntax guidelines, **32**, **334**
BASIC statements, **162**
not allowed after THEN, **178**
not executable from the keyboard, **162**
BATT (*battery*) annunciator, **14**, **271**
Battery pack,
Care of, **271**
Rechargeable, 11, **270-271**
Recharging, **270-271**
Replacing, **271-272**
Warranty on, **271**
BEEP OFF/ON command, **30-31**
BEEP statement, **30**
Blank space specifier, image format string, **240**
Boolean expression, **88**, 177
Boolean value, 87, **88**
Branching out of a FOR-NEXT loop, **181**
Branching, program, **176-179**
Tracing, **252-253**
BYE command, 29, **174**
Bypassing a pending subroutine return (POP), **183-185**

C

Calculations, arithmetic, **69-71**
Calculator mode
arithmetic, **68-71**
effect on files, **69**
variables, **81**
Calendar,
Extended, **110-111**
Gregorian, **112**
Searches, **111-112**
Year, **110**
Calibrating the clock, **95-97**
CALL statement, **230-231**
Comparing, with RUN, **231**
Memory requirements of, **292**
Recursive, **234-236**
Cancelling
HP-IL display and printer assignments, **129**
ON ERROR declarations, **259-260**
trace operations (TRACE OFF), **253**
Card file
catalog, **117-118**
copying, 21-23, **118-120**
protection, **121-122**
specifiers, **116-117**
verification, 61, **115-116**
Card reader
care, **273**
commands, **116**
messages, 21-23, 60-61, **115-116**
operation, 21-23, 60-61, **114-122**
warnings, 22-23, 116, **300**
Cards, magnetic, 21-22, **114**
Cleaning, 115, **273**
Marking, **273**
Carriage-return character ([CTL][M]), **43**, 239, **289**
Carriage-return/line-feed, 43, **286**
CAT$ (*catalog*) function, 198, **202-203**, 236
CAT ALL command, **49**
Catalog
command, 47, **49**, 103, 117-118, 134, 144
entry, **47**, 103, 203
function (CAT$), 198, **202-203**
Cataloging
cards, **117-118**
mass storage medium, **134**
Catalog, system, **49**, 63
CAT APPT command, **49**, **103**
CAT CARD command, **117-118**, 172
CAT (*catalog*) command, 47, **49**, 103, 117-118, 144, 172
CAT KEYS command, 49, **144**
CEIL (*smallest integer*) function, **82**, **332**
Changing date format, **94**
Changing increment values in FOR-NEXT loops, **180-181**
Changing key definitions. See *redefining keys*
Character,
Backspace, **43**, 289
Carriage-return, **43**, 289
Escape, **42**, 138-139, 287, 289
Keystroke, display, **42**, **287**
Line-feed, **43**, 289
Null, **42**, 289
Character
codes, decimal, 41, **288-291**
set, 41, **288-291**
string, **196**
Characters,
Control, 28, **42-43**, 138-139, **288-291**
Converting, to decimal codes (NUM *function*), **41**
Display, **288**
Underlined, **41**, **288-291**
Checking a halted program, **257**
CHECK program, **157-158**
CHR$ function, **41**, 43, **288**
Cleaning the computer, **274**
Clearing
HP-IL device assignments, **130**
HP-IL loop devices to initial state, **131-132**
timers, **187**
variable values, **82**
CLEAR LOOP command, **131-132**
CLEAR VARS command, **82**
Clock
accuracy, **268**
adjustment (ADJST), **95-97**
functions, **98**
operation, **93**
Closing a data file, **219-220**
Codes
Character, 41, **288-291**
Device, **126-127**
Escape, **138-139**, **152-153**, 287, 294
Comma
delimiter, **239**
specifier in format string, **244**
Command appointments, **108-109**
Commands,
Debugging, **252-256**
Card reader, **116**
File, **172**
Global declaration, **233-234**
HP-IL, **126-140**
Local declaration, **233-234**
Non-programmable, **162**
System, **162-163**
System, syntax diagrams, **335-348**
That set a machine condition, **163**
TIME, **92-93**
Trigonometric, **85**
Comment delimiter (!), **166**
Comments, program, **166**
Common
antilog, **85**
logarithm (LOG10) function, **84**, **332**
Compact field specifier, image format string, **245-246**
Comparing
numeric expressions, **87-88**
RUN and CALL, **231**
string expressions, **203-204**
Computed
GOSUB, **183**
GOTO, **182-183**

Computer Museum

Concatenating
BASIC statements (&), **163-164**
strings (&), **165**, 196
Conditional branching, 176, **177-179**
Conditional test, **177**
Conformance of BASIC interpreter to ANSI standards, **268-269**
Connecting the HP-IL loop, **125-126**
Constant functions, **83**
CONT and RUN, comparing, **159**
CONT (*continue*) command, **159**, 172
Control characters, **42-43**, **288**
Control ([CTL]) key, 28, **285**
Controlling run-time errors, **252**
Converting
degrees to radians (RAD *function*), **86**
lowercase characters to uppercase (UPRC$ *function*), 198, **201**
numbers to strings (STR$ function), 198, **201**
radians to degrees (DEG *function*), **86**
rectangular to polar coordinates, **86-87**, 233
strings to numbers (VAL *function*), 198, **200**
COPY command, 22, **60-61**, **112**, 118-121, **135-136**, **148**, 172
Copying
appt file, **104**, 112
files from cards to memory, 22, **119-120**
files from memory to cards, 60-61, **118-119**
files to and from mass storage, **112**, **135-136**, **148**
a prerecorded program into memory, **21-23**
Correcting the clock, **95-97**
Cosecant (CSC) function, **86**, **332**
COS (*cosine*) function, **86**, **332**
COT (*cotangent*) function, **86**, **332**
Creating data files, **216-217**
Creating files, 47, 52-53, **62-63**
CSC (*cosecant*)function, **86**, **332**
[CTL][BACK] (*escape*) keystroke, **42-43**, 138-139, **287**, **289**
[CTL][FET] (*control fetch*) keystroke, **38**
[CTL][H] (*backspace*) keystroke, **43**, **289**
[CTL][J] (*line-feed*) keystroke, **43**, **289**
[CTL] (*control*) key, 28, **285**
[CTL][LOCK] (*enable numeric keypad*) keystroke, **28**
[CTL][M] (*carriage-return*) keystroke, **43**, **289**
[CTL][↑] (*control up-arrow*) keystroke, **138-139**, **287**
[CTL][↓] (*control down-arrow*) keystroke, **138-139**, **287**
[CTL][←] (*control left-arrow*) keystroke, **35**, 38, **287**
[CTL][→] (*control right-arrow*) keystroke, **35**, 38, **287**
Current file, **47**, 63
Current line. See *pending file*
Cursor,
Addressing of, **294**
Backspacing the, **35**
Controlling the, **138-139**, 149-150, **294**
Insert (▮), **36**
Replace (▮), **36**

D

Data files, 210, **216-222**
Creating, **216-217**
Closing, **219-220**
Opening, **216-217**
Random access, **220-221**, 222-223
Reading data from, **219**
Serial access, **220**
Storing data in, **217-219**
Data items, **212**
memory requirements, **292**
Data lines,
Deleting, **221**
Long, **228**
Data pointers, **212**, 213
Memory requirements of, **292**
Moving, **213**, **221-223**
Data, rereading, **213**, **221-222**
DATA statement, **210-211**, 212, 213, 218-222, 223-224
Data track, **114**
DATE function, **98**
DATE$ function, **98**
Day-date searches, **111-112**
Deallocating programs, **159**
Debugging programs, **252-260**
Decimal codes, character, **41**, **288-291**
Converting to character (CHR$ *function*), **41**
Decimal point specifier, **243**
Decision making statements, **177-179**
Declarations, global and local, **233-234**, **293**
Declaring
arrays, **193-194**
display and print devices, **128**
variable types, **80-81**, **194-195**
Default input prompt (?), **168**
DEFAULT OFF/ON command, **89-90**
Defaults, machine, **295**
Default values, math error, **89**, **299**
DEF FN (*define function*) statement, **205**, **207**
Definitions, key. See *key redefinitions*
DEF KEY command, 25, **143-144**, **145-148**, 152
DEG (*radians-to-degrees*) function, **86**, **332**
Degrees mode, **85**
DELAY command, **39**, **129**, 160
Delaying program execution (WAIT *statement*), **166**
Deleting
appointments, **104**
characters, **35**
data lines, **221**
files. See *files, purging*
lines in files (DELETE *command*), **59**, 172
Delimiters, in image format strings, **239**
[DEL] (*delete character*) key, **35**, 285
Deviations from Minimal BASIC, HP-75, **269**
Device codes, **126-127**
Difference between CONT and RUN, **159**
Digital cassette drive (*HP 82161A*), **132**
Digit separator specifiers, image format string, **244-245**
Digit specifiers, image format string, **241-243**
Dimensioning arrays (DIM *statement*), **194**
Dimensioning string variables, **194**
Display
annunciators, **14**
character keystroke, **42**, **287**
characters, 28, 42, **288**
clearing, **51**
echoing, **36-37**
escape codes, **138-139**, **294**
formating (IMAGE), **238-239**
line, **14**
line length, **39**
of numbers, **36**
templates, examples of, **6-7**
DISPLAY IS command, **128-129**
DISPLAY IS device escape codes, **138-139**, **294**
Displaying information, 36-40, **129-130**, **166-168**, **238-250**
Displaying values of variables, **80**
Display window, **14**
DISP statement, **129-130**, **166-167**
DISP USING list, **239**
DISP USING statement, **238-239**, 247
Format string in, **247-248**
Division operator (/), **70**, 72
DIV (*integer division*) operator, **70-71**
Documenting programs, **166**, 195
Down-arrow ([↓]) key, **53**, **285**

Due appointments, acknowledging, **105**
Duplicating files
  in mass storage, **136**
  in memory, **61-62**

E

Echoing, display, **37-39**
EDIT command, 25, **48**, **62**, 172
Editing
  appointments, **104**
  files, 25-27, **47-49**, **62-64**
  keys and keystrokes, **285**, **286-287**
  keys files, **151-152**
  operations, 46, **53-59**, 62
  programs, **156-157**
  text, **25-27**, **50**
[EDIT] key, 14, 17, **46**, 50, **285**
EDIT Mode, 15, 17, 25, **46**
  Operations in, 25, 46, **53-59**, **62-64**
  Typing in, 19, **25**
EDIT prompt, **12**
Electrostatic discharge, CAUTIONS, **267**
ELSE keyword, **178**
END DEF (*end function definition*) statement, **207**
ENDLINE command, **140**
End-of-line sequences, **140**
END statement, **165-166**
Environmental limits, **267**
EPS (*epsilon*) function, **76**, **83**, **332**
ERRL (*error line*) function, **260-261**
ERRN (*error number*) function, **20**, 53, **260-261**
Error
  conditions, recovery from, **19**, 20, 53, **89**, **298-306**
  line function, (ERRL), **260-261**
  messages, alphabetical listing with numbers, **307**
  messages, viewing, **20**
  number function (ERRN), **20**, 53, **260-261**
  numbers, messages, and conditions, table of, **298-306**
  numbers, viewing, **20**
  processing, run-time, **258-261**
  recovery subroutines, **259**
  tracing, **252-256**
ERROR annunciator, 14, 19, 52, **298**
Errors,
  APPT Mode, **305**
  Card reader, **300**
  File and device, **300-301**
  HP-IL, 131, **303**
  Initialization, 174, **252**
  Logical, 174, **252**
  Mathematical, **89-90**, **299**
  Mass memory, **306**
  Program, **301-302**
  Run-time, 174, **252**
  Syntax, **52-53**, 174, **252**, **305-306**
  System, **300**
  TIME Mode, **304**
Equals operator (=), **87**, 203
Escape character ([CTL][BACK]), **42**, 138-139, **287**
Escape codes, **138-139**, 287
  Display, **294**
  in key definitions, **152-153**
E (*exponent*) specifier, image format string, **245**
EXACT command, 93, **95-97**
Examining programs, **159-161**
Executing programs, **158**
EXOR (*exclusive or*) operator, **88**, 330
EXP (*natural antilog*) function, **84**, **332**
Exponent, **75**
Exponential notation (E), **74-75**
Exponentiation operator (^), **70**, 72
Exponent symbol specifier, image format string, **245**
Expression,
  Arithmetic, **68**
  Evaluation of, **37**, **68-69**
  Multiple, evaluation of, **37**
  Numeric, 18, **87**
  String, 152, **196**, **203-204**
EXTD command, **110-111**
Extended calendar, **110-111**
Extended day-date search, **94**, **111-112**
Extensions to Minimal BASIC, HP-75, **268**

F

FETCH command, **53-54**, **146-147**, 161, 172
Fetching key definitions, **146-147**
[FET] (*fetch*) key, 48, **54**, 160
Field specifiers. See *image specifiers*
File command summary, **65**
File commands, using with allocated programs, **172-173**
File editing, **47-59**
  Deleting lines, **59**
  Fetching lines, **53-54**
  Inserting lines, **55**
  Moving lines, **55-56**
  Revising lines, **54-55**
  Stepping through lines, **53**
File errors, **303-304**
File manipulations
  Copying, **60**, **112**, **118-120**, **135-136**, **148**
  Duplicating, **61-62**, **136**
  Listing, 26, **56-57**
  Locating, **62-63**
  Merging, **60**, 172
  Naming, **64**
  Printing, **56-57**
  Purging, **50**, 137, 144
  Renaming, 21, **59**, 104, 137, 147-148
  Renumbering, **57-58**
Filenames, **45**, 66
File numbers, **217-223**
File pointer, **48**, 49, 52, 53, 62, 160-161
Files,
  Accessing text, from programs, **224-228**
  Copying, 60, **112**, **118-120**, **135-136**, 148
  Creating and editing, **47**, 50-52, 62-63
  Location of, in memory, **45**, 63
  Memory requirements of, **292**
  Special, **274-278**
Files, data, **210**, 216-222
  Creating, **216-217**
  Closing, **219-220**
  Opening, **216-217**
  Random access of, **220-221**, 222-223
  Reading data from, **219**
  Serial access of, **220**
  Storing Data in, **217-219**
File specifiers,
  Card, **116-117**
  Mass storage, **133-134**
File types
  Appointment, 46, **102-104**, 112
  BASIC, **46**
  Card, **116-117**
  Data, 210, **216-222**
  Keys, 46, **144** 147-148, 151-152
  LEX (*language extension*), 46, 274, **277**
  LIF1, 46, **274-277**
  Mass storage, **133-134**
  Other (?), **278**
  Private BASIC (PB), 46, 65, **117**
  ROM, 274, **277-278**
  Text, **25**, 46, 51, 224-228
FINDIT progrm, **227-228**
FIRST and SECOND programs, **231-232**

Floating point format, **73-74**
FLOOR function, **82, 332**
Format strings, **239**
  Including in PRINT/DISP USING statements, **247-248**
  Reusing, **246-247**
Formatting output
  with IMAGE, **238-239**
  with PRINT/DISP statements, **129-130, 166-167**
  with PRINT/DISP USING, **247-248**
  with TAB function, **167**
FOR-NEXT statements, **179-181**, 195-196
  Executing, from the keyboard, **180**
  Nested, **181**
  STEP option, **180**
FP (*fractional part*) function, **82, 332**
Function assignment statement, **207**
Functions,
  Arguments of, **82**
  Clock, **98**
  Constant, **83**
  General numeric, **83, 332-333**
  Key, **284-286**
  Logarithmic, **84, 332**
  Number alteration, **82-83, 322-333**
  Numeric, **82-86, 332-333**
  String, **198-203, 333**
  Trigonometric, **85, 332-333**
Functions, user-defined, **205-209**
  Definitions of, **206**
  Milti-line, **207**
  Names of, **205**
  Numeric, **205**
  Parameters of, **205, 206, 208**
  Single line, **205**
  String, **205**, 206, 207

G

Global declarations, **233-234, 293**
Global variables, **206**
Glossary, **320-329**
GOSUB, computed, **183**, 188-189
GOSUB statement, **182**
GOTO, computed, **182-183**
GOTO statement, **176-177**, 188
Greater-than operator (>), **87**, 203
Greatest integer (FLOOR, INT) functions, **82, 332**
Guidelines, syntax, **32, 334**

H

Halted programs, debugging, **257**
Halting program execution, **165-166**
  with [ATTN] key, **159**
  with END statement, **165-166**
  with STOP statement, **165-166**
Hewlett-Packard Interface Loop (*HP-IL*), **124-125**
Hierarchy, arithmetic, **72**
HP-IL errors, **303**
HP-IL mass storage
  Assigning mass storage devices, **132**
  Cataloging the medium, **134**
  Copying files to and from mass storage, **135-136**
  Duplicating files in mass storage, **136**
  Initializing the medium, **132-133**
  Packing the medium, **137-138**
  Purging mass storage files, **137**
  Renaming mass storage files, **137**
  Specifying mass storage files, **133-134**
HP-IL memory requirements, **292**
HP-IL operations
  Assigning device codes, **126-127**
  Canceling display and printer device assignments, **129**
  Clearing device assignments, **130**
  Clearing loop devices to initial status, **131-132**
  Connecting the loop, **125-126**
  Declaring display and print devices, **128**
  Displaying and printing information, **129-130**, 140
  DISPLAY IS device control codes, **138-139**
  Listing device assignments, **127**
  Transmission interruptions, **130-131**
  Turning the loop off and on, **130**
Humidity limits (*of computer*), **267**

I-J

Idle-loop, **202**
IF...THEN statement, **177-179**
  ELSE option in, **178**
  Executing, from the keyboard, **178**
  Multiple instructions after, **179**
Image format strings, **239**
Image format summary, **249-250**
Image specifiers,
  Blank space (x), **240**
  Character (a), **241**
  Compact field, (k), **245-246**
  Digit separator (c, p), **244-245**
  Exponent (e), **245**
  Invalid, 243, **247**
  Numeric (d, z, *), **241-242**
  Radix (., r), **243**
  Replication, **246**
  Sign (s, m), **243-244**
IMAGE statement, **238**, 239-250
Immediate-execute key redefinitions, **145-146**
Incorrectly nested loops, **181**
Indirect recursion, **235**
INF (*machine infinity*) function, **76, 83, 332**
Initialization errors, 174, **252**
INITIALIZE command, **132-133**
Initialized programs, determining size, **292**
Initializing the mass storage medium, **132-133**
Initializing programs, **158**
Initializing variables, **158**
Input buffer, **38**
Input data precision, **73**
Input prompt (?), **168**
  string expression, **169-170**
INPUT statement, **168-170**, 195-196
Insert cursor (↟), **36**
Inserting lines in files, **55**
Insert/replace ([I/R]) key, **36, 287**
Integer division operator (DIV or \), **70-71**, 72
INTEGER statement, 80-81, **194-195**
Interchange (*LIF1*) files, **274-277**
Interrupting programs, **159**
INT (*greatest integer*) function, **82, 332**
IP (*integer part*) function, **82, 332**

K

KEY$ function, 198, **202**
Keyboard, **34**
  arithmetic, 18, 51, **68-70**, 72
  control of programs (KEY$ *function*), 198, **202**
  illustration, **6**
  operations, **284-287**
  overlays, **28**, 264
  shifted, **27**
Keypad, numeric, **28**, 69
Key redefinitions, 25, **142-153**
  Data input, **153**
  Editing and system keys, **148-149**
  Fetching, **146-147**
  FOR-NEXT loops in, **180**
  Functions for, **284-286**

Immediate-execute, **145-146**
Memory requirements of, **292**
Multiple command, **147**
TIME and APPT Mode command, **147**
Typing aid, **143-144**
Undoing, **144-145**
Keys,
Carriage-return/line-feed, **43, 286**
Editing, 6, **35, 285**
Erasure, **35-36**, 285
Immediate execute, **145-146**
Modifier, 27, **285**
Redefining, 25, **142-153**
Repeating, **19**
System, 6, 43, **285-286**
Typewriter, **284-285**
Keys files, **46**, 144, **147-148, 151-152**
cataloging, 49, **144**
copying, **148**
editing, **151-152**
multiple, **147-148**
Keystroke combinations, **286-287**
Display character, **42, 287**
Display device, **287**
Editing, **286-287**
Escape, **287**
System, **286**
Key waiting buffer, **204**
Keyword
abbreviations, **31, 296**
definition, **162**

L

Language extension files (*LEX files*), 46, 274, **277**
Larger of two values (MAX) function, **83, 332**
Largest machine number (INF *function*), **76**
Last result function (RES), **71**
Left-arrow key (←), **35**, 38, **285**
LEN (*string length*) function, 198, **199**
Length of a string variable, 194, **199**
Lesser of two values (MIN) function, **83, 332**
Less-than operator (<), **87**, 203
LET FN (*function assignment*) statement, **207**
LET (*assignment*) statement, **79**, 195-196
LEX (*language extension files*), 46, 274, **277**
LIF1 (*logical interchange files*), 46, **274-277**
Line-feed character (CTL J), **43, 289**
LINEFEED program, **43**
Line length, setting of, **39**
Line numbering, automatic, 25, **51-52**
Line numbers, **50**, 55, **162**
Lines in files,
Deleting, **59**
Entering, **26**, 50
Inserting, **55**
Listing, 26, **56-57**, 159-160
Moving, **55-56**
Printing, **56-57**
Recalling, **53-54**
Renumbering, **57-58**
Revising, **54-55**
Lines, multistatement (&), **163-164**
Line width, controlling, **39-40**
LIST command, **56-57, 159-160**
Listing
HP-IL device assignments, **127**
lines in files (LIST *command*), **56-57, 159-160**
programs, **159-160**
LIST IO command, **127**
Literal string, **196**
Local declarations, **233-234, 293**
Local variables, **206**
Locating files, **62**
LOCK command, **28-29**, 174
Locking the HP-75 against use by others, 28, **174**
Logarithmic functions, **84, 332**
LOG (*natural logarithm*) function, **84, 332**
Logical
errors, 174, **252**
interchange files (*LIF1 files*), 46, **274-277**
operators, **88**
LOG10 (*common logarithm*) function, **84, 332**
Long data lines, **228**
Long lines in programs, **163-164**
Loop counter, **179**
Looping, program, 176, **179-181**
Loop interruptions. See *transmission interruptions*
Loops, nested, **181**
Low battery
annunciator, **14, 270**
safeguards, **270**
Lower bound of arrays, setting, **193**

M

Machine defaults, **295**
Machine infinity (INF) function, **83, 332**
Magnetic card. See also *card reader*
CAUTIONS, **21**
Cleaning, **115, 273**
Using, 22, **114**
Main program variables, **182**, 206, **208**
MANAGER program, **173**
Manipulating strings, **197-198**
Mantissa, **75**
MARGIN command, **40**
Mass memory errors, **306**
Mass storage devices, **132**
Memory requirements of, **292**
Mass storage files, **133-134**
Mathematical errors, **299**
Recovering from, **89-90**
Matrix, **192**
MAX function, **83, 332**
MEM (*available memory*) function, **47**
Memory requirements, system, **47, 292**
Merging files (MERGE *command*), **60**, 172
Message/command indicator, **102**
MIN function, **83, 332**
Minimal BASIC, conformance of HP-75 interpreter to, **268-269**
MOD (*modulo*) function, **83, 333**
Modifying string variables, **197-198**
MONEY program, **23, 308-309**
Moving
across the display, **35**
the data pointer, **221-223**
lines in files, **55**
Multiline user defined functions, **207**
Multiple arithmetic operations, **71**, 72
Multiple variable assignments, **80**
Multiplication operator (*), **70**, 72
Multistatement lines (&), **163-164**
Mutual recursion, **235-236**

N

NAMELIST program, **214-216, 315-318**
Naming files (NAME *command*), **64**, 172
Natural antilog (EXP) function, **84, 332**
Natural logarithm (LOG) function, **84, 332**
Nested
FOR loops, **181**, 196
subroutines, **182**
timers, **189-190**
NEXT statement, **179-180**
Normal adjustment of clock, 95, **97**
Not-equal to operator (≠ or <>), **87**, 203
NOT operator, **88**, 330

Null string, 145, **198**
Numbers,
  Displaying, 37, **73-75**
  Floating point, **73-74**
  Formatting, **73**
  Range of, **76**
Numeric
  array variables, 78, **331**
  expressions, 18, **87**
  field overflow (*image format*), **247**
  function names (*user defined*), **205**
  functions, **82-86**, **332-333**
  keypad, **28**
  overflow (*value*), **76**
  precision, **73**, 80, **331**
  specifiers, image format string, **241-245**
  underflow, **76**
  variables, **331**
NUM function, **41**, **288**

O

OFF IO command, **130**
OFF TIMER statement, **187**
One-dimensional array, **192**
ON ERROR...GOSUB/GOTO statements, **259**
ON ERROR RETURN, **260**
ON...GOSUB/GOTO statements, **182-183**
ON/OFF ERROR statements, **258-260**
ON TIMER...GOSUB/GOTO statements, **188-189**
ON TIMER statement, **186-187**
Opening data files, **216-217**
Operating
  computer with battery removed, **11**
  limits, **267**
  modes, **14**
  precautions, **267**
  system version (VER$), **267**
Operation, verifying proper, **278-279**
Operators,
  Arithmetic, **69-76**
  Logical, **88**
  Precedence of, **89**, **330**
  Relational, **87-88**
  String, **330-331**
OPTION ANGLE DEGREES/RADIANS command, **85**
OPTION BASE statement, **193**
OR operator, **88**, 330
Other type files (?), **278**
Overflow,
  Image format, **247**
  Numeric value, **76**
Owner's Pac program listings, **308-318**

P

Packing the mass storage medium (PACK *command*), **137-138**
Parameters,
  Function, **205**, 206, 208-209
  Statement, **162**
Parentheses
  enclosing array subscripts, **195**
  in numeric expressions, **72-73**, 89
Pass-by-value, **206**
Passing values between programs, **232-233**
Passwords
  for copying files, **117**, **133-134**
  for LOCK command, **28-29**
PAYATTN program, **170-172**, **310-314**
Pending line, **48**
PI function, **83**, **333**
PLIST command, 26, **56-57**, **159-160**
Plug-in modules, **274**
Pointer,
  Data, **212**
  File, **48**, **160-161**
Polar Coordinates program, **233**
POP statement, **183-185**
POS (*position*) function, 198, **200**
Potential for radio and television interference, **283**
Power
  consumption, **269**
  off, **13**, 28
  on, **13**
  supply information, **269-272**
Precedence of operators, **89**, **330**
Precision, numeric, **73**, 80, **331**
PRGM (*program*) annunciator, 14, **21**
PRINT # statement, **217-219**, 221, 222-223, 224-228, 232
PRINTALL program, **203-204**
PRINTER IS command, **128**
Printing files, **56-57**
PRINT USING statement, **238-239**
  Formatting in, **247-248**
PRINT statement, **129-130**, **167**
Private card files (PCRD), **117**
Processing run-time errors, **258-260**
Product information, **283**
Program
  branching, 176, **177-179**
  debugging, **252-260**
  definitions, **162-163**
  editing, **156-157**, **161**
  errors, 174, **252**, **301-302**
  looping, 176, **179-181**
  remarks, **166**
  subroutines, 176, **182**, **183-185**
  timer interrupts, 176, **186-190**, 234, 257
Programming and applications assistance, **283**
Program pointer, **160-161**
Program remarks, **166**
Programs,
  Calling (CALL *statement*), **230-231**, 257
  Checking, **257**
  Continuing, **159**
  Deallocating, **159**
  Examining, **159-161**
  Executing, **158**
  Halting, **165-166**
  Initializing, **158**
  Inputing data into, **168-169**
  Interrupting, **186-190**, 257
  Listing, **159-161**
  Output from, **166-168**
  Passing values between, **232-233**
  Prerecorded, copying to memory, **21-22**
  Running, 21, 23, **158**
  Single-step execution of, **256-257**
  Suspending operation of, **166**
  Tracing execution of, **252-256**
  Writing, 21, **156-157**
Program variables, 81, **158**
  Assigning values to, **165**
  Assigning values from the keyboard to, **168-169**
  Definition of, **164**
  Tracing (TRACE VARS), **252-253**
Prompt,
  BASIC, **17**
  Input, **168-169**
  String expression, **169-170**
  Text, **18**
  Variable, **170**
Proper operation, verifying, **278-279**
PROTECT command, **121-122**
Protected prompt in input statement, **169**
Purging files (PURGE *command*), 17, **50**, 104, 137, 172

PUT statement, **204-205**
PWIDTH command, **39-40**, 130, 160, 167-168

Q

Quotation marks (' ', " ")
  as string delimiters, **19**, 37-38, **240-241**
  in key definitions, **145**
Quoted prompt, **169**
Quoted string, **196**

R

RAD (*degrees-to-radians*) function, **86**, **333**
Radians mode, **85**
Radio and television interference, potential for, **283**
Radix symbol specifiers, image format string, **243**
Random
  access to data files, **220-221**, 222-223
  number (RND) function, **83**, **333**
  number seed, **83-84**
  number sequence, **83-84**, 234
  PRINT # and READ #, **221-222**
RANDOMIZE statement, **83-84**, 234
Range of numbers, **76**
READ # statement, **219-221**, 222-223, 224-228, 232
Reading data
  from a file, **219-221**, 222-228
  within a program, **210-213**
READ statement, **211-213**
REAL statement, 80-81, **194-195**
Reassigning values to variables, **80**
Recalling
  lines to the display, **38**, 48
  calculator expressions, **71**
Rechargeable battery,
  Care of, **271**
  Life of, **269**
  Recharging the, **11**, **270-271**
  Replacing the, **271-272**
  WARNING for, **270**
Recovering from mathematical errors, **89-90**
Recovering from program run-time errors, **259**
Recursion, **234-236**
Redefining
  [ATTN] key, cautions, **152**
  editing and system keys, **148-149**
  [SHIFT] [ATTN] keystroke, **149**
Redefining keys, 25, **142-153**
  to accept input, **153**
  to control the cursor, **149-150**
  as immediate execute keys, **145-146**
  for TIME and APPT Mode, **147**
  as typing aids, **143-144**
Relational operators, **87-88**, 177
Relative adjustment of clock, **95**
Remainder (RMD) function, **83**, **333**
REM (*remark*) statement, **166**
Renaming files (RENAME *command*), 21, **59**, 104, 137, 147-148, 173
Renumbering lines in files (RENUMBER *command*), **57-58**, 173
Repair services. See *service*
Repeat interval, appointment, **107**
Replication of specifiers in image statements, **246**
Rept template, appointment, **106**
Rereading data in programs (RESTORE *statement*), **213**
RESET (*clock*) command, **97**
Reseting the computer, **13**, **286**
RES (*last result*) function, **71**
RESTORE # statement, **221-222**
RESTORE IO command, **130**
RESTORE statement, **213**
RETURN statement, **182**, **183-184**
Reusing image format strings, **246-247**
Revising lines in files, **54-55**
Right-arrow key ([→]), **35**, 38, **285**
Right margin, setting of, **40**
RMD (*remainder*) function, **83**, **333**
RND (*random number*) function, **83**, 234, **333**
ROM
  files, **277-278**
  modules, **274**
ROOTS program, **207-208**
Rounding of displayed values, **73**
[RTN] (*return*) key, **14**, **50-51**
RUN and CONT, comparing, **159**
RUN command, **158**, 173
[RUN] key, **158**
Run-time errors, 174, **252**
  Processing, **258-260**

S

Scheduling appointments, 16, **100-102**
Scientific notation. See *exponential notation*
SEC (*secant*) function, **86**, **333**
Seed, random number, **83-84**
Self-scheduling appointments, **106**
Semicolon (;)
  in DISP and PRINT statements, **37**, 166
  in key definitions, **143**, **145-146**
Serial access to data files, **220**
Serial number (*of computer*), **267**
Service, **280**
  centers European, **281**
  centers U.S., **281**
  international, **281-282**
  Repair charge, **282**
  shipping instructions, **282**
  warranty, **282**
SET command, **93**
Set-time template, **12**, **93**
Setting
  the lower bound of arrays, **193**
  a program timer, **186-187**
  the right margin (MARGIN *command*), **40**
Setting the system clock, **12-13**
SGN (*sign*) function, **83**, **333**
[SHIFT] [APPT] (*shift appointment*) keystroke, **106**
[SHIFT] [ATTN] (*shift attention*) keystroke, **13**, **286**
[SHIFT] [BACK] keystroke, **35**
[SHIFT] [DEL] (*shift delete*) keystroke, **36**, 104, **286**
[SHIFT] [FET] (*shift fetch*) keystroke, 20, **286**
[SHIFT] [I/R] (*shift insert/replace*) keystroke, **42**, 144, **288**
[SHIFT] key, **27**, **285**
[SHIFT] [RUN] (*single step execution*) keystroke, **256-257**, **286**
[SHIFT] [TAB] keystroke, 35, **102**, **286**
[SHIFT] [↓] (*shift down-arrow*) keystroke, **53**, **286**
[SHIFT] [↑] (*shift up-arrow*) keystroke, **53**, **286**
[SHIFT] [←] (*shift left-arrow*) keystroke, **35**, 38, **286**
[SHIFT] [→] (*shift right-arrow*) keystroke, **35**, 38, **287**
SHORT statement, 80-81, **194-195**
Sign (SGN) function, **83**, **333**
Sign symbol specifier, image format string, **243-244**
Simple numeric variables, **78**
Simulating pressing keys in programs (PUT), **204-205**
Simulating string arrays, **205**
SIN (*sine*) function, **86**, **333**
Single line user defined functions, **205-207**
Single-step execution of programs, **256-257**
Smallest integer (CEIL) function, **82**, **332**
Smallest machine number (EPS), **76**
Specifying
  card files, **116-117**
  mass storage files, **133-134**
SQR (*square root*) function, **83**, **333**

Standard accessories, **264**
STANDBY OFF command, **29, 174**
STANDBY ON command, **29**
Statements
  BASIC, **162**
  not allowed after THEN, **178**
  not executable from the keyboard, **162**
STATS (TIME *status*) command, **93-94, 109-111**
STATS template, **94, 109-111**
Status indicators. See *annunciators*
STEP keyword, **180-181**
Stopping program execution (STOP *statement*), **165-166**
Stopwatch program, **220**
Storing and retrieving arrays (*data files*), **223-224**
Storing data in a file, **217-221, 223-228**
STR$ function, 198, **201**
String
  arrays, simulating, **205**
  comparisons, **203-204**
  concatenation (&), **164-165**
  constant, **196**
  echoing, **37-38**
  expressions, 152, **196, 203-204**
  function names (*user defined*), **205**
  functions, 196, **197-203, 333**
  manipulations, **197-198**
  operators, **330-331**
  specifiers, image format string, **240-241**
  variables, 79, **164-165**, 194, 196, 331
Strings, image format, **239**
Subroutines, 176, **182, 183-185**
Subscripts,
  Arrays, **192**
  Substring, **197-198**
Substrings, **197-198**, 205, 331
Subtraction operator (−), **69**, 72
Suspending program execution (WAIT *statement*), **166**
Syntax
  diagrams, **335-348**
  errors, **52**, 174, **252, 305-306**
  guidelines, **32, 334**
System catalog, **49**, 63
System commands,
  Non-programmable, **162**
  Setting a machine condition with, **163**
  Syntax diagrams of, **335-348**
System clock,
  Accuracy of, **268**
  Adjusting the, **95-97**
  Setting the, **12-13, 93**
System errors, **300**
System keys and keystrokes, **285-286**
System memory requirements, 47, **292**
System resets,
  Conditions that cause, **13**
  Default conditions after, **295**
  Preventing, during battery replacement, **271**

T

TAB function, **167-168**
TAB key, 16, 35, **102, 285**
TAN (*tangent*) function, **86, 333**
Temperature limits, operating, **267**
Temporary file. See *volatile file*
Text
  editing, **25-27**
  prompt, **18**
Text files, **46**
  accessing from programs, **224-225**
Text, unquoted, in DATA statement, **211**
TIME command field, **94**
TIME commands, 92-93, **334**
  ADJST (*adjust clock*), 93, **95-97**
  EXACT, 93, **95-97**
  In key definitions, **147**
  RESET, **93**
  SET, 92, **93**
  STATS (*status*), 92, **94-95**, 109-111
TIME display fields, **12**
  AM/PM, **93**
  Command, 92, **94**
TIME display formats, **92**
  Changing, **93-94**
TIME function, **98**
TIME$ function, **98**
TIME key, 14, **285**
TIME Mode, 14, **15, 92-98**
  commands, **92-93, 334**
  warning, **304**
Timeout period, **13, 29**
Timer
  branches and subroutines, **188-189**
  interrupts, 176, **186-188**
  memory requirements, **292**
  nesting, **189-190**
  numbers, **186**
Tracing operations,
  Cancelling, **253**
  Tracing branches with, **252-253**
  Tracing variable assignments with, **252-253**
Tracing program execution, **252-256**
  TRACE FLOW command, **252-253**
  TRACE OFF command, **253**
  TRACE VARS command, **252-253**
Transforming files (TRANSFORM *command*), 64, 173, **274-278**
  Memory requirements for, **292**
Transmission interruptions, HP-IL, **130-131**
Trigonometric
  commands, **85**
  functions, **85-87, 332-333**
Turning
  computer on and off, **11-13**, 174
  HP-IL loop on and off, **130**
  off appointment mode, **106**
  off program timers, **187**
  off trace operations, **253**
Two-dimensional array, **192-193**
Typewriter keys, **284-285**
Typing aids, **143-144**

U

Unary minus, **69**, 72
Unconditional branching
  CALL statement, **230**
  GOSUB statement, **182**
  GOTO statement, **176-177**
Underflow, numeric, **76**
Undoing key redefinitions, **144-145**
UNPROTECT command, **121-122**
Unquoted text, **211**
Up-arrow (↑) key, **53, 285**
UPRC$ (*uppercase*) function, 198, **201**
User defined functions, **205-209**
  Multi-line, **207**
  Parameters of, **205**, 206, 208
  Single line, **205-206**
Using the ac adapter/recharger, **11**
Using file commands with allocated programs, **172-173**
Using trace commands, **254-256**

V

VAL function, 198, **200**
Variable assignments,
  DATA statement, **210-211**
  INPUT statement, **168-170**

LET statement, **79, 164**
Multiple, **80**
Reassignments, **80**
Variable, prompt, **170**
Variables, calculator, **81, 164**
Clearing, **82**
Variables, displaying values of, 80, **166-168**
Variables, memory requirements, **292**
Variables, names, **78-79, 193, 331**
Variables, numeric array,
Declaring, **193-194**
Dimensioning, **194**
Initializing, **195-196**
Naming, 78, **193, 331**
Setting lower bounds of, **193**
Storing and retrieving using PRINT # and READ #, **223-224**
Variables, precision, **331**
INTEGER statement, 80-81, **194-195**
REAL statement, 80-81, **194-195**
SHORT statement, 80-81, **194-195**
Variables, program, **81**
Assigning values to (LET), **165**
Assigning values to, from the keyboard (INPUT), **168-170**
Clearing, **82**
Initializing, **158**
Reading values into (READ *statement*), **210-213**
Tracing assignments to, **253**
Types of, **164**
Variables, simple numeric, **78-79**, 80-81, **164**, 331
Variables, string,
Comparing, **203-204**
Dimensioning, **164, 194**
Modifying, 164-165, **197-198**
Naming, **79, 164, 331**
Redefining, **197-198**
Substrings of, **197**
VER$ (*version number*) function, **267**
Verifying
card files, **23**
HP-IL operations, **279**
proper operation, **278-279**
Volatile files, **45**

W-X-Y-Z

WAIT statement, **166**
Warranty, **279**
Warranty information, **280**
WIDTH command, **39-40**, 130, 160, 167
Workfile, 18, 50, **63-64**

HP-75 Instruction Set Index

System Commands	Page
ALARM OFF	106
ALARM ON	106
ASSIGN IO	126
AUTO	51
BEEP OFF	30
BEEP ON	30
BYE	29
CAT	49/134*
CAT ALL	49
CAT CARD	117
CLEAR LOOP	131
CLEAR VARS	81
CONT	159
COPY	118/135*
DEFAULT OFF	89
DEFAULT ON	89
DEF KEY	143
DELAY	39
DELETE	59
DISPLAY IS	128
EDIT	62
ENDLINE	140
FETCH	53
FETCH KEY	146
INITIALIZE	132
LIST	56
LIST IO	127
LOCK	28
MARGIN	40
MERGE	60
NAME	64
OFF IO	130
OPTION ANGLE DEGREES	85
OPTION ANGLE RADIANS	85
PACK	137
PLIST	56
PRINTER IS	128
PROTECT	121
PURGE	50/137*
PWIDTH	39
RENAME...TO	59/137*
RENUMBER	57
RESTORE IO	130
RUN	158
STANDBY OFF	29
STANDBY ON	29
TRACE FLOW	252
TRACE OFF	253
TRACE VARS	253
TRANSFORM	275
UNPROTECT	121
WIDTH	39

BASIC Statements	Page
ASSIGN #	216
BEEP	30
CALL	230
DATA	210
DEF FN	205
DIM	194
DISP	166
DISP USING	238
END	165
END DEF	207
FOR...TO...STEP	179
GOSUB	182
GOTO	176
IF...THEN...ELSE	177
IMAGE	238
INPUT	168
INTEGER	194
LET	165
LET FN	207
NEXT	179
OFF ERROR	259
OFF TIMER #	187
ON ERROR	258
ON TIMER #	186
ON...GOSUB	183
ON...GOTO	182
OPTION BASE	193
POP	183
PRINT	167
PRINT #	217
PRINT USING	238
PUT	204
RANDOMIZE	83
READ	211
READ #	219
REAL	194
REM	166
RESTORE	213
RESTORE #	221
RETURN	182
SHORT	194
STOP	165
WAIT	166

BASIC Functions	Page
ABS	82
ACOS	86
ANGLE	86
ASIN	86
ATN	86
CAT$	198
CEIL	82
CHR$	41
COS	86
COT	86
CSC	86
DATE	98
DATE$	98
DEG	86
EPS	83
ERRL	260
ERRN	260
EXP	84
FLOOR	82
FP	82
INF	83
INT	82
IP	82
KEY$	198
LEN	198
LOG	84
LOG10	84
MAX	83
MEM	47
MIN	83
MOD	83
NUM	41
PI	83
POS	198
RAD	86
RES	71
RMD	83
RND	83
SEC	86
SGN	83
SIN	86
SQR	83
STR$	198
TAB	167
TAN	86
TIME	98
TIME$	98
UPRC$	198
VAL	198
VER$	267

Arithmetic Operators	Page
+	69
-	69
*	70
/	70
^	70
DIV or \	70
Relational Operators	
=	87
<> or #	87
>	87
>=	87
<	87
<=	87
Logical Operators	
AND	88
OR	88
EXOR	88
NOT	88
TIME Mode Commands (Nonprogrammable)	
ADJST	95
EXACT	95
EXTD	110†
RESET	97
SET	93
STATS	93/109†

* The second page reference is to HP-IL uses of the command.

† The second STATS reference and the EXTD reference are for APPT mode.

Getting Started

Keyboard and Display Control

File Editing

Keyboard Calculations

Numeric Functions and Expressions

TIME Mode Operations

APPT Mode Operations

Card Reader Operations

HP-IL Operations

Redefining the Keyboard

Programming Fundamentals

Branches, Loops, and Subroutines

Arrays, Strings, and User-Defined Functions

Storing and Retrieving Data

Program Calls

Display and Printer Formatting

Debugging Operations

Accessories Included with the HP-75

Owner's Information

Keyboard Operations

Reference Tables

Error Conditions

Owner's Pac Program Listings

Glossary

Syntax Summary

Index

HEWLETT PACKARD

Portable Computer Division
1000 N.E. Circle Blvd., Corvallis, OR 97330, U.S.A.

European Headquarters
150, Route du Nant-D'Avril
P.O. Box, CH-1217 Meyrin 2
Geneva-Switzerland

HP-United Kingdom
(Pinewood)
GB-Nine Mile Ride, Wokingham
Berkshire RG11 3LL

00075-90001 Rev. D

Printed in U.S.A. 6/86