

# MINC-11

## Book 2: MINC Programming Fundamentals

June 1980

This manual teaches the fundamentals of computer programming, using MINC and the BASIC programming language. *Book 3: MINC Programming Reference* should be used in conjunction with this book for technical details and quick reference.

This manual supersedes *Book 2: MINC Programming Fundamentals*, Order Number AA-D799A-TC.

Order Number AA-D799B-TC

MINC-11

VERSION 1.2

Software and manuals should be ordered by title and order number. In the United States, send orders to the nearest distribution center. Outside the United States, orders should be directed to the nearest DIGITAL Field Sales Office or representative.

**NORTHEAST/MID-ATLANTIC REGION**

Technical Documentation Center  
Cotton Road  
Nashua, NH 03060  
Telephone: (800) 258-1710  
New Hampshire residents: (603) 884-6660

**CENTRAL REGION**

Technical Documentation Center  
1050 East Remington Road  
Schaumburg, Illinois 60195  
Telephone: (312) 640-5612

**WESTERN REGION**

Technical Documentation Center  
2525 Augustine Drive  
Santa Clara, California 95051  
Telephone: (408) 984-0200

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by DIGITAL or its affiliated companies.

Copyright ©, 1978, 1980, Digital Equipment Corporation.  
All Rights Reserved.

The postage-prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DEC	DECnet	IAS
DECUS	DECsystem-10	MASSBUS
Digital Logo	DECSYSTEM-20	PDT
PDP	DECwriter	RSTS
UNIBUS	DIBOL	RSX
VAX	EduSystem	VMS
MINC-11	VT	

# CONTENTS

<b>CHAPTER 1 INTRODUCTION</b>	<b>1</b>
WHAT IS MINC?	2
Using the Terminal	2
The BASIC Language	3
Commands and Statements	3
The MINC Workspace	4
Special Terminal Keys	4
MINC VOLUMES	5
USING THIS MANUAL	6
<b>CHAPTER 2 USING MINC IN THE IMMEDIATE MODE</b>	<b>7</b>
THE PRINT STATEMENT	8
Numeric Literals	9
Arithmetic Operations and Priority of Operators	11
SOME MINC FUNCTIONS	16
The Square Root Function	17
Trigonometric Functions	17
Exponential Function	19
Logarithmic Functions	19
VARIABLES	20
Assigning Values to Variables	21
VARIABLES AND THE MINC WORKSPACE	22
The CLEAR Command	22
The Scratch Command	22
MULTIPLE STATEMENT LINES	22
<b>CHAPTER 3 USING MINC IN THE PROGRAM MODE</b>	<b>25</b>
A PROGRAM EXAMPLE	26
PROGRAMS AND THE MINC WORKSPACE	27
The NEW Command	28

STATEMENT NUMBERS	28
INPUT STATEMENTS	29
ARITHMETIC OPERATIONS AND PRIORITY OF OPERATORS	31
MAKING OUTPUT READABLE	31
String Literals	32
Print Zones	33
The TAB Function	35
STRING VARIABLES	36
Assignment Statements	38
Inputting String Information	38
Concatenating Strings	39
THE REMARK STATEMENT	40
<b>CHAPTER 4 MINC PROGRAM COMMANDS</b>	<b>43</b>
CORRECTING PROGRAMS	43
Deleting Lines	44
The Substitute Command	44
The Resequence Command	47
SAVING PROGRAMS	48
The SAVE Command	49
Program Files	49
The OLD Command	52
The REPLACE Command	53
The DIRECTORY Command	54
Printing a Program	56
Checking the Length of a Program	57
LISTING AND RUNNING PROGRAMS	58
Listing Programs	58
Running Programs	59
COMPILING A PROGRAM	61
USING OTHER VOLUMES	62
MINC Volumes	62
Deleting a Program File	66
Recovering Unused Space	66
Initializing a Volume	68
Finding Bad Blocks	70
Duplicating a Volume	71
Copying a File	72
<b>CHAPTER 5 PROGRAM CONTROL</b>	<b>75</b>
IF STATEMENTS AND LOGICAL EXPRESSIONS	75
IF Statements in BASIC Are Like English	75
Logical Expressions	76
IF/THEN Statements	78
IF/GO TO Statements	81
Programming the "Otherwise"	81
GO TO Statements	81
Unconditional GO TO Statements	81

ON/GO TO Statements	82
RESEQUENCING PROGRAMS WITH GO TOs	82
PROGRAM TERMINATION	83
END Statements	83
The STOP Statement	83

## **CHAPTER 6 USING A REPETITIVE PROCESS 85**

LOOPS USING IF STATEMENTS AND GO TOs	86
FOR LOOPS AND THE FOR/NEXT STATEMENTS	88

## **CHAPTER 7 ARRAYS AND NESTED LOOPS 93**

CREATING AN ARRAY	94
WHY USE ARRAYS?	95
ONE-DIMENSIONAL ARRAYS	96
TWO-DIMENSIONAL ARRAYS	98
NESTED LOOPS	100
Nested FOR Loops	103
USING ARRAY ELEMENT 0	104
One-Dimensional Arrays	104
Two-Dimensional Arrays	105
SUBSCRIPTED VARIABLES	106
Subscripts	106
Examples	107
Subscripted Variables	109
HOW ARRAYS ARE STORED IN THE WORKSPACE	110

## **CHAPTER 8 DATA TYPES AND FUNCTIONS 113**

DATA TYPES	113
Real Variables and Literals	113
Integer Variables and Literals	113
Integer Arithmetic	115
Mixed Mode Arithmetic	115
ARITHMETIC AND TRIGONOMETRIC FUNCTIONS	117
Integer Function	118
Absolute Value Function	118
Random Number Function and RANDOMIZE Statement	118
Computing the Sign of an Expression	120
STRING FUNCTIONS	120
Clock and Calendar Functions	121
The TIME and DATE Commands	121
String Manipulation Functions	122
Finding the Length of a String	122
Trimming Trailing Blanks Off a String	123
Finding the Position of a Substring	123
Copying Segments from a String	124
Conversion Functions	125
Character and ASCII Code Conversions	126
Numbers and Their String Representation Conversions	127

Example — Converting Lower Case to Upper Case	128
USER-DEFINED FUNCTIONS	130
<b>CHAPTER 9 SUBROUTINES</b>	<b>135</b>
THE GOSUB AND RETURN STATEMENTS	137
EXAMPLE OF SUBROUTINES	138
THE ON/GOSUB STATEMENT	140
RESEQUENCING PROGRAMS WITH GOSUBS	141
<b>CHAPTER 10 MINC ROUTINES</b>	<b>143</b>
<b>CHAPTER 11 FILE CONTROL</b>	<b>145</b>
PROGRAM FILES AND OTHER FILES	145
SEQUENTIAL FILES	147
Opening a Sequential File	147
Closing a Sequential File	149
Using a Sequential File	149
Storing Data in a Sequential File	149
Accessing Data in a Sequential File	151
Checking For the End of the Input File	152
Restoring a File to the Beginning	153
Example of Using Sequential Files	153
Another Example	162
VIRTUAL ARRAY FILES	163
Opening a Virtual Array File	165
Closing a Virtual Array File	167
Using Virtual Array Files	168
Example of Using a Virtual Array File	168
DELETING A FILE	170
RENAMING A FILE	170
<b>CHAPTER 12 OTHER BASIC STATEMENTS</b>	<b>173</b>
READ AND DATA STATEMENTS	173
The RESTORE Statement	176
MINC SYSTEM FUNCTIONS	176
<b>CHAPTER 13 FORMATTED OUTPUT</b>	<b>177</b>
PRINT	178
PRINT USING	178
FORMATTING NUMERIC OUTPUT	179
FORMATTING STRING OUTPUT	182
PRINT USING STATEMENT ERROR CONDITIONS	184
<b>CHAPTER 14 COMBINING PROGRAMS</b>	<b>185</b>
CHAINING PROGRAMS TOGETHER	186
Example — A Sequential File Maintenance Program	187
Preserving Values of Variables in a Chain	194
APPENDING PROGRAMS	196
OVERLAYING A PROGRAM	200

<b>CHAPTER 15</b>	<b>KEYPAD EDITING WITH MINC</b>	<b>203</b>
	THE THREE EDITING COMMANDS	204
	MINC'S KEYPAD AND OTHER SPECIAL KEYS	206
	BASIC Programs and the Keypad Editor	207
	INSPECTING AN ASCII FILE	207
	INS Operations and Symbols	208
	INS Exercises	210
	Introducing ↑ FILE, ↓ FILE and Tone	212
	Introducing Searching	213
	Introducing Unique Search Models	216
	EDITING AN ASCII FILE	221
	EDI Operations and Symbols	222
	EDI Exercises	224
	Control Characters and the Keypad Editor	231
	Screen Width and the Keypad Editor	233
	CREATING AN ASCII FILE	233
	CRE Operations and Symbols	234
	CRE Exercises	234
<b>CHAPTER 16</b>	<b>DEBUGGING YOUR PROGRAMS IN THE IMMEDIATE MODE</b>	<b>235</b>
<b>CHAPTER 17</b>	<b>WHERE TO GO FROM HERE</b>	<b>239</b>
<b>APPENDIX A</b>	<b>ASCII CHARACTER SET</b>	<b>241</b>
<b>INDEX</b>		<b>245</b>

In MINC you perform this calculation by typing in the following BASIC command:

```
PRINT (27 + 32)*(15-8)/327
```

MINC displays the answer beneath the PRINT statement, like this:

```
PRINT (27 + 32)*(15-8)/327
1.263
```

When you use MINC as a calculator, you are using it in what is called the *immediate mode*. In the immediate mode, MINC performs your instructions as soon as you press RETURN.

The rules for using MINC as a calculator are quite simple and are explained in this chapter.

## THE PRINT STATEMENT

To get MINC to display information on the terminal, you must use the PRINT statement. As you saw above, the form of the PRINT statement is:

```
PRINT expression (RET)
```

where expression represents the number or calculation that is to be printed, and (RET) represents pressing the RETURN key. The expression is printed in blue ink, because it is optional in a PRINT statement. If you omit the expression, MINC prints a blank line.

The following three examples are valid PRINT statements where 7, 23.8, and the result of  $5324.7 + 78625.9$  are to be printed by each PRINT statement respectively.

```
PRINT 7
```

```
PRINT 23.8
```

```
PRINT 5324.7 + 78625.9
```

If you enter each of the above statements, MINC prints the results on the screen as follows:

```
PRINT 7
7
```

## FIGURES

Figure	1.	Organization of a New Diskette	63
	2.	Initialized Diskette with No Files	64
	3.	Diskette with Program Files	65
	4.	MINC System Volume with Program Files	65
	5.	Diskette with File Deleted	67
	6.	Collected Diskette	67
	7.	One-Dimensional Array	96
	8.	One-Dimensional Array	99
	9.	Two-Dimensional Array	99
	10.	One-Dimensional Array	106
	11.	One-Dimensional Array	110
	12.	Two-Dimensional Array	111
	13.	Two-Dimensional Array Stored in the Workspace	111
	14.	The Keypad	206
	15.	The Calendar in File EDITOR.001	210
	16.	The Faulty Calendar in File EDITOR.002	223

In MINC you perform this calculation by typing in the following BASIC command:

```
PRINT (27 + 32)*(15-8)/327
```

MINC displays the answer beneath the PRINT statement, like this:

```
PRINT (27 + 32)*(15-8)/327
1.263
```

When you use MINC as a calculator, you are using it in what is called the *immediate mode*. In the immediate mode, MINC performs your instructions as soon as you press RETURN.

The rules for using MINC as a calculator are quite simple and are explained in this chapter.

## THE PRINT STATEMENT

To get MINC to display information on the terminal, you must use the PRINT statement. As you saw above, the form of the PRINT statement is:

```
PRINT expression (RET)
```

where expression represents the number or calculation that is to be printed, and (RET) represents pressing the RETURN key. The expression is printed in blue ink, because it is optional in a PRINT statement. If you omit the expression, MINC prints a blank line.

The following three examples are valid PRINT statements where 7, 23.8, and the result of  $5324.7 + 78625.9$  are to be printed by each PRINT statement respectively.

```
PRINT 7
```

```
PRINT 23.8
```

```
PRINT 5324.7 + 78625.9
```

If you enter each of the above statements, MINC prints the results on the screen as follows:

```
PRINT 7
7
```

# CHAPTER 1

## INTRODUCTION

This book teaches you the commands necessary to use your MINC system. However, this book teaches more than just the syntax of the commands — it also teaches you how to put these commands together to produce your desired results. This book teaches you good techniques for making MINC work more effectively as well as for putting the commands together more effectively.

This book is designed to be read sequentially from the front to the back because the later chapters build on your knowledge from the earlier ones. Consequently, the later chapters are not all easy to understand. In order to show you how to use your MINC effectively, some of the later examples become somewhat complex. However, the examples are explained quite thoroughly; and if you take the time to understand the examples, you will finish this book with a good knowledge of your MINC and how to use it.

All of the examples in this book have been chosen carefully. Examples can never reflect every possible environment, but the ones presented in this manual all reflect possible uses for MINC in the laboratory environment. Many of the examples are actually tools that you will want to use in your own work with MINC.

This book discusses all of the MINC commands but does not discuss all of the technical details of each command. The more complex and technical points of the commands are left for *Book 3: MINC Programming Reference*.

This book repeatedly makes reference to Book 3. Feel free to look up a subject in Book 3 for further information. Book 3 is arranged in alphabetical order by topic for easy reference.

## WHAT IS MINC?

MINC, which stands for Modular INstrument Computer, is both a calculator and a computer. When MINC is ready to accept commands from you, it displays a READY message. MINC has two modes: the *immediate mode* and the *program mode*. In the immediate mode, MINC performs the instructions immediately as you type them. In the program mode, MINC saves the instructions that you type and performs them only when you request it.

You communicate with MINC through the terminal. You *input* information to MINC by typing on the terminal keyboard, and MINC *outputs* information to you by displaying its messages on the terminal screen.

Notice the flashing box on the terminal screen. This flashing box is called the cursor. If anything happens on the screen, it happens where the cursor is. When you type a character on the screen, MINC displays the character at the current cursor location and then moves the cursor to the right, one place.

You can type a command to MINC whenever MINC is displaying the READY message. You terminate a command by pressing the RETURN key. If you are in the immediate mode when you press the RETURN key, MINC immediately tries to *execute* the command. If you are in the program mode when you press the RETURN key, MINC does not try to execute the command until you tell it to.

You cannot damage MINC by typing an incorrect command. If you notice a mistake before you press RETURN, you can erase your mistake by pressing the DELETE key until the mistake is erased, and then you can type in the correct command.

If you press RETURN before you notice the mistake, MINC will tell you that you made a mistake. Then MINC will display the READY message and will wait for you to type the command again.

## Using the Terminal

You type commands to MINC using the terminal keyboard, which is like a typewriter. MINC understands commands in both upper and lower case.

When you type a character, MINC reads it and then displays it on the terminal screen. Usually, when you type a lower case character, MINC displays a lower case character. Sometimes, however, when MINC displays a question for you to answer, it will display your response in upper case, even when you type your response in lower case characters.

You communicate with MINC using the BASIC language. Basic is a language that is like English but has a limited vocabulary. In BASIC, each word in the vocabulary has only one meaning to MINC.

## The BASIC Language

For example, in BASIC, the verb PRINT tells MINC to display something on the terminal screen. You tell MINC what to print, and MINC prints it.

On the other hand, in English, the verb print can mean one of the following:

- Produce a page of text using a printing press (like to print a newspaper).
- To write in a certain manner — a child is taught to print before being taught to write in script.
- To produce a photograph on paper from a negative.

Thus, BASIC is less ambiguous than English and is easier for MINC to understand.

Due to the limited vocabulary, you must be careful to type exactly what you mean. For example, you cannot use a lower case ell (l) instead of a one (1). A lower case ell (l) means something very different to MINC than a one (1) does. Similarly, you cannot use a capital O (oh) in place of a 0 (zero).

The BASIC language gives two types of instructions to MINC: *commands* and *statements*. Until now, the term command has been used loosely to mean any instruction given to MINC. However, there is a difference between commands and statements. A BASIC *command* can be used only in the immediate mode. A BASIC *statement* can be used in either the immediate mode or the program mode.

## Commands and Statements

You can put a series of BASIC statements together to form a

*program*. Thus, a program is just a group of statements that perform a larger task than you can perform with just one statement.

### The MINC Workspace

MINC has a *workspace* in which it stores a program that you type and any values that you tell it to save. The workspace has a limited size, and thus, can only hold a limited amount of information.

MINC does not save the results of any command in the workspace, but it does save the results of some statements — even in the immediate mode. As you type in a program, MINC saves that program in the workspace. However, MINC can only save one program in the workspace at a time. If you type in a new program, you destroy the old program in the workspace.

### Special Terminal Keys

There are some terminal keys that have special meanings to MINC.

**The RETURN Key** The RETURN key signals the end of your command or statement. In examples, this key is represented by the following symbol:

`(RET)`

This symbol appears only in the first few examples in this manual to show you where to use it. In later examples, you should remember to press the RETURN key at the end of the line shown in the example.

**The NO SCROLL Key** The NO SCROLL stops all output from appearing on the terminal screen. If you press the NO SCROLL key a second time, the output resumes on the screen. Thus, if you are typing on the terminal and nothing appears on the screen, press the NO SCROLL key, just in case you pressed the NO SCROLL key by mistake. The NO SCROLL key is an on-off switch for the terminal output.

**Control Characters** There are special characters called *control characters*. For example, you send MINC a “Control C” by pressing the CTRL key at the same time as the C key (similar to sending a capital C by pressing the SHIFT key at the same time as the C key). In a paragraph the “Control C” character is represented as CTRL/C. In an example, it is shown as:

`(CTRL/C)`

When you press CTRL/C, the following symbol appears on the screen.

^C

The control characters that are important to MINC are: CTRL/C, CTRL/S, CTRL/Q, and CTRL/U. There are other control characters that have significance, but the other control characters are explained in Book 3.

The CTRL/C character stops a program. This character is explained in Chapter 3.

The CTRL/S character stops all output from appearing on the terminal screen. The CTRL/Q character shows all the output that was hidden with the CTRL/S character. Thus, if you are typing on the terminal and nothing appears on the screen, press CTRL/Q, just in case you typed CTRL/S by mistake.

The CTRL/S and CTRL/Q characters together are the same as the NO SCROLL key.

The CTRL/U key signals MINC to ignore the current line that you are typing. When you press CTRL/U, MINC returns the cursor to the next line. The line you want to erase still appears on the screen, but MINC ignores it.

Before you can use your MINC, you must place a MINC system diskette in the left diskette drive. The diskette is a more permanent storage medium than the MINC workspace. If you do not have a MINC system diskette, read *Book 1: Introduction to MINC* to learn how to create a system diskette from the Master diskette.

The diskette is a physical medium for storage. A *volume* is a logical storage medium to MINC. For example, a volume to a person can be one book in a set of encyclopedias — any one of the set, not one in particular. A volume to MINC is any storage medium — not just one particular diskette. Thus, the term volume is used throughout this manual to refer to a diskette.

MINC has logical devices as well as logical storage media. The diskette drive is a physical device. The diskette is placed in a diskette drive. The logical names for the two diskette drives are SY0: and SY1: (the colon is part of the name). SY0: is the name

## MINC VOLUMES

of the left diskette drive, and SY1: is the name of the right diskette drive.

## USING THIS MANUAL

To learn BASIC most effectively, you should read this book and try the examples on your MINC as you read them. The best way to learn BASIC is to use it.

To try the examples in this book, you must have a demonstration diskette in the left diskette drive (SY0:). The directions for starting your MINC system and obtaining a copy of the demonstration diskette are in Part II of *Book 1: Introduction to MINC*.

## CHAPTER 2

# USING MINC IN THE IMMEDIATE MODE

The immediate mode is used for two purposes:

- To display calculations.
- To command MINC to perform tasks such as running and saving programs.

This chapter describes how to use MINC to do calculations. Chapter 4 describes the immediate mode commands that direct MINC to perform other tasks.

Imagine yourself using a small calculator to do the following calculation:

$$\frac{(27 + 32)(15 - 8)}{327}$$

you would probably follow these steps:

1. Enter 27.
2. Add 32.
3. Store result (in memory or on paper)
4. Enter 15.
5. Subtract 8.
6. Multiply by what is stored in memory or on paper.
7. Divide by 327.

At this point the calculator display is showing the correct answer, which is 1.263.

In MINC you perform this calculation by typing in the following BASIC command:

```
PRINT (27 + 32)*(15-8)/327
```

MINC displays the answer beneath the PRINT statement, like this:

```
PRINT (27 + 32)*(15-8)/327
1.263
```

When you use MINC as a calculator, you are using it in what is called the *immediate mode*. In the immediate mode, MINC performs your instructions as soon as you press RETURN.

The rules for using MINC as a calculator are quite simple and are explained in this chapter.

## THE PRINT STATEMENT

To get MINC to display information on the terminal, you must use the PRINT statement. As you saw above, the form of the PRINT statement is:

```
PRINT      (RET)
```

where expression represents the number or calculation that is to be printed, and (RET) represents pressing the RETURN key. The expression is printed in blue ink, because it is optional in a PRINT statement. If you omit the expression, MINC prints a blank line.

The following three examples are valid PRINT statements where 7, 23.8, and the result of  $5324.7 + 78625.9$  are to be printed by each PRINT statement respectively.

```
PRINT 7
```

```
PRINT 23.8
```

```
PRINT 5324.7 + 78625.9
```

If you enter each of the above statements, MINC prints the results on the screen as follows:

```
PRINT 7
7
```

```
PRINT 23.8
23.8
```

```
PRINT 5324.7 + 78625.9
83950.6
```

You can insert blanks or spaces to make the PRINT statement more readable. MINC ignores blanks. All of the following examples have the same meaning to MINC.

```
PRINT 7
```

```
PRINT7
```

```
P R I N T 7
```

You can also type the PRINT statement in any mixture of upper and lower case. All of the following examples have the same meaning to MINC.

```
prINT 7
```

```
print 7
```

```
PRINT 7
```

However, in this manual, all MINC statements are typed in upper case to distinguish them from other parts of the text.

A single number can be the expression of a PRINT statement. Another name for a number is a *numeric literal*. Any number whose magnitude is between .01 and 999999 can be printed just the way it looks. Zero is also printed the way it looks. For example:

```
PRINT .01
.01
```

```
PRINT 999999
999999
```

```
PRINT -.01
-.01
```

```
PRINT 0
0
```

## Numeric Literals

Numeric literals cannot contain commas or spaces. Numeric

literals can contain up to six significant digits; that is, numeric literals can have up to six digits, not counting zeroes preceding the number or zeroes trailing the decimal point. Some valid numeric literals are:

0.123456

999999

99999.9

123.456

In a PRINT statement, MINC rounds decimal places to six significant digits. That is, if the value of the seventh digit (last decimal place) is five or less, the remaining digits are lost; if the value of the seventh digit is six or greater, the sixth digit is incremented by one. For example:

<i>PRINT Statement</i>	<i>What Is Printed</i>
PRINT .1234565	.123456
PRINT .1234566	.123457
PRINT 1.2345672	1.23457

Numbers that have a magnitude greater than 999999 or less than .01 are printed in scientific notation, where E denotes the exponent, or power of 10, involved. For example:

<i>PRINT Statement</i>	<i>What Is Printed</i>
PRINT .01	.01
PRINT .0099	9.90000E-03 ( $9.9 \times 10^{-3}$ )
PRINT 999999	999999
PRINT 1000000	1.00000E+06 ( $1 \times 10^6$ )
PRINT .0100	.01
PRINT 090	90
PRINT 1E-01	.1
PRINT 1E6	1.00000E+06 ( $1 \times 10^6$ )

Examples 5 and 6 show that MINC removes preceding and trailing zeroes. Examples 7 and 8 show that you can use E notation in a PRINT statement. In Example 7, MINC converts the E notation to .1 because the value is between .01 and 999999. In Example 8, MINC converts the result to 1.00000E+06 because this is the form in which MINC prints E notation, even though MINC understands shortcuts.

is between  $10^{-38}$  and  $10^{38}$ .

MINC performs the following arithmetic operations: addition, subtraction, multiplication, division, and exponentiation. In order to obtain the correct result of a calculation, you should know the order in which MINC performs these operations. MINC performs calculations in a certain order unless forced by you to do otherwise. All MINC calculations must be written on one line.

### Arithmetic Operations and Priority of Operators

Any of the arithmetic operators (described in the following sections) combined with numeric literals are valid expressions. Expressions containing arithmetic operations are called *arithmetic expressions*.

**Addition and Subtraction** Addition is denoted by a plus sign (+). Sums are obtained by putting those numbers to be added in a PRINT statement with plus signs between them. For example:

```
PRINT 7+2
9
```

```
PRINT 4.3+9.2+8.7
22.2
```

```
PRINT 6E6 + 5E5
6.50000E+06
```

Subtraction is denoted by a minus sign (-). Differences are obtained by putting those numbers to be subtracted in a PRINT statement with minus signs between them. For example:

```
PRINT 8-5
3
```

```
PRINT 9.4-6.5-2
.9
```

```
PRINT 5.3E-7 - 4.9E6
-4.90000E+06
```

Notice that in the last example, 0.00000053 is negligible next to 4,900,000.

Addition and subtraction are of equal priority; that is, if MINC is given a series of additions and subtractions, it proceeds from left to right. For example:

```
PRINT 5+7-2
10
```

```
PRINT 6-2-3+4
5
```

```
PRINT 4.9-3.2+6.7-10.9
-2.5
```

In the second example, the first subtraction (6-2) is done first, then the second subtraction (4-3), and finally the addition (1+4).

**Multiplication and Division** Multiplication is denoted by an asterisk (\*) rather than an  $\times$ , and the \* must never be left out if a multiplication is to be performed. In mathematical texts, multiplication is represented in several ways; sometimes with an  $\times$ , sometimes with a dot, and sometimes by writing expressions without any operator.

In MINC, the only way to obtain the product of two numbers is to insert an asterisk (\*) between the numbers to be multiplied. For example:

```
2*3
```

You can obtain products of numbers by putting those numbers to be multiplied in a PRINT statement. For example:

```
PRINT 8*5
40
```

```
PRINT 4.35*6.72
29.232
```

```
PRINT 6E4*3E9
1.80000E+14
```

The multiplication operation has a higher priority than addition and subtraction and thus is done first when combined with addition or subtraction. For example:

```
PRINT 2+3*7
23
```

MINC does the multiplication (3\*7) first and then the addition (2+21). Another example:

```
PRINT 2+3*7-9
14
```

the addition ( $2 + 21$ ) and subtraction ( $23 - 9$ ) are done from left to right since they are of equal priority.

Division is denoted by a slash (/) and the divisor must appear on the same line as the dividend. For example, three-sevenths in mathematical notation must appear as  $3/7$  in MINC notation.

Examples of division are:

```
PRINT 3/5
.6
```

```
PRINT 10/3
3.33333
```

```
PRINT 20/3
6.66667
```

```
PRINT 10E6/5E4/2
100
```

```
PRINT 6/0
?MINC-W-Dividing by zero
0
```

In the final example, an error message notes that division by zero is not defined and consequently cannot be computed.

Multiplication and division are operations of equal priority. When they are combined, MINC performs these operations from left to right. For example:

```
PRINT 2*3/9
.666667
```

```
PRINT 4/10*5
2
```

```
PRINT 5*4/10
2
```

```
PRINT 5E8 * 2E7 / 9E6
1.11111E+09
```

Division, like multiplication, is of higher priority than addition and subtraction, and MINC performs division before addition or subtraction. For example:

```
PRINT 5/2+3
5.5
```

```
PRINT 2+3*5-6/3
15
```

In the second example, the multiplication ( $3*5$ ) is done first, then the division ( $6/3$ ), and then the addition and subtraction proceed from left to right.

**Exponentiation** In mathematics, to raise 2 to the third power, you write:

$$2^3$$

Because calculations in MINC must be written on one line, exponentiation is denoted by a caret (^), and thus to raise 2 to the third power you write:

$$2^3$$

```
PRINT 2^3
8
```

MINC processes exponentiation operations, like the other operations, from left to right. For example:

```
PRINT 2^3^2
64
```

First MINC computes  $2^3 (=8)$  and then squares the result ( $=64$ ).

**Overriding Priorities** You can override the priority of multiplication and division over addition and subtraction by the use of parentheses. Anything in parentheses has priority over anything outside parentheses. In the following two examples, see how the parentheses affect the result.

```
PRINT 2+3*7
23
```

```
PRINT (2+3)*7
35
```

In the second example, MINC performs the addition ( $2+3$ ) first because this operation is enclosed in parentheses. Another example:

```
PRINT 6+(2+7)/(2+1)-4
5
```

MINC performs the addition  $(2+7)$  first, then the addition  $(2+1)$ , because they are in parentheses; then MINC performs the division because it has the next higher priority; and finally MINC performs the addition  $(6+3)$  and subtraction  $(9-4)$  from left to right.

You can place parentheses within parentheses, to *nest* one expression within another. For example:

```
PRINT (2+7*(6+5))/24+5
      8.29167
```

MINC performs the innermost addition  $(6+5)$  first. Then MINC performs the operations within the outer parentheses according to the usual priorities. Finally, MINC performs the operations outside the parentheses according to the usual priorities. That is:

$$(2+7*(6+5))/24+5$$

$$6+5=11$$

$$7*11=77$$

$$2+77=79$$

$$79/24=3.29167$$

$$3.29167+5=8.29167 \quad \text{The Answer}$$

Note that every left parenthesis must be matched somewhere in the expression by a right parenthesis.

Table 1. Priority of Operators

<i>Operation</i>	<i>Priority</i>
( )	highest
* or /	
+ or -	lowest

Exponentiation has the highest priority of all the operations as shown in Table 1, but like the rest, that priority can be overridden by parentheses. For example:

```
PRINT 2+3^3
      29
```

```
PRINT (2+3)^3
125
```

```
PRINT (2^3)^2
64
```

```
PRINT 2^(3^2)
512
```

```
PRINT 2*2^3
16
```

```
PRINT (2*2)^3
64
```

## SOME MINC FUNCTIONS

MINC can perform some functions that are difficult to calculate by hand. For example, MINC can compute square roots, logarithms, and trigonometric functions.

A *function* is an operation on an argument or arguments that computes a value. For example, the symbol

$$\sqrt{x}$$

represents the value equal to the square root of  $x$ . The  $x$  is the *argument* of the square root function.

To print the value of the square root of 2 on MINC, type:

```
PRINT SQR(2)
1.41421
```

MINC prints the value of the square root of 2 on the terminal screen.

A function, which consists of the function name followed by a parenthesized expression called an *argument*, can be used as an expression of a PRINT statement, as above. A function can also be used with a larger expression wherever an expression is valid. For example:

```

      function
      name argument
PRINT(-5 + SQR(5^2-4*3*2))/(2*3)
           element of
           PRINT expression
           PRINT expression

```

MINC functions calculate results to six significant digits.

Some MINC functions are described in the following sections. The remaining functions are described in Chapter 8.

The square root function, denoted as SQR, calculates the value of the square root of the argument. The following four examples show uses of the SQR function. The third example shows that the square root function can be used as the argument of the square root function; that is, function calls can be *nested*. The fourth example shows that an expression can be the argument of the SQR function as well as the SQR function being an element of the larger expression of the PRINT statement.

### The Square Root Function

```
PRINT SQR(4)
2
```

```
PRINT SQR(2)
1.41421
```

```
PRINT SQR(SQR(16))
2
```

```
PRINT (-5 + SQR(5^2 - 4*3*2))/(2*3)
-.666667
```

As with all MINC operations, the precision of the result of the square root function is to six significant digits.

The PI function always takes on the value of  $\pi$  to six significant digits (3.14159). The PI function has no argument and can be used wherever the value 3.14159 is required.

### Trigonometric Functions

The sine, cosine, and arc tangent functions, which are denoted by SIN, COS, and ATN respectively, calculate the value of the sine, cosine, and arc tangent of their arguments respectively.

The arguments of both the sine and cosine functions are angles that must be expressed in radians. You can convert an angle in degrees to radians with the following identity:

$$\text{radians} = \text{degrees} * \text{PI} / 180$$

where PI is the function that takes on the value 3.14159.

The form of the sine function is:

SIN(expression)

The form of the cosine function is:

COS(expression)

For example:

```
PRINT SIN(PI)
1.87254E-07
```

```
PRINT COS(PI)
-1
```

Notice that MINC returns a value of .0000001 for the sine of  $\pi$ . This value is approximately 0. See the Numeric Precision section of Book 3.

If you want the value of the sine of 239 degrees, type:

```
PRINT SIN(239*PI/180)
-.857167
```

As with the SQR function, you can nest the trigonometric functions. For example,

```
PRINT SQR(COS(45*PI/180))
.840896
```

The form of the arc tangent function is:

ATN(expression)

The ATN function calculates the value in radians in the range  $+\pi/2$  to  $-\pi/2$ . For example:

```
PRINT ATN(32.435)
1.53998
```

You must compute the other trigonometric functions — tangent, cotangent, arc sine, arc cosine — using the SIN, COS, and ATN functions. For example, the following PRINT statement prints the tangent of  $\pi/3$ .

```
PRINT SIN(PI/3)/COS(PI/3)
1.73205
```

The exponential function raises the number  $e$  (approximately 2.71828) to the power specified by the argument of the function. That is,  $\text{EXP}(7)$  is equivalent to  $e^7$ .

## Exponential Function

The form of the exponential function is:

$\text{EXP}(\text{expression})$

For example:

```
PRINT EXP(7)
1096.63
```

```
PRINT EXP(SQR(2 + 3))
9.35647
```

The LOG function calculates the value of

## Logarithmic Functions

$\log_e(\text{expression})$

The LOG function is the inverse of the EXP function since the following relationship is true:

$\text{LOG}(\text{EXP}(\text{expression})) = \text{expression}$

For example:

```
PRINT LOG(2.718281)
1
```

```
PRINT LOG(EXP(5))
5
```

The LOG10 function returns the value of

$\log_{10}(\text{expression})$

For example:

```
PRINT LOG10(10)
1
```

```
PRINT LOG10(100)
2
```

Logarithms to any base may be easily computed using the following formula:

$$\log_a(\text{expression}) = \frac{\log_e(\text{expression})}{\log_e(a)}$$

For example, the MINC statement to compute the log base 2 of 512 is:

```
PRINT LOG (512)/LOG(2)
9
```

## VARIABLES

Up to now, all of the calculations have involved numeric literals. However, most mathematical formulas have variables in them — that is, quantities that can vary. For example, the following formula represents the area of a circle.

$$\text{area} = \pi r^2$$

The  $\pi$  represents the number 3.14159 and the  $r$  represents the radius of the circle. The  $r$  is a variable — that is, it is a placeholder whose value varies for different circles. On the other hand,  $\pi$  is a constant whose value is always 3.14159.

Similarly, the formula for conversion from Fahrenheit to Celsius has constants and variables.

$$T_c = 5/9*(T_f - 32)$$

$T_c$  is the variable representing the result in degrees Celsius and  $T_f$  represents degrees Fahrenheit. The numbers 32 and 5/9 are numeric literals.

In MINC, a *numeric variable* is the name of storage for a numeric value. The storage is allocated in the workspace.

You can use variables in the immediate mode to save space on a line in a long calculation and to cut down on typographical errors. If you are going to do several calculations with the same large expression, you can save time by storing the value of the expression in a variable.

For example, if you want the sine, cosine, tangent, and arc tangent of 45 degrees, you might type the following:

```
PRINT SIN(45*PI/180)
PRINT COS(45*PI/180)
PRINT SIN(45*PI/180)/COS(45*PI/180)
PRINT ATN(45*PI/180)
```

However, using the variable A to represent 45 degrees converted to radians, you need only type:

```
A = 45*PI/180
PRINT SIN(A)
PRINT COS(A)
PRINT SIN(A)/COS(A)
PRINT ATN(A)
```

MINC represents numeric variables with a letter or a letter followed by a digit. For example, the variable names F and F1 represent two distinct variables. You can print the area of a circle with radius 5 by using the following statements:

```
R=5

PRINT PI*R^2
```

In MINC, a numeric variable represents a numeric value. In the previous example, the variable R has a numeric value of 5. The next section describes how to assign a value to a variable.

You can assign a value to a variable with an *assignment statement*. The form of the assignment statement is:

### Assigning Values to Variables

```
LET variable-name = expression
```

The keyword LET is optional. The value on the left of the equals sign must be a variable name. The value on the right side of the equals sign must be a valid expression. There is another type of numeric variable called an *integer variable* which can contain only whole number values. Integer variables are discussed in Chapter 8.

The following example shows assigning the value 7 to the variable R.

```
R=7
```

Hereafter, whenever you use the variable R, MINC goes to the workspace and uses the value 7. For example:

```
PRINT PI*R^2
153.938
```

The equals sign (=) does not denote mathematical equality. Instead, in MINC it means, "take the value to the right of the

equals sign and place it in the variable to the left of the equals sign." For example, both of the following statements are acceptable to MINC, but the second is not valid in mathematics.

A=2

A=A+1

The first statement tells MINC to give variable A the value 2. The second statement tells MINC to calculate the value of A + 1 (which is 2 + 1 or 3), and then to place this value in the variable A. At the end of statement 2, A has the value 3.

## VARIABLES AND THE MINC WORKSPACE

As stated earlier, MINC stores variables in the workspace. Whenever you use a variable, MINC sets up a place in the workspace for it. The value of the variable is zero until you assign a new value to it.

When you turn MINC off, all the variables you used are erased from the workspace. Thus, when you turn MINC on, there are no variables until you create one by assigning a value to it.

### The CLEAR Command

To reset all variables to zero, use the CLEAR command. The form of the CLEAR command is:

CLEAR

When you use the CLEAR command, your variables still exist in the workspace, but MINC has set their values to zero.

### The Scratch Command

The SCR command, which stands for scratch, erases the entire workspace. The form of the SCR command is:

SCR

When you use the SCR command, MINC zeroes all variables and erases any program you stored in the workspace. (See Chapter 3.)

## MULTIPLE STATEMENT LINES

You can type more than one BASIC statement on a line on the screen by using the backslash (\). The form of the backslash is:

statement \ statement

The backslash works only for BASIC statements and not for commands. Note that the backslash (\) is different from the slash (/), which indicates division.

For example:

```
A = 45*PI/180 \ PRINT SIN(A) \ PRINT COS(A) \ PRINT ATN(A)
```

```
.707107
```

```
.707107
```

```
.665774
```

```
READY
```

By placing more than one statement on a line, you can see all of the results together instead of separated by blank lines and READYS.

Note that you can only put as many statements on a line as will fit. Remember that BASIC statements cannot cross line boundaries.

In MINC you perform this calculation by typing in the following BASIC command:

```
PRINT (27 + 32)*(15-8)/327
```

MINC displays the answer beneath the PRINT statement, like this:

```
PRINT (27 + 32)*(15-8)/327
1.263
```

When you use MINC as a calculator, you are using it in what is called the *immediate mode*. In the immediate mode, MINC performs your instructions as soon as you press RETURN.

The rules for using MINC as a calculator are quite simple and are explained in this chapter.

## THE PRINT STATEMENT

To get MINC to display information on the terminal, you must use the PRINT statement. As you saw above, the form of the PRINT statement is:

```
PRINT expression (RET)
```

where expression represents the number or calculation that is to be printed, and (RET) represents pressing the RETURN key. The expression is printed in blue ink, because it is optional in a PRINT statement. If you omit the expression, MINC prints a blank line.

The following three examples are valid PRINT statements where 7, 23.8, and the result of  $5324.7 + 78625.9$  are to be printed by each PRINT statement respectively.

```
PRINT 7
```

```
PRINT 23.8
```

```
PRINT 5324.7 + 78625.9
```

If you enter each of the above statements, MINC prints the results on the screen as follows:

```
PRINT 7
7
```

## CHAPTER 3

# USING MINC IN THE PROGRAM MODE

The advantage of a computer is its ability to perform repetitive or iterative tasks automatically. As a simple example, consider computing the sine and cosine of many angles in degrees. When using MINC in the immediate mode to compute the sines and cosines of 45°, 272°, 130°, and 90°, you would have to type all of the following lines. The comma between expressions instructs MINC to print two numbers on the same line.

```
A=PI/180
```

```
PRINT SIN(45*A),COS(45*A)
.707107    .707107
```

```
PRINT SIN(272*A),COS(272*A)
-.999391    .034892
```

```
PRINT SIN(130*A),COS(130*A)
.766045    -.642788
```

```
PRINT SIN(90*A),COS(90*A)
1          0
```

This is a lot of typing, and if you need the sines and cosines of more angles, you will have to do more typing and will have more chance for error. You can type a BASIC program that will compute the sine and cosine of any angle in degrees, but the program needs to be typed only once.

A *program* is a series of statements that can be executed as a whole. It can perform more than one calculation or task.

## A PROGRAM EXAMPLE

Suppose a physics student has a MINC available to her for her homework. Since she needs to compute sines and cosines for many of her physics problems, she wrote a program to compute as many sines and cosines as she needs. Her program asks for an angle in degrees and then prints out the sine and cosine. The program keeps asking for another angle until she types in a CTRL/C (presses the CTRL key at the same time as the C key).

The student's program follows. You should type it in just to see how a program works. *Do not worry if you do not quite understand the program.* All of the programming concepts shown here are explained later in this manual.

What you type in is shown in red ink. MINC's responses are shown in black.

This program should be typed exactly as it appears here. You must type each statement perfectly. If you make a typographical error, simply retype the line. MINC executes the statements in numerical order, even if you did not enter them in order. Remember that spaces within a statement are there only to make the statement easier to read — they do not matter to MINC.

```
READY
NEW
NEW FILE NAME -- SINES
```

The NEW command erases and then names the workspace. Thus, if you want to save the program you are typing in, give the program a name. In this case the workspace is named SINES.

```
READY

20 PRINT "SINE","COSINE"
30 PRINT "DEGREES";
40 INPUT X
60 Z = X*PI/180
70 PRINT ,SIN(Z),COS(Z)
80 GO TO 30
90 END
```

If your program is not typed perfectly, retype lines that are not correct by retyping the line number followed by the statement. For example, if you had typed:

```
40 INPUT C
```

you can now type:

40 INPUT X

and MINC will correct the line by replacing the old line 40 with the new one.

To see the way your correct program looks, type:

LIST

The LIST command causes your entire program to appear on your terminal screen.

When you are convinced that your program is typed properly, type:

RUN

MINC will then *run* or *execute* your program; in other words, MINC will perform the instructions you have given it.

Output from the above program follows. What MINC types on the screen is represented in black ink while your responses are represented in red. After entering each response, press the RETURN key.

SINES	20-APR-80	03:28:32	
		SINE	COSINE
DEGREES? 45		.707107	.707107
DEGREES? 272		-.999391	.0348992
DEGREES? 130		.766045	-.642788
DEGREES? 90		1	0
DEGREES? ^C			
STOP at line 40			
READY			

If you type RUN, MINC will perform this program again; you can enter more angles and MINC will compute more sines and cosines. You can also save this program and call it back when you need more sines and cosines, saving yourself many steps. (How to save programs is discussed in Chapter 4.)

You can run the program many times because MINC stores programs in the workspace as you type them. Before you type in a new program, you should make sure that any previous program is erased from the workspace by typing the SCR command (explained in Chapter 2) or the NEW command (explained in the following section).

## PROGRAMS AND THE MINC WORKSPACE

When you type a program, MINC stores the program in the workspace in the upper case, whether you typed in the program in upper case or lower case. MINC also formats the program for you. MINC leaves at most one space between words, no matter how you type in the program.

## The NEW Command

Like the SCR command, the NEW command erases the workspace. However, the NEW command also allows you to give the workspace a name. The SCR command automatically names the workspace NONAME.

The NEW command has two forms. If you type only NEW followed by RETURN, MINC prompts you for the new workspace name.

```
NEW(RET)  
NEW FILE NAME --
```

Here you type in a name from one to six characters long. The name can be any combination of letters and digits.

The second form of the NEW command is:

NEW name

where name is the name that you want (from one to six characters). In this second case, MINC does not prompt you for the name.

You should use the NEW or the SCR command to erase the SINES program from the workspace before you try the next examples. Unlike the NEW command, the SCR command automatically names the workspace NONAME. Thus, if you care about the name of the workspace, use the NEW command.

## STATEMENT NUMBERS

A statement number preceding a BASIC statement determines whether MINC is in the immediate mode or the program mode. For example:

```
PRINT 7
```

causes MINC to print a 7 immediately, while:

```
10 PRINT 7
```

does not cause MINC to do anything immediately. When you type in a statement with a statement number, MINC does not

execute the statement, but rather places the statement in the workspace. The above example is a one-line program consisting of one PRINT statement. To execute a program (for example, 10 PRINT 7), you must type the RUN command. For example:

```
READY
SCR
```

```
READY
10 PRINT 7
RUN
```

```
NONAME          20-APR-80          04:46:28
```

```
7
```

```
READY
```

The program mode is much more powerful than the immediate mode because it permits MINC to perform a series of statements together as in the example in the beginning of the chapter.

Statement numbers tell MINC the order in which to execute the BASIC statements. MINC executes the statements in ascending numerical order, whether you entered the statements in numerical order or not. The numbers do not have to be consecutive. In fact, it is wise to leave a gap between statement numbers in case you omitted a statement and have to insert it later.

The main purpose of a computer is to accept data as input, process the data, and produce output that is readable and usable.

## INPUT STATEMENTS

Assignment statements assign values to variables, these values being determined by the program. In contrast, INPUT statements stop a program to accept data values from the user (the person using the program — possibly someone other than the programmer).

If you use variables along with the INPUT statement a program can process any value that you input, as long as it is within the boundaries of the program. The following example demonstrates how to use the INPUT statement.

Suppose an engineer needs to find the volume of many cylinders. For each cylinder, the engineer must know the area of the base and the volume. The area of a circle is:

$$\pi r^2$$

where  $r$  is the radius of the circle (base). The volume of a cylinder is:

$$\pi r^2 h$$

where  $h$  is the height of the cylinder.

To find the area and volume of a cylinder with radius 2cm and height 5cm, the engineer could enter the following line into MINC.

```
PRINT PI*2^2, PI*2^2*5
```

For each cylinder having different dimensions, the engineer would have to enter this same line with the new dimensions. If, however, the engineer wrote a program to calculate the area and volume of the cylinder, he would have to type the program only once and it would take as input the radius and height and would print the area of the cylinder base and volume of each cylinder. The following program would do this:

```
NEW CYLIND
```

```
READY
```

```
10 INPUT R,H  
20 PRINT PI*R^2, PI*R^2*H
```

When the engineer types RUN, the program, from statement 10, will produce a question mark as a prompt indicating that MINC is waiting for him to type in the radius and the height separated by commas. The engineer must press the RETURN key to signify the end of his input line. Then, underneath these input amounts, MINC will print the area and volume of the cylinder. The screen would look like this:

```
RUN
```

```
CYLIND                20-APR-80                04:48:37
```

```
?2,5
```

```
12.5664                62.8319
```

Red represents what the engineer typed and black represents what MINC typed.

In this example, the INPUT statement assigns to the variables

R and H the values that the engineer entered at the question mark. Now, in line 20, MINC uses 2 and 5 as the values for the variables R and H respectively, to compute the area of 12.5664 and volume of 62.8319.

The form of the INPUT statement is:

INPUT variable-list

where variable-list is one variable or, as in this example, many variables separated by commas. For example:

```
10 INPUT F
20 INPUT A,B,C1
```

In statement 10, MINC gives a question mark as a prompt for you to enter one piece of information followed by a RETURN. An INPUT statement produces one question mark for each variable it expects. In statement 20, MINC issues a question mark and expects three numbers separated by commas or RETURNS. In the second case, if you have entered only two numbers before you pressed RETURN, MINC issues another question mark for the third number.

When you type RUN and MINC starts to execute the program, all variables have the value zero. If there is an INPUT statement in the program, then those variables listed in the INPUT statement are given the new values that you enter at the question mark.

The arithmetic operations, addition, subtraction, multiplication, division, and exponentiation, are the same in the program mode as in the immediate mode; the order of priority being: exponentiation as the highest, multiplication and division second, and addition and subtraction lowest.

At the time MINC performs a calculation, it treats a variable as a number, giving it the value you entered in the INPUT statement. The priority of the operations remains the same with variables as with numbers.

The engineer in the earlier example decided that he was not satisfied with his program for calculating the area and volume of his cylinders. He did not like the fact that the only prompt for input was a question mark which required him to remember

## ARITHMETIC OPERATIONS AND PRIORITY OF OPERATORS

## MAKING OUTPUT READABLE

what he had to type in. A much more meaningful dialog on the terminal would be:

```
CYLIND          21-APR-80          09:43:03

Radius? 2
Height? 5
Area: 12.5664
Volume: 62.8319
```

In this case, he would know easily what he should type in, and the results are clearly labelled. The changes to his program are minimal. His original program was:

```
10 INPUT R,H
20 PRINT PI*R^2, PI*R^2*H
```

His new version of the program is:

```
CYLIND          21-APR-80          10:27:27

10 PRINT 'Radius'; \ INPUT R
20 PRINT 'Height'; \ INPUT H
30 PRINT 'Area: '; PI*R^2
40 PRINT 'Volume: '; PI*R^2*H

READY
```

and the output is the same as shown previously.

These new changes are explained in the next two sections.

## String Literals

Statement 10 of the engineer's new program prints out the label "Radius". This label is enclosed in apostrophes within the PRINT statement and is called a *string literal*. A string literal can be alphabetic letters or numbers or both.

Digits in a string literal are characters rather than numbers. Notice the difference in the following two PRINT statements. The first prints a numeric value, so MINC computes the value and prints out the result, which is 5. The second prints a string literal. PRINT "2+3" means to print the character "2" followed by the character "+" followed by the character "3".

```
PRINT 2+3
5
```

```
PRINT '2+3'
2+3
```

When you are entering string literals, you must enclose them in a pair of apostrophes (') or in a pair of quotation marks ("). If you need to use an apostrophe within a string literal, then you should use quotation marks to delimit the string literal. If you need to use quotation marks within a string literal, then you should use apostrophes to delimit the string literal. In all other cases, you can use either apostrophes or quotation marks, but you cannot use an apostrophe at one end of the string literal and a quotation mark at the other. For example.

```
PRINT "THE BOY'S COAT"
THE BOY'S COAT
```

Although spaces have no meaning within BASIC statements, within string literals spaces are significant. In the second of the two following examples, the space character is part of the string constant because it is enclosed within the quotation marks.

```
P R I N T 2 + 3
5
```

```
PRINT "2 + 3"
2 + 3
```

In the following example, notice that MINC does not capitalize lower case letters within string literals.

Examples:

```
10 PRINT "the area"
20 PRINT 'of a circle is'
30 PRINT "computed using the"
40 PRINT 'formula pi r squared'
RUN
```

```
NONAME                12-JUN-80                11:20:15
```

```
the area
of a circle is
computed using the
formula pi r squared
```

In line 10 of the program CYLIND, the engineer placed a semicolon (;) after his string literal. The semicolon prevents the terminal from going to a new line before printing the question mark prompt from the INPUT statement. In the second of the

**Print Zones**

two partial examples below, there is no semicolon after RADIUS and consequently the prompt from the INPUT statement is printed on the next line.

```
10 PRINT 'Radius';
20 INPUT R
RUN
```

```
NONAME          24-APR-80          10:30:46
```

Radius?

READY

```
10 PRINT 'Radius'
20 INPUT R
RUN
```

```
NONAME          24-APR-80          10:31:15
```

Radius  
?

MINC considers the terminal screen to be divided into five 14-space columns or zones. A comma in a PRINT statement causes the terminal to skip to the next print zone. For instance, in the following example, the first comma causes the terminal to skip to the second zone, SINE is printed in the second zone, the second comma causes the terminal to skip to the third zone, and COSINE is printed in the third zone.

```
PRINT ',SINE','COSINE'
          SINE      COSINE
```

Nothing is printed in the first zone (because the initial comma causes the terminal to skip the first zone).

Two successive commas skip an entire zone. For example, the following PRINT statement would print A in the first zone and B in the third zone.

```
PRINT 'A',,, 'B'
A              B
```

In the next example, MINC prints 6 on the next line because there are only five print zones per line.

```
PRINT 1,2,3,4,5,6
  1      2      3      4      5
  6
```

To suppress the print zones, use the semicolon. For example:

```
PRINT 1;2;3,4,5,6
 1  2  3          4          5          6
```

Because there is a semicolon between 1 and 2 and between 2 and 3, MINC prints the 1, 2, and 3 in the same zone. MINC leaves one space before each number for a sign (in this case space for positive) and one space after each number as a separator.

The comma between 3 and 4 causes the 4 to be printed in zone two. The 5 and 6 are printed in zones three and four because of the commas.

Note that a semicolon puts no spaces between string constants. For example:

```
PRINT 'A';'B'
AB
```

To skip lines, use PRINT statements without arguments — one PRINT statement for each skipped line, as shown below.

```
PRINT
```

If you end the PRINT statement with a semicolon, MINC does not start the next PRINT statement on a new line. For example:

```
10 PRINT 1;
20 PRINT 2;
30 PRINT 3;
40 PRINT 'this is the end of the line'
```

```
READY
RUN
```

```
NONAME          20-JUL-80          11:40:05
```

```
1 2 3 this is the end of the line
```

```
READY
```

The TAB function causes the terminal to skip to a specified column within the line to be printed. The form of the TAB function is:

```
PRINT TAB(expression);
```

## The TAB Function

where expression is a column number greater than 0. If the expression is greater than the number of columns on a line, MINC continues to skip lines until it can tab to a column within the line. If the column number specified in the expression is less than 0, the following error message is printed.

?MINC-F-Arguments in definition do not match function called

If the expression is not an integer, MINC uses only the integer portion of the number. In the following two examples, a hyphen (—) represents a space, so that you can count the spaces more easily and understand what the TAB function does. For example:

```
PRINT TAB(5);'RADIUS'
----- RADIUS
```

```
PRINT TAB(7.32);'FUN'
----- FUN
```

In the second example, because the expression is not integer, MINC uses only the integer portion — that is, the terminal spaces to column 8, skipping 7 spaces.

You should use a semicolon after the TAB function because a semicolon prevents MINC from skipping to the next print zone.

If the column number specified is less than or equal to the current column number, printing starts at the current position.

## STRING VARIABLES

In many applications, it is necessary to input alphabetic characters. For example, suppose you want to use one of the graphic features of MINC to plot a set of data points. This program may first accept the labels of the axes (alphabetic data) as input and then the numeric data to plot. In this way, the program will work for any graph that needs to be produced.

For example, the following program accepts as input a name and address and prints them out. Remember that the backslashes let you put more than one statement on the same line.

```
10 PRINT 'YOUR NAME'; \ INPUT N$
20 PRINT 'YOUR ADDRESS'; \ INPUT S$,C$
30 PRINT \ PRINT N$ \ PRINT S$ \ PRINT C$
RUN
```

```
YOUR NAME? JOHN S. DOE
YOUR ADDRESS? 11 MAIN ST.
? WORCESTER MA. 01605
```

```
JOHN S. DOE
11 MAIN ST.
WORCESTER, MA. 01605
```

The variables N\$, S\$, and C\$ are *string variables*. That is, their values must be strings. A string variable name is any variable name followed by a dollar sign (\$). Examples of string variable names are:

```
A$
B$
D7$
```

The string variable A\$ is a separate and distinct variable from the numeric variable A. Both names A\$ and A can be used within the same program.

Input to string variables entered in response to the question mark from the INPUT statement need not be enclosed within quotation marks. MINC will accept input with or without quotation marks. However, if a comma is part of the input string, you must enclose the data within quotation marks or MINC will think you are entering more than one string. String variables can have as a value any collection of characters, spaces, and numbers. The maximum length of a string variable is 256 characters. For example,

```
10 PRINT 'input only one string variable'
20 INPUT A$
30 PRINT A$
RUN
```

```
NONAME                26-JUL-80                14:23:18
```

```
input only one string variable
? one, two
?MINC-W-Extra values from keyboard or file ignored at line 20
one
```

```
READY
```

Just as MINC sets numeric variables to zero when you run a program (until you give them a value), MINC sets string variables to the *null string* — the string with zero characters (the string constant " "). A string variable can take on the value of

the null string if you just press RETURN in response to the INPUT prompt.

Spaces are significant within string variables just as they are significant within string constants.

### Assignment Statements

Just as you can set the value of numeric variables with an assignment statement, you can set the value of string variables with an assignment statement. The form of the string assignment statement is:

LET string-variable-name = string-expression

The LET is printed in blue ink because it is optional. The string-expression is a string literal, another string variable name, or a combination of string literals, string variables, and string operators (which are described in the next section).

### Inputting String Information

You can also use the INPUT statement to input both numeric and string information. However, the INPUT statement is limited in usefulness for string information. You can use the LINPUT statement for string information.

The LINPUT statement has essentially the same function as the INPUT statement. However, the LINPUT statement can accept only string information. The LINPUT statement accepts and stores all characters *including quotation marks and commas* except the line terminator. The terminator is not stored with the string.

The form of the LINPUT statement is:

LINPUT string-variable-list

The string-variable-list is composed of string variables separated by commas. Each string variable is assigned a line of input, which consists of all the characters up to the line terminator (RETURN).

For example:

READY  
LIST

```

10 LINPUT B$
20 PRINT B$
30 END
RUN

```

```

NONAME          09-MAY-80          09:50:47

```

```

? "Now, look here!", said John.
"Now, look here!", said John.

```

```

READY

```

The following program, which is identical to that above except that it inputs string data with an INPUT statement, prints a warning message. The comma after the quotation mark terminates the string. Notice that the quotation marks delimit the string but are not part of it.

```

10 INPUT B$
LIST

```

```

NONAME          09-MAY-80          09:52:15

```

```

10 INPUT B$
20 PRINT B$
30 END

```

```

READY
RUN

```

```

NONAME          09-MAY-80          09:52:18

```

```

? "Now, look here!", said John.
?MINC-W-Extra values from keyboard or file ignored at line 10
Now, look here!

```

```

READY

```

The only string operation is concatenation. *Concatenation* puts one string after another without any intervening characters. Concatenation is specified by either the plus sign (+) or the ampersand (&). For example:

```

PRINT 'ONE' + 'WORD'

```

```

ONWORD

```

```

READY

```

```

A$ = 'GOOD' & 'BYE'

```

## Concatenating Strings

```
READY  
PRINT A$
```

```
GOODBYE
```

```
READY  
10 INPUT A$,B$,C$  
20 PRINT A$ + B$&C$  
RUN
```

```
NONAME          09-MAY-80          10:23:22
```

```
? ONE,WO,RD  
ONEWORD
```

```
READY
```

## THE REMARK STATEMENT

Since BASIC does not allow very mnemonic variable names, and since programmers quite often forget the details of their programs, the **REMARK** statement allows a programmer to place comments within a program. These comments are ignored by MINC when your program is run. Comments in a program are valuable and should be used if someone else will use your program or if you will be looking at your program at a later date — long after you remember how you wrote the program. **REM** is a valid abbreviation for **REMARK**. For example:

```
10 REMARK THIS PROGRAM CONVERTS DEGREES  
20 REM to radians  
30 INPUT D  
40 PRINT D*PI/180  
RUN
```

```
NONAME          09-MAY-80          10:27:35
```

```
? 90  
1.5708
```

Both forms of the **REMARK** statement, **REMARK** and **REM**, do the same thing.

MINC does not convert lower case characters to capitals in a **REMARK** statement. By using lower case characters in a **REMARK** statement, you can visually break up a program.

MINC stores **REMARK** statements in the workspace along with the other statements in the program. If a program gets too large

for the workspace, you can remove the REMARK statements to provide space. This will not affect the execution of the program.

The REM statement message itself can contain any printing character on the keyboard except the backslash, which will terminate the remark. Remarks are also terminated by a new line. For example:

```
10 REM THIS IS A "?@#! COMMENT
20 REM N$ represents the person's name
30 REM - A$ represents the person's address \INPUT A$
40 REM - C$ represents the person's city and state \INPUT C$
50 REM ** this is the end of this section **
```

In MINC you perform this calculation by typing in the following BASIC command:

```
PRINT (27 + 32)*(15-8)/327
```

MINC displays the answer beneath the PRINT statement, like this:

```
PRINT (27 + 32)*(15-8)/327
1.263
```

When you use MINC as a calculator, you are using it in what is called the *immediate mode*. In the immediate mode, MINC performs your instructions as soon as you press RETURN.

The rules for using MINC as a calculator are quite simple and are explained in this chapter.

## THE PRINT STATEMENT

To get MINC to display information on the terminal, you must use the PRINT statement. As you saw above, the form of the PRINT statement is:

```
PRINT expression (RET)
```

where expression represents the number or calculation that is to be printed, and (RET) represents pressing the RETURN key. The expression is printed in blue ink, because it is optional in a PRINT statement. If you omit the expression, MINC prints a blank line.

The following three examples are valid PRINT statements where 7, 23.8, and the result of  $5324.7 + 78625.9$  are to be printed by each PRINT statement respectively.

```
PRINT 7
```

```
PRINT 23.8
```

```
PRINT 5324.7 + 78625.9
```

If you enter each of the above statements, MINC prints the results on the screen as follows:

```
PRINT 7
7
```

## CHAPTER 4

# MINC PROGRAM COMMANDS

After the engineer wrote the program to compute the area of a circle and the volume of a cylinder, he wanted to save the program so he can use it repeatedly. This chapter describes the commands used for saving programs as well as those used for changing them.

*Commands* are those BASIC instructions that cannot be used with line numbers in the program mode. *Statements* are those BASIC instructions that can be used with line numbers in the program mode. In the immediate mode, statements look like commands.

For example, PRINT is a statement because you can put it into a program. RUN is a command, and cannot be put in a program.

Suppose the engineer had initially typed his program like this:

```
1 PRINT 'Radius';\ INPUT R
2 PRINT 'Area'; PU*R^2
```

This program computes only the area of a circle. However, the engineer made a typographical error and typed PU instead of PI. He can easily correct this error by retyping the whole line as follows.

```
2 PRINT 'Area'; PI*R^2
```

Now that the engineer retyped line 2, MINC replaces the old line 2 in the workspace with this corrected one. The old line 2 is now gone from the workspace.

### CORRECTING PROGRAMS

## Deleting Lines

You can delete a line of a program by typing the statement number followed by RETURN. This is the easiest way to delete lines from a program, but this method can become tedious if you have many lines to delete. You can also delete lines with the DEL command. The form of the DEL command is:

DEL statement-spec, statement-spec, ...

The first statement-spec is necessary, the rest are optional. Each statement-spec can be in one of the following forms:

stmt#	deletes the specified statement.
stmt#-stmt#	deletes all the statements in the specified range. You need the hyphen between the numbers.
stmt#-	deletes all statements from the specified statement to the end of the program.
-stmt#	deletes all statements from the beginning of the program to the specified statement.

To delete statement 10, type:

DEL 10

To delete statements 100 and 500, type:

DEL 100,500

To delete statement 25, statements 100 through 150, and statements 450 to the end of the program, type:

DEL 25, 100-150,450-

## The Substitute Command

Since the engineer had made such a minor typographical error, and since line 2 is fairly long, he could have used the substitute (SUB) command to correct line 2. To use the SUB command, the engineer would have to type:

SUB 2 [PU[PI

where 2 is the line number with the error and the bracket (I) delimits the bad string from the good string. When the engineer types this SUB command, MINC corrects line 2 in the workspace, replacing PU with PI.

The general form of the SUB command is:

SUB stmt# d bad-string d good-string

where:

stmt#	is the number of the line to be changed.
d	is any character to delimit the strings. The delimiter must not appear in either string. The bracket ([]) character is a good choice for a delimiting character because it is not used in BASIC programs except in string constants.
bad-string	is the old series of characters to be deleted. Do not delimit the string with quotation marks.
good-string	is the new series of characters to be inserted. Do not delimit the string with quotation marks.
whole-number	is the constant that specifies which occurrence of bad-string in the line is to be replaced with good-string when there is more than one occurrence. Whole-number is optional and, if omitted, MINC replaces the first occurrence.

After you have made the changes to the line, MINC prints the new line.

Suppose you have the following incorrect line and want to replace the second F9 with I.

```
100 F9 = A*SIN(X) + F9
```

Type the following SUB command to correct line 100.

```
SUB 100 [ F9 [ I [ 2
100 F9 = A*SIN(X) + I
```

READY

You can use the SUB command to change a line number. For example, if the current line 20 is:

```
20 PRINT A,B,C,D
```

and you want to change the line number to 100, type the following SUB command:

```
SUB 20 [ 20 [ 100
100 PRINT A,B,C,D
```

READY

MINC makes the correction and then prints the new line. Since line 100 is a different line from line 20, MINC copies line 20 into line 100, but does not delete line 20. If you no longer want line 20, you can delete it with the DEL command (explained previously in this chapter).

You cannot use the SUB command to delete a line number (that is, change a program line to an immediate mode statement). If you try to do this, MINC prints the following message and does not execute the command.

```
?MINC-F-SUB creates an invalid statement or has a syntax error
```

MINC also prints this message if you entered the SUB command in the wrong format, such as omitting the delimiting character that ends the bad-string.

MINC uses the first character after the line number in the SUB command as the delimiting character, but it ignores all spaces and tabs until it finds a character. Consequently, you cannot use space or tab as a delimiting character. You also cannot use a digit as the delimiting character because MINC will consider it part of the line number.

If MINC cannot find the bad-string you specify, it reprints the line with no changes. Thus, you must type bad-string exactly as MINC lists it. To see how MINC lists a line, you use the LIST command explained later in this chapter.

Remember that MINC stores a program in upper case (except for string literals and REM statements), even though you typed it in lower case. Therefore, MINC would not find the following bad-string:

```
sub 2 [ pu [ pi
```

MINC has stored the typographical error, pu, as PU. Consequently, MINC would make no change in line 2. When you use

the SUB command, if you are not changing a string literal or a REM statement, it is a good idea to set the CAPS LOCK button so that you type in capital letters. Otherwise MINC will not find your bad-string in a SUB command (even though it recognizes the SUB command in lower case).

Suppose the engineer had initially typed his program like this:

```
1 PRINT 'Radius'; \ INPUT R
2 PRINT 'Area'; PI*R^2
```

### The Resequencing Command

If he wants to add height and volume calculations to his cylinder program, he must change the program. Since the statements are numbered consecutively, the engineer cannot fit a statement between lines 1 and 2 without renumbering the program.

He can solve his problem by typing the following resequence (RESEQ) command:

```
RESEQ
```

MINC renumbers the entire program starting with the first statement of the program, numbering it as 10 and the following statements in steps of 10. Thus, after the engineer types the RESEQ command, MINC changes his program to this:

```
10 PRINT 'Radius'; \ INPUT R
20 PRINT 'Area'; PI*R^2
```

You can use the RESEQ command to resequence your entire program or sections of your program. You can specify the new line number at which you want the renumbered section (or entire program) to start, the range of line numbers that you want resequenced (the old line numbers), and the increment to be used between each line number.

The general form of the resequence command is:

```
RESEQ newstart,oldstart-oldfinish,increment
```

As shown in the engineer's example, all of these values are optional:

newstart	specifies the new starting statement number.
oldstart	specifies the lowest existing statement number to be resequenced.

- `-oldfinish` specifies the highest existing statement number to be resequenced. You must specify the hyphen before `oldfinish`.
- `increment` specifies the increment to be used between each statement number.

If you do not specify an `newstart`, MINC uses the highest existing line number less than `oldstart`. For example, in the following `RESEQ` command, the `newstart` value is missing. If the highest line number under 105 is 100, MINC renumbers the first new line (line 105 or first highest above 105) to 100 plus the increment, or 110.

```
RESEQ ,105-200,10
```

In the above example, MINC renumbers old lines 105 through 200. If you omit the `newstart` value, you must leave in the comma before `oldstart` so that MINC knows that the number is `oldstart` and not `newstart`.

If you leave out `oldstart`, MINC starts resequencing at the beginning of the program. For example, the following `RESEQ` command resequences the program from the first line through old line 200 in increments of 10. If you leave out `oldfinish` instead, MINC resequences from `oldstart` to the end of the program.

```
RESEQ ,-200
```

If you do not specify an increment, MINC increments the statement numbers by 10.

For more details of the `RESEQ` command, see Book 3.

## SAVING PROGRAMS

You can save any program that you type into the workspace for subsequent use. Once you have saved a program, you can do all of the following:

- Replace it. That is, change it and save the changed copy.
- Look at your directory. That is, look at a list of all the programs you saved.
- Delete a program. That is, throw a program away.

- Recover unused space. That is, consolidate all the empty space created by deleting programs.
- Print a program.
- Combine two programs.

Suppose you want to save the engineer's program, which you named CYLIND with the NEW command, and which you typed into the workspace. You can save this program named CYLIND by typing the following command.

```
READY
SAVE
```

After you have typed this SAVE command, the program is named CYLIND and saved on the volume in SY0: (the left drive).

When you type only:

```
SAVE
```

then MINC uses the default name in the workspace. Since you used the NEW command, the name in the workspace is CYLIND. If you had used the SCR command instead of the NEW command, the name in the workspace would be NONAME.

If you do not want your saved program to have the default name, use the following form of the SAVE command.

SAVE name

The specified name overrides the workspace name. For example:

```
SAVE CIRCLE
```

With this SAVE command, the program stored in the workspace is saved on volume SY0: with the name CIRCLE (no matter what the name of the workspace is). The program is also still in the workspace. For more details of the SAVE command, see Book 3.

When you save a program, the program is stored on a volume in a *program file*. For example, if you type:

```
SAVE CYLIND
```

## The SAVE Command

## Program Files

the program in the workspace is stored on SY0: in a program file called CYLIND.BAS. The .BAS denotes the type of the file named CYLIND. CYLIND.BAS is a file that holds a BASIC program.

The full *file specification* for the CYLIND program is:

SY0:CYLIND.BAS

where:

SY0: is the device on whose volume the program is stored. If you leave out the device, MINC defaults to SY0:. Note that you must always have a colon as part of the device name.

CYLIND is the program name. You can select any name between one and six characters in length.

.BAS is the file type denoting that CYLIND is a file that contains a BASIC program. The .BAS file type is the default file type for BASIC programs.

You can store a program in a file with any file type (up to three characters) you choose. However, if you make up your own file type, you must always enter the file type, or MINC will not find the correct file.

For example, you can enter:

SAVE CYLIND.PRG

Now, whenever you want to reference this program (to run it, for example) you must always use the name and file type, CYLIND.PRG. If you use the default file type, you need only to type the name.

For example, if you want to run the program stored in CYLIND.PRG, you must type:

RUN CYLIND.PRG

However, if you want to run the program stored in CYLIND.BAS, type:

RUN CYLIND

Below are other examples:

**SAVE**                      saves the program currently in the workspace in a file called SY0:NONAME.BAS, unless you have used the NEW command. If you have used the NEW command to name the workspace, SAVE saves the program with the name of the workspace.

**SAVE FUN**                saves the program currently in the workspace in a file called SY0:FUN.BAS.

**SAVE SY0:FUN.PRG**      saves the program currently in the workspace in a file called SY0:FUN.PRG. Remember that you must always type the file type when it is other than the default.

**SAVE SY0:HELLO.BAS**    saves SY0:HELLO.BAS.

Later in this chapter, saving programs on SY1: will be discussed.

The general form of the SAVE command is:

**SAVE**

where filespec takes the form:

dev:name.typ

and

dev:      is the device (default SY0:)

name     is the program file name (default NONAME)

.typ      is the file type (default .BAS)

There are types of files other than program files. For example, if you wanted to collect data from an instrument, you would save

this data in a *data file*. You cannot run data files or any file that is not a program file. These types of files are discussed in Chapters 11, 14, and 15.

If you try to save a program with the same name as another program already stored on the volume, MINC prints an error message. For example, if you have already saved the program CYLIND.BAS and you want to correct the program and save it again, you cannot type:

```
SAVE CYLIND
```

MINC prints the following error message.

```
?MINC-F-File name in use; REPLACE or change name or volume
```

You cannot use the SAVE command to store the new version of CYLIND. You must use the REPLACE command (described in one of the following sections) or change the file description (the volume, name, or file type).

## The OLD Command

The OLD command brings a saved program from the program file on a volume into the workspace. For example, the following OLD command brings the CYLIND program stored in SY0:CYLIND.BAS into the workspace.

```
READY
OLD
OLD FILE NAME -- CYLIND
```

```
READY
LIST
```

```
CYLIND          13-APR-80          13:46:43
```

```
10 PRINT 'Radius'; \ INPUT R
20 PRINT 'Height'; \ INPUT H
30 PRINT 'Area';PI*R^2
40 PRINT 'Volume'; \ PI*R^2*H
```

```
READY
```

Like the SAVE command, the OLD command uses SY0: as the default device and .BAS as the default file type.

Notice in the above example, if you type only OLD, MINC prompts you for the OLD FILE NAME. You can also type the file name as part of the OLD command. For example:

OLD filespec

where filespec is of the form:

dev:name.typ

The OLD command first changes the workspace name, then erases the workspace (performs a SCR), and finally brings in the program stored in the program file named by the filespec.

If there is no program file with the name specified in filespec, MINC prints the following error message:

?MINC-F-Specified or default volume does not have file named

When this error occurs, the workspace name changes, but MINC does not scratch the workspace. Thus, you must be careful not to inadvertently change a program name through this sort of error.

If you save a program, then call it back into the workspace with an OLD command, and then alter the program, you cannot use the SAVE command to put the changed program back into the old file. You can save this changed program with a new name, or you can use the REPLACE command.

### The REPLACE Command

The REPLACE command is like the SAVE command except that REPLACE saves a program even if it means writing over an existing program file. This difference between the SAVE and REPLACE commands helps prevent you from inadvertently deleting program files that you previously saved.

The form of the REPLACE command is:

REPLACE

If you omit the file specification, MINC uses the current workspace name as the default specification.

Study the following example and notice the difference between SAVE and REPLACE. This example uses the name SY0:TEST.BAS throughout (established with the NEW command).

```
NEW
NEW FILE NAME -- TEST
```

```
READY
```

## PROGRAMMING FUNDAMENTALS

```
10 PRINT 'This is a test'
SAVE
```

```
READY
20 PRINT 'This is another test'
LIST
```

```
TEST                09-MAY-80                00:03:43
```

```
10 PRINT 'This is a test'
20 PRINT 'This is another test'
```

```
READY
SAVE
?MINC-F-File name in use; REPLACE or change name or volume
```

```
READY
REPLACE
```

```
READY
```

Now the program stored on the diskette in file TEST.BAS is the new version with two lines.

For more details of the REPLACE command, see Book 3.

### The DIRECTORY Command

You can see the names of all the program files and other types of files that you save on a diskette with the DIRECTORY command (DIR).

On each diskette, MINC keeps a directory on each diskette of all the files stored on that diskette. For example, suppose you have saved two programs named CYLIND.BAS and SINES.BAS and you want to look at the directory. You can use the DIR command as shown below:

```
READY
DIR

10-MAY-80
Volume ID: MINC System
Owner: engineer
SINES.BAS 1 11-MAY-80 CYLIND.BAS 1 12-MAY-80
<unused> 122
2 Files, 2 Blocks
122 Free Blocks

READY
```

The DIR command causes MINC to print out the directory,

which consists of the current date, the volume identifier, the owner, the list of files, and a description of the available and used space. All of these features are described below.

The *Volume ID* and *Owner* were named by you or someone else when the diskette was initialized (described later in this chapter under *Initializing a Volume*).

The list of program files takes the form:

```
name.typ  b  dd-mmm-yy  name.typ  b  dd-mmm-yy
```

where:

name.typ      is the name of the file and its file type.

b              is the size of the program file in blocks. A block is a unit of size on a diskette. Blocks and diskettes are discussed more thoroughly later in this chapter (in the section titled MINC Volumes).

dd-mmm-yy    is the date that the program was saved or replaced.

The <unused> shows you the unused, available space on the diskette.

The last part of the directory listing takes the form:

```
n Files, m Blocks
x Free Blocks
```

where:

n Files              is the number of files in the directory.

m Blocks            is the total number of blocks that the n programs take.

x Free Blocks      is the total available space on the diskette.

There are a number of features of the DIR command. With the DIR command you can:

- Look at a directory of all files on a diskette.

- Look at a directory of all file names on a diskette that have the same file type; for example, all file names with the .BAS type.
- Look at a directory of all files with the same name but different file types.
- Direct the directory listing to another file.
- Direct the directory listing to the line printer.

For more details of these advanced features of the DIR command, see the DIRECTORY command in Book 3.

## Printing a Program

Until now, you have been using the LIST command to see the program that is stored in the workspace. You cannot use the LIST command to view the contents of a file stored on a volume.

To see a listing of the contents of a file that is not in the workspace on the terminal screen, use the TYPE command. The form of the TYPE command is:

TYPE filespec

where filespec takes the form:

dev:name.typ

If you leave out the device, MINC defaults to SY0:. If you leave out the file type, MINC defaults to a program file (.BAS file type). You can display files other than program files by specifying the name and file type. (For more about files other than program files, see Chapter 11.)

The TYPE command does not affect the use of the workspace. It reads the file from the volume and displays it without storing it in the workspace. That is, if you have a program in the workspace and you command MINC to type another program, the original program in the workspace remains unaffected.

For example:

```
READY
TYPE CYLIND
10 PRINT 'Radius'; \ INPUT R
20 PRINT 'Height'; \ INPUT H
30 PRINT 'Area'; PI*R^2
40 PRINT 'Volume'; \ PI*R^2*H
```

READY

You can use the NO SCROLL key to stop a long program from scrolling off the top of the screen before you have a chance to read it.

For more details of the TYPE command, see Book 3.

The LENGTH command lets you check to see how much workspace a program requires. The length of the program is given in units of workspace called *words*.

### Checking the Length of a Program

The form of the LENGTH command is:

LENGTH

To see how many words are available in your workspace, type the LENGTH command after the SCR command.

READY  
SCR

READY  
LENGTH

0 USED, 5491 FREE

READY

In the following example, when the CYLIND program is in the workspace, it uses 58 words (58 USED) and leaves 5433 words available (5433 FREE).

CYLIND                      11-MAY-80                      11:22:06

```
10 PRINT 'Radius'; \ INPUT R
20 PRINT 'Height'; \ INPUT H
30 PRINT 'Area'; PI*R^2
40 PRINT 'Volume'; \ PI*R^2*H
```

READY  
LENGTH

58 USED, 5433 FREE

The length of a program can vary from run to run. For example, if one of your programs accepts string input, the length of the

program depends on the length of the input string. If you check the length of the program before you run it, all the strings are null and consequently take less room in the workspace than after you run the program and give the strings a value other than null.

For more details of the LENGTH command, see Book 3.

### LISTING AND RUNNING PROGRAMS

So far you have been using the LIST command to list the program in the workspace on the terminal screen and the RUN command to run a program.

This section describes other features of the LIST and RUN commands.

#### Listing Programs

As said previously, you can use the LIST command to display program lines that are in the workspace. To list your entire program on the terminal screen, type:

```
LIST
```

MINC first displays a header line that consists of the program name, the current date, and the time. MINC then displays the entire program on the screen, and finally displays the READY message.

You can also list sections of the program, individual statements, groups of statements, or any combination of these.

The full form of the LIST command is:

```
LIST statement-spec, statement-spec,...
```

Each statement-spec is optional and can take one of the following forms:

stmt#	lists the specified statement.
stmt#-stmt#	lists all statements in the specified range.
stmt#-	lists all statements from the specified statement to the end of the program.
-stmt#	lists all statements from the beginning of the program to the specified statement.

The LISTNH command is the same as the LIST command except that MINC displays no header (NH).

The following example lists the entire program without a header line.

```
LISTNH
10 PRINT 'Radius'; \ Input R
20 PRINT 'Height'; \ Input H
30 PRINT 'Area'; PI*R^2
40 PRINT 'Volume'; \ PI*R^2*H
```

This next LIST command lists the header line and statement number 20 of the program.

```
LIST 20

CYLIND          11-MAY-80          11:22:10

20 PRINT 'Height'; \ INPUT H
```

It is often helpful to list a line before using the SUB command. By listing the line, you can see exactly what it looks like so you can correctly change the line.

This last example of the LIST command lists statements 25 and 50, all statements from 100 through 200, and all lines from statement 500 through the end of the program.

```
LIST 25,50,100-200,500-
```

You can run a program that is stored in the workspace by typing:

```
RUN
```

When MINC executes the RUN command, it first displays a header line. It then initializes all numeric variables to 0 and all string variables to the null string. Finally MINC starts executing the program at the lowest numbered line.

The RUNNH command has the same effect as the RUN command except that MINC does not print the header line.

It is a good practice to use the RUN command rather than the RUNNH command. In case your program has some sort of error, you at least see the header and know the program tried to run.

## Running Programs

If you want to run a program from a program file, that is, a program stored on the diskette but not in the workspace, type:

RUN filespec

where filespec takes the form:

dev:name.typ

If you do not specify the device, MINC defaults to SY0:. If you do not specify the file type, MINC defaults to the program file type (.BAS).

When you specify a file with the RUN command, MINC erases the workspace (equivalent to SCR), changes the workspace name, and brings the specified program into the workspace. Finally MINC starts execution of the program. This process destroys the previous contents of the workspace, so be sure to save it if you want to keep it before executing the RUN filespec command.

The RUN filespec command does not display a header line and is equivalent to the RUNNH filespec command. It leaves the program specified by filespec in the workspace when it is finished.

For example, the following sequence of commands brings the CYLIND program into the workspace and runs the program.

```
READY
RUN CYLIND
```

```
CYLIND          11-MAY-80          12:10:08
```

```
Radius?
```

If MINC cannot find the file specified in a RUN filespec command, it changes the workspace name and displays the following error message.

```
?MINC-F-Specified or default volume does not have file named
```

However, MINC does not delete the program originally in the workspace. This can happen if you mistype the RUNNH command, such as RUNHN. In this case, MINC interprets the command as RUN HN, where HN is the file specification.

If the file specification begins with NH, MINC assumes that the

NH is part of the RUNNH command. For example, MINC interprets a RUN NHTEST command as RUNNH TEST. To run a program whose file specification begins with NH, use the RUNNH command.

MINC does not store a program in the workspace exactly the way you type it but instead compresses (compiles) each line. By compiling the program internally, MINC allows you to fit larger programs in the workspace than you could if MINC did not compile each line.

## COMPILING A PROGRAM

Whenever you list the program or save it on a diskette, MINC translates the program from the compiled form to the form that you entered.

The COMPILE command saves the internal, compiled version on a diskette. The BASIC version is still in the workspace. Once you have saved this compiled version, MINC can bring this version into the workspace faster than it can bring a version saved with the SAVE command into the workspace. Thus the OLD and RUN filespec commands work faster.

The form of the COMPILE command is:

COMPILE filespec

Where filespec takes the form:

dev:name.typ

If you omit the filespec, MINC uses the current workspace name with the .BAC file type, which stands for a compiled program. If you omit the device, MINC defaults to SY0:. If you specify the filespec, MINC stores the compiled version into that file on the diskette.

You can COMPILE only the program currently stored in the workspace.

If you type the following command:

RUN name

MINC tries to run name.BAC first. If name.BAC is not on the diskette, then MINC tries to run name.BAS.

You can use the TYPE command to type a compiled program. However, the output is unintelligible and quite confusing.

## USING OTHER VOLUMES

So far you have saved programs and run programs; all of these programs have been stored on the diskette in the SY0: device. You can use both SY0: and SY1: with MINC, but there are some general procedures you must follow to prepare a new diskette for use.

In general, to prepare a new volume (diskette) for use, you must:

- Place the new volume in SY1:
- Initialize the volume (set up an empty directory).

After you have initialized the volume, you can copy programs, save programs, and store data on it.

The following sections discuss MINC volumes, their structure, and their use.

## MINC Volumes

The volumes that you have been using up to now are MINC *system volumes*. That is, these volumes have system files on them that make MINC work. When you put these system volumes in SY0: (the left drive) and turn on the power, they let you know that they are system volumes by printing one of the following messages:

```
MINC BASIC V1.2 for the 11/23  
MINC BASIC V1.2 for the 11/03
```

Not all volumes are system volumes. You can use a volume to store programs you have written or to store data that you have collected. If you try to start MINC with one of these *nonsystem volumes* in SY0:, you will get an error message similar to the following:

```
@  
?BOOT-F-No boot on Volume
```

When you get such a message, you must put a system volume in SY0: and start again.

You must always have a MINC system volume in SY0: or your MINC system will not work.

The following sections describe volumes in more detail.

**Master Volumes** With your MINC system you received Master volumes. They are a form of MINC system volume. You cannot use these volumes because they have been set up so that you can

only copy them. They direct you to place an empty volume in SY1: and then they copy the MINC system onto your volume. For more information, see Book 1.

You cannot manipulate a Master volume. That is, you cannot save programs or data on a Master volume. To save programs or data, you must copy the Master volume onto your own volume or volumes and then use your own volume.

You should keep many copies of the MINC system volume. That way, in case one volume is somehow damaged, you have another copy. The procedure for copying volumes is described in the following sections.

**The Structure of Volumes** A volume is a storage area for MINC. A volume has a finite amount of space; that is, it can hold only a finite amount of information.

The unit of size on a volume is a *block*, which holds 512 characters.

Figure 1 represents the organization of a new diskette. Notice that nothing is stored on it.

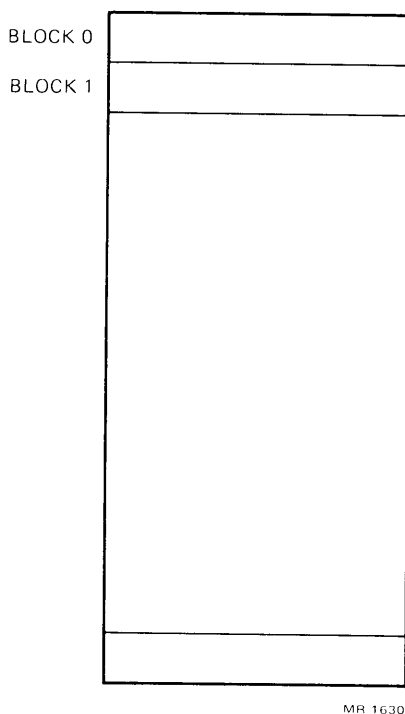


Figure 1. Organization of a New Diskette

Figure 2 represents the organization of an initialized diskette. When you *initialize* a diskette, MINC sets up a directory on that diskette. Nothing is stored in the directory, however, until you begin using the diskette for program storage or to copy system files. (The procedure for initializing a diskette is described later in this chapter.)

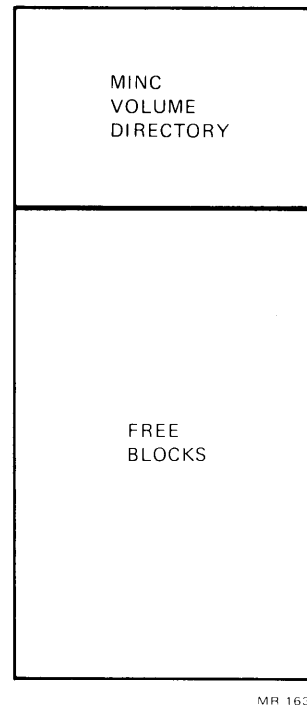


Figure 2. Initialized Diskette with No Files

Figure 3 represents a diskette that stores only program files. After initializing a new diskette, you can place it in SY1: and type the following SAVE commands.

```
READY  
OLD SINES
```

```
READY  
SAVE SY1:SINES.BAS
```

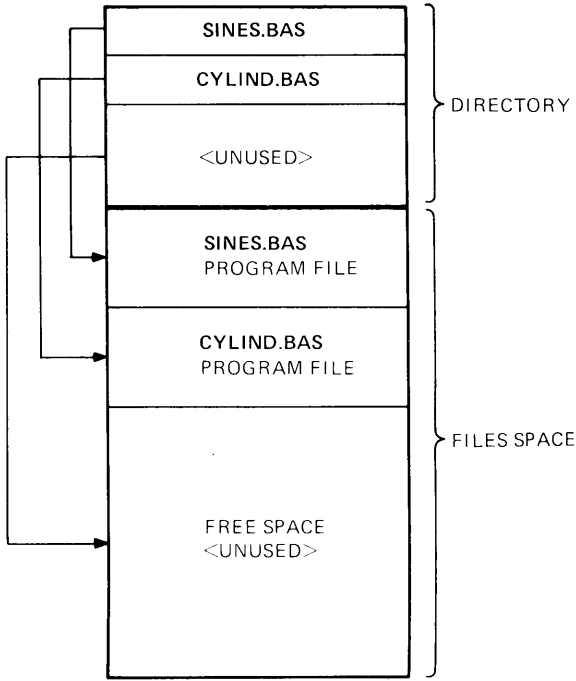
```
READY  
OLD CYLIND
```

```
READY  
SAVE SY1:CYLIND.BAS
```

```
READY
```

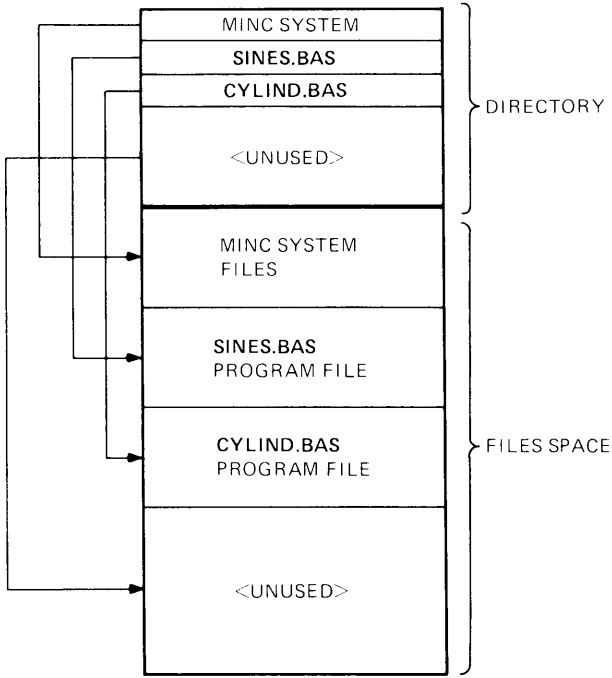
Now the diskette in SY1: holds only two program files and organizationally looks like Figure 3.

Figure 4 represents a system volume on which you saved the SINES program and CYLIND program on SY0:.



MR 1632

Figure 3. Diskette with Program Files



MR 1633

Figure 4. MINC System Volume with Program Files

When you look at the directory of a system volume, you cannot see the MINC system files; however, they are on the volume. Thus, the directory of the volume in Figure 3 would look the same on the screen as the directory of the volume in Figure 4. The only difference in the directory listings is that the number of free blocks is less for the system volume than for the nonsystem volume. After all, the MINC system files use up some of the space on the volume.

### Deleting a Program File

You can delete any file from a diskette with the UNSAVE command. The form of the UNSAVE command is:

UNSAVE filespec

where filespec takes the form:

dev:name.typ

If you leave out the device, MINC defaults to SY0:. If you leave out the file type, MINC defaults to the program file type (.BAS). You can unsave files other than program files by specifying the name and file type. (For more about files other than program files, see Chapter 11.)

For example:

```
UNSAVE SY0:SINES.BAS
```

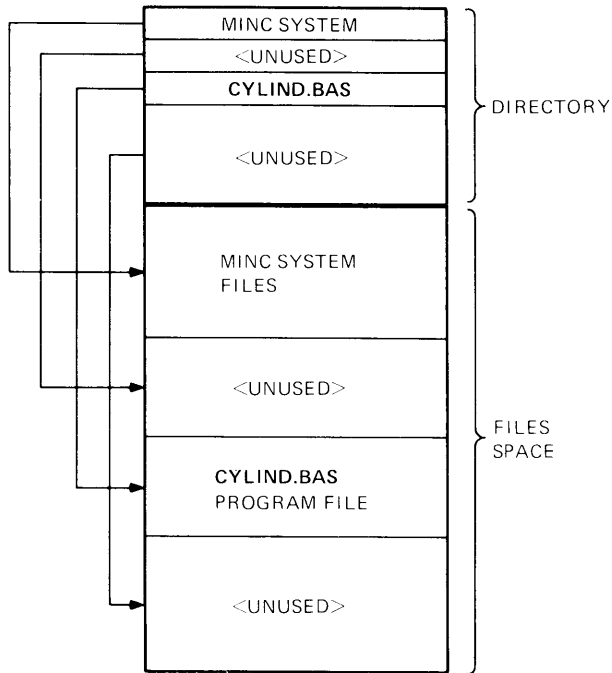
Files take up space (blocks) on a diskette, and eventually they can fill it. Use the UNSAVE command to delete files that you no longer need to store, to make more space.

When you unsave a file, MINC takes the file name out of the directory, and the space where the file was stored becomes immediately available. Figure 5 shows the organization of a system diskette with an unsaved file.

### Recovering Unused Space

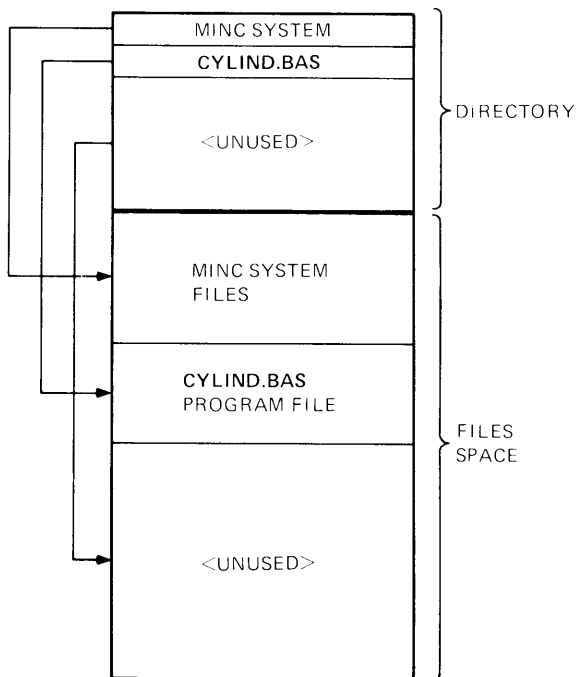
When you unsave a file, the space where the file was stored becomes immediately available. However, if you look at Figure 5, the SINES.BAS file was between the MINC system files and the CYLIND.BAS file. Thus, where the SINES.BAS file was deleted, it left a "hole" in the directory and in the available space. Only programs of the same size or smaller will fit in the holes.

To consolidate all of the unused space into one large area, use the COLLECT command.



MR 1634

Figure 5. Diskette with File Deleted



MR 1635

Figure 6. Collected Diskette

The form of the COLLECT command is:

COLLECT dev:

where dev: is either SY0: or SY1:.

If you collect the diskette shown in Figure 5, graphically it will look like Figure 6.

You can also use COL instead of COLLECT. COL is a valid abbreviation for COLLECT.

#### NOTE

Before you collect a diskette, you should verify it for bad blocks. This procedure is explained later in this chapter.

### Initializing a Volume

Before using a new volume, you must initialize it. When you initialize a volume, MINC formats it — setting up a directory area and an area available for files. Besides putting the volume in the necessary format, initializing it also erases any previous information stored on the volume.

To initialize a volume, make sure you have a system volume in SY0: and the volume to be initialized in SY1:. Then type one of the following commands:

INI SY1:

INITIALIZE SY1:

MINC responds with:

Install volume to be initialized in SY1:, and press RETURN

```
Current volume id:eeeeeeeeeeee
Current owner:   eeeeeeeeeeee
Proceed with initialization (Y or N)?
```

If the volume is new (that is, never used), the e's appear as the "Current volume id" and the "Current owner". If the volume has been used before, the "Current volume id" and the "Current owner" are those names given in the previous initialization. The volume id and owner are a means of ensuring that you are not initializing a valid volume (that is, one with valid information on it).

If you have put a valid volume in SY1: and do not want to proceed with the initialization, type N for no. If you want to proceed, type Y for yes. If you type anything but Y or N, MINC uses N as the default answer and does not initialize the volume.

If you want to proceed, type Y for yes at the question mark. MINC then prompts you for the following information.

Type new Volume id:

The id stands for identifier. Type the identification name that you wish to give the volume. It is recommended that you give the volume a meaningful identifier. A maximum of 12 letters is allowed for the volume identifier. If this volume will be a system volume, you should denote it as such in the volume id. For example, if you are copying one of the Master diskettes you might want to name the copy "mastersystem," and then make more copies of the "mastersystem" diskette for general use. You can answer the question as follows:

Type new Volume id: Lab 3 system

Then, the next question is as follows.

Type new owner name:

If your name is Jane Doe and this volume is your personal volume, you might want to type Jane Doe as the owner name.

Then MINC will initialize the volume. This process takes one or two minutes. Finally, MINC prints the message:

Initialization is complete; found xxx Bad blocks

MINC does not actually type "xxx". It types a 3-digit number representing the number of *bad blocks*. Remember a block is a unit of diskette space. A bad block is a block that MINC cannot use because it has been damaged in some way. Most diskettes do not have bad blocks. You can limit the occurrence of bad blocks by taking good care of your diskettes. However, with normal use, a diskette can develop a bad block after a long period of time through no fault of your own. See Book 1 for the instructions on caring for a diskette.

It is possible to use a volume that has bad blocks. However, if you are a beginner, put volumes with bad blocks aside for now, because the procedure for using them can be more difficult. For

now, use only volumes with "000 Bad blocks." (See the DUPLICATE command in Book 3 for a description of using volumes that have bad blocks.)

A complete example of initializing a new volume follows.

```
READY
ini sy1:
Install volume to be initialized in SY1:, and press RETURN
```

```
Current volume id:eeeeeeeeeeee
Current owner:  eeeeeeeeeeee
Proceed with initialization (Y or N)?Y
Type new Volume id:Lab 3 system
Type new owner name:Jane Doe
```

Initialization is complete; found 000 Bad blocks

Now you can use this new volume to store programs or a system or data.

## Finding Bad Blocks

Some diskettes have or develop bad blocks. You can use a diskette with bad blocks, but the manipulation of them is somewhat difficult.

When you initialize a diskette, MINC finds all the bad blocks and marks them as such in a file called FILE.BAD. If it finds no bad blocks it tells you "000 Bad blocks."

Sometimes a good diskette develops bad blocks. This can happen with ordinary use after a long period of time.

To determine whether a diskette has bad blocks, use the VERIFY command. The form of the VERIFY command is:

VERIFY

where dev: is SY0: or SY1:. If you do not specify the device, MINC verifies SY0:. VER is a valid abbreviation for VERIFY.

For example:

```
READY
VERIFY SY0:
There were no bad blocks found
```

```
READY
```

If the VERIFY command does find bad blocks, it prints a mes-

sage similar to the following example. The only column that gives you useful information is the filename column. In this example, both of the bad blocks are marked in the FILE.BAD file. Thus, they will not affect your use of the diskette.

Bad Blocks	Type	Filename	Rel Blk
414	Hard	FILE.BAD	0
417	Hard	FILE.BAD	0

READY

If one of your diskettes develops bad blocks in one of your files, your file name will appear under the Filename column. For example, if bad blocks develop in CYLIND.BAS, you can recover the file. See the Error Recovery section in Book 3.

If one of your diskettes develops bad blocks in one of the MINC system files, the recovery procedure can be more difficult. See the Error Recovery in Book 3.

You can use the DUP command to copy an entire diskette — including the MINC system files. The form of the duplicate command is:

DUP

You should use the DUP command to keep backup copies of each of your diskettes. You should do this often, at least once a day, because diskettes are fragile.

Use the INI command to prepare a new diskette before using it for duplicating.

The DUP command prompts you to put the diskette to be duplicated in SY0: and the backup diskette in SY1:. The DUP command then copies the entire contents of the diskette in SY0: to the diskette in SY1:, destroying all of the previous contents of the diskette in SY1:.

If you are going to use a used diskette as a backup, you must initialize it first to mark any bad blocks that may have developed since the last initialization. The DUPLICATE command prints the following message if your diskette has not been recently initialized.

?UTILITY-F-Target volume must be newly initialized

## Duplicating a Volume

The following example shows the entire procedure for duplicating a diskette.

INI SY1:  
Install volume to be initialized in SY1:, and press RETURN

Current volume id:eeeeeeeeeeee  
Current owner: eeeeeeeeeeee  
Proceed with initialization (Y or N)?Y  
Type new Volume id:MINC system  
Type new owner name:Student 003

Initialization is complete; found 000 Bad blocks

READY  
verify sy1:  
There were no bad blocks found

READY  
DUP  
Install volume to be duplicated in SY0::

Install initialized, empty volume in SY1:, are you ready (Y or N)?Y

SY1 volume id is: MINC system  
SY1 owner is: Student 003

Do you want to duplicate another volume (Y or N)?N

Re-install system volume in SY0:, and then press RETURN

MINC BASIC V1.2 for the 11/23 (or 11/03)

10-MAY-80  
04:33:40

READY

The DUP command will even duplicate a diskette with bad blocks. Thus, if one of your files develops a bad block, you can recover the file and restore it to good condition on another diskette. For more information on this procedure, see DUPLICATE in Book 3.

## Copying a File

You can use the COPY command to transfer a file from one volume to another or to list the file on a line printer.

The form of the COPY command is:

COPY from-filespec to-filespec

where:

- |               |  |
|---------------|--|
| from-filespec | is the file specification of the file to be copied. The default file type is .BAS. The name of the file to be copied must be present.  |
| to-filespec   | is the file specification of the new file that holds the copy. The default file name is the from-file name. The default file type is the from-file type. The default device is SY0:. If you use LP:, MINC prints the file on the line printer. |

You must specify at least partially both the from-filespec and the to-filespec; that is, specify at least the file name if not the device or file type.

For example, either of the following COPY commands copy CYLIND.BAS from SY0: to SY1: as the program CYLIND.BAS.

```
COPY SY0:CYLIND.BAS SY1:CYLIND.BAS
```

```
COPY CYLIND SY1:
```

The COPY command copies one file at a time.

If you try to copy a file to a filespec that already exists, MINC displays the following message:

```
Output file name is already in use;
do you want to erase the current contents (Y or N)?
```

If you type N, MINC does nothing and returns to the READY prompt. If you type Y, MINC performs the copy, destroying the previous contents of the file.

In MINC you perform this calculation by typing in the following BASIC command:

```
PRINT (27 + 32)*(15-8)/327
```

MINC displays the answer beneath the PRINT statement, like this:

```
PRINT (27 + 32)*(15-8)/327
1.263
```

When you use MINC as a calculator, you are using it in what is called the *immediate mode*. In the immediate mode, MINC performs your instructions as soon as you press RETURN.

The rules for using MINC as a calculator are quite simple and are explained in this chapter.

## THE PRINT STATEMENT

To get MINC to display information on the terminal, you must use the PRINT statement. As you saw above, the form of the PRINT statement is:

```
PRINT expression (RET)
```

where expression represents the number or calculation that is to be printed, and (RET) represents pressing the RETURN key. The expression is printed in blue ink, because it is optional in a PRINT statement. If you omit the expression, MINC prints a blank line.

The following three examples are valid PRINT statements where 7, 23.8, and the result of  $5324.7 + 78625.9$  are to be printed by each PRINT statement respectively.

```
PRINT 7
```

```
PRINT 23.8
```

```
PRINT 5324.7 + 78625.9
```

If you enter each of the above statements, MINC prints the results on the screen as follows:

```
PRINT 7
7
```

## CHAPTER 5

# PROGRAM CONTROL

Until now, all of the programs given in examples have proceeded sequentially. That is, MINC executes each of the programs step by step in order by statement number until the last statement is executed, and then terminates the program. For example, the following program executes statement 20, then 30, then 40, then 50, and then, since there are no more statements, the program stops executing.

```
20 PRINT 'Radius';\ INPUT R
30 PRINT 'Height';\ INPUT H
40 PRINT 'Area'; PI*R^2
50 PRINT 'Volume'; PI*R^2*H
```

This chapter explains how you can gain more control over the order in which MINC executes statements and how to make some statements execute more than once.

IF statements are one method by which you can control the flow of your program. The *flow* of a program is the order in which the statements are executed. In all previous examples, the flow has been sequential — that is, MINC starts executing statements with the lowest statement number and proceeds to those with the highest.

In English, the statement, “If it is sunny, then wash the car,” is a conditional sentence: if the events in the first clause occur (“if it is sunny”), the events in the second clause should follow.

A more complicated if statement is: “If it is sunny, then wash the car; otherwise stay in the house.”

### IF STATEMENTS AND LOGICAL EXPRESSIONS

#### IF Statements in BASIC Are Like English

BASIC has comparable forms of IF statements. One form of the IF statement (comparable to the simpler form in English) is demonstrated below.

```
IF A<B THEN A = A + 1
```

In English this statement means, "If A is less than B, then increment A by 1"; and that is exactly what happens in BASIC.

The IF statement has three slightly different forms:

```
IF logical-expression THEN statement  
IF logical-expression THEN stmt#  
IF logical-expression GO TO stmt#
```

All of these forms of the IF statement are described in the following sections.

Later, programming the "otherwise" clause of an IF statement is explained.

#### NOTE

Do not use IF statements in multiple-statement lines. The outcome of trying to put more than one statement on the same line as an IF statement is not obvious and can actually be quite confusing. See IF statements in Book 3.

### Logical Expressions

A *logical expression* is an expression that can take on one of two values — either the expression is true or it is false. In the previous example, "it is sunny" is a logical expression because it is either true or false. In BASIC, "A is less than B" is likewise a logical expression.

The form of a logical expression is:

```
expression1 relational-operator expression2
```

where expression1 and expression2 are both arithmetic expressions or both string expressions as defined earlier.

Relational operator can be one of six operators that work on expressions and are defined in the following tables.

**Arithmetic Relational Operators** Table 2 shows and explains the six arithmetic relational operators.

Table 2. The Arithmetic Relational Operators

<i>Mathematical Symbol</i>	<i>BASIC Symbol</i>	<i>Example</i>	<i>Meaning</i>
=	=	A=B	A is equal to B.
<	<	A<B	A is less than B.
≤	<=	A<=B	A is less than or equal to B.
>	>	A>B	A is greater than B.
≥	>=	A>=B	A is greater than or equal to B.
≠	<>	A<>B	A is not equal to B.

The symbols =, <, >, <=, >=, and <> are accepted by BASIC but are converted to <=, >=, and <> and are shown in that form in a listing.

The arithmetic relations are the same in BASIC as in mathematics.

The expressions in a logical expression can, of course, be more than one numeric variable. For example:

$(A + 6 * X - 37) / 43 < (C + 19) ^ 2$

Whether this expression is true or false depends on the values of the variables in the expression. The expression could be false at the beginning of a program, and true later, after the values of the variables had changed.

**String Relational Operators** The same six relational operators that apply to arithmetic expressions also apply to string expressions. In string expressions, however, these relations apply to alphabetical sequence. The comparison is done character by character, left to right, on the internal representation of a character (called the *ASCII code*). For example, it is true that:

"ABC" < "ABD"

because "ABC" alphabetically precedes "ABD". MINC compares the first character from each expression; if they are equal, then MINC compares the second characters. This process continues until a character differs or until the strings match.

Because strings can also contain numbers and punctuation, each character, whether it is alphabetic, numeric, or other, has an ASCII code. The characters have a *collating sequence* that tells the "alphabetical order" of all the characters depending on

the value of each character's ASCII code. The collating sequence and the ASCII codes are given in Appendix A.

For example, it is true that:

"ABC" < "abc"

because the ASCII codes for the capital letters are lower than the ASCII codes for the lower case letters.

In any string comparison, MINC ignores trailing blanks (that is, "ABC" is equivalent to "ABC "). However, MINC does not ignore preceding blanks. Table 3 shows and explains the string relational operators.

Table 3. The String Relational Operators

<i>Operator</i>	<i>Example</i>	<i>Meaning</i>
=	A\$=B\$	The strings A\$ and B\$ are alphabetically equal.
<	A\$<B\$	The string A\$ alphabetically precedes B\$.
>	A\$>B\$	The string A\$ alphabetically follows B\$.
<=	A\$<=B\$	The string A\$ is equivalent to or precedes B\$ in alphabetical sequence.
>=	A\$>=B\$	The string A\$ is equivalent to or follows B\$ in alphabetical sequence.
<>	A\$<>B\$	The strings A\$ and B\$ are not alphabetically equal.

## IF/THEN Statements

One form of the IF/THEN statement was demonstrated in the first section of this chapter. The two forms of the IF/THEN statement are:

IF logical-expression THEN statement  
 IF logical-expression THEN stmt#

Examples of the first form are:

```
10 IF B<=23 THEN B=B+1
```

```
10 IF B^2-4*A*C < 0 THEN PRINT 'ROOTS IMAGINARY'
```

```
10 IF A$<>B$ THEN F=1
```

```
10 IF C$>B$ THEN IF A$=B$ THEN S$=C$
```

In the first example, if it is true that B is less than or equal to 23, then B is incremented by one. If B is greater than 23, then MINC passes on to the next statement in the program (which is not listed here). In the final example, a second IF statement follows the THEN statement, showing that IF statements can be nested.

A program example using this form of the IF statement follows:

```

10 REM* This program prints the square root
12 REM* Of positive inputs. It flags
14 REM* Negative values.
16 REM*
20 PRINT 'Number';INPUT N
30 IF N<0 THEN PRINT 'Square root of negative # not allowed'
40 IF N>=0 THEN PRINT 'The square root is'; SQR(N)
RUN

```

```

SQR T           20-APR-80           4:15:27

```

```

Number? 4
The square root is 2

```

```

READY
RUN

```

```

SQR T           20-APR-80           4:15:10

```

```

Number? -2
Square root of negative # not allowed

```

Examples of the second form are:

```

100 IF A$<B$ THEN 50

```

```

100 IF X + 3 >= 18 THEN 200

```

```

100 IF 52 <= X THEN 25

```

The second form of the IF statement says, "If the logical expression is true, then *transfer control* to the statement whose number is after the THEN." MINC has control over the execution of the program. IF statements cause MINC to transfer control from the IF statement to the statement whose number is after the THEN. In the first example, if A\$ is less than B\$, then the next statement executed will be statement 50. If A\$ is not less than B\$, then the next statement executed will be the statement that immediately follows line 100.

## PROGRAMMING FUNDAMENTALS

A program example using the second form of the IF statement is:

```
10 REM - This program sorts 3 input strings in
15 REM - alphabetical order.
20 PRINT 'Input 3 strings'
30 INPUT A$, B$, C$
60 REM - test if they are in alphabetical order.
70 IF A$<B$ THEN 110
75 REM - swap them.
80 S$=A$
90 A$=B$
100 B$=S$
110 IF A$<C$ THEN 150
115 REM - swap them.
120 S$=A$
130 A$=C$
140 C$=S$
150 IF B$<C$ THEN 190
155 REM - swap them.
160 S$=B$
170 B$=C$
180 C$=S$
190 PRINT 'The sorted list is'
195 PRINT A$
200 PRINT B$
210 PRINT C$
RUNNH
```

```
Input 3 strings
? hello
? goodbye
? always
The sorted list is
always
goodbye
hello
```

Notice that a third “temporary” variable is needed for a swap. You cannot swap A\$ and B\$ by saying:

```
A$=B$B$=A$
```

because of the nature of an assignment statement.

```
A$=B$
```

*erases the previous value of A\$ and puts the value of B\$ there. Consequently, the original value of A\$ is lost. In this example, S\$ saves the original value of A\$ while B\$ is stored in A\$.*

The form of the IF/GO TO statement is:

## IF/GO TO Statements

IF logical-expression GO TO stmt#

and MINC executes it exactly like the IF logical-expression THEN stmt# statement. The previous example that prints out three strings in alphabetical order would work the same if you replace THEN with GO TO. GO TO is perhaps clearer in this type of IF statement because the term GO TO itself implies transfer of program control.

The sorting example program is actually an example of programming the "otherwise" portion of an IF statement. For example, the following program segment says, "If A\$ is less than C\$ then compare B\$ and C\$; otherwise swap A\$ and C\$."

## Programming the "Otherwise"

```
110 IF A$<C$ GO TO 150
115 REM --- otherwise swap A$ and C$
120 S$=A$
130 A$=C$
140 C$=S$
150 IF B$<C$ THEN ...
```

Statements 120, 130, and 140 are all part of the "otherwise."

GO TO statements can be used anywhere in a BASIC program to transfer control to another statement.

## GO TO STATEMENTS

Upon execution of an unconditional GO TO statement, MINC transfers control to the statement-number in the GO TO statement. This form of GO TO statement is called *unconditional* because control is always transferred when the statement is executed, as opposed to the IF/GO TO statement, which is a conditional GO TO. The form of the unconditional GO TO is:

## Unconditional GO TO Statements

GO TO stmt#

A program using an unconditional GO TO was shown in Chapter 3 for computing sines and cosines. The following statement numbers are the same as those used in Chapter 3. Some REMARK statements have been added to help you understand the program.

```
5 REM - This program computes sines and cosines
10 REM - of any angle given in degrees.
20 PRINT 'Sine', 'Cosine'
30 PRINT 'Degrees';
35 REM - Input angle in degrees.
40 INPUT X
55 REM - Compute radians from angle.
60 Z=X*PI/180
70 PRINT ,SIN(Z), COS(Z)
75 REM - Repeat for next angle
80 GO TO 30
```

At statement 80, control always passes back to statement 30 because of the unconditional GO TO.

### ON/GO TO Statements

Unconditional GO TO statements transfer control to a particular statement. ON/GO TO statements transfer control to one of several statements depending upon the value of an expression. Thus, ON/GO TO statements are a form of conditional GO TO. The other form of a conditional GO TO is the IF/GO TO statement.

The form of the ON/GO TO statement is:

ON numeric-expression GO TO list-of-line-numbers

For example:

```
10 ON X GO TO 100,200,300
```

If X is equal to 1, control transfers to statement 100. If X is equal to 2, control transfers to statement 200. And, if X is equal to 3, control transfers to line 300. If X is less than 1 or greater than 3, then MINC prints the following error message.

?MINC-F-Value of control expression is out of range at line 10

Line 10 is the line number with the ON/GO TO statement.

An example of using ON/GO TO statements is given in Chapter 7.

### RESEQUENCING PROGRAMS WITH GO TOs

When you use the RESEQ command to renumber your statements, MINC also renumbers any references to these statements. That is, all line numbers referenced in GO TO, ON/GO TO, IF/THEN, and IF/GO TO statements are changed to reflect the new numbering. However, MINC does not change any references to statement numbers within remarks.

Note that if you change statement numbers yourself, you must be careful to change any references to these statements.

Obviously, every program must terminate at some time. Programs can terminate in any of the following ways:

## PROGRAM TERMINATION

- Normally, by executing the highest numbered statement.
- Normally, by executing a STOP or END command (explained in the following sections).
- Abnormally, by encountering a fatal error (denoted by messages beginning with ?MINC-F-).
- Normally or abnormally, by someone's pressing CTRL/C.

Usually you want your programs to terminate normally — that is, under the program's control. One of the best ways to terminate a program normally is to make sure that MINC executes a STOP or an END statement. These statements are defined in the following sections.

An END statement when present must come as the very last statement in the program. When control passes to the END statement, the program terminates.

## END Statements

Programs will terminate by default by "falling out the bottom," that is, by looking for the statement after the last statement. However, the use of an END statement makes it clear that the programmer definitely wanted to end the program at a certain point — that no statements were forgotten at the end. Especially in a program where control must pass to a statement that terminates the program, END statements are clearer and better.

Like the END statement, the STOP statement terminates execution of a program. However, the STOP statement can be placed anywhere in a program. The STOP statement also prints a message giving the line number of the STOP statement that terminated execution.

## The STOP Statement

In MINC you perform this calculation by typing in the following BASIC command:

```
PRINT (27 + 32)*(15-8)/327
```

MINC displays the answer beneath the PRINT statement, like this:

```
PRINT (27 + 32)*(15-8)/327
1.263
```

When you use MINC as a calculator, you are using it in what is called the *immediate mode*. In the immediate mode, MINC performs your instructions as soon as you press RETURN.

The rules for using MINC as a calculator are quite simple and are explained in this chapter.

## THE PRINT STATEMENT

To get MINC to display information on the terminal, you must use the PRINT statement. As you saw above, the form of the PRINT statement is:

```
PRINT expression (RET)
```

where expression represents the number or calculation that is to be printed, and (RET) represents pressing the RETURN key. The expression is printed in blue ink, because it is optional in a PRINT statement. If you omit the expression, MINC prints a blank line.

The following three examples are valid PRINT statements where 7, 23.8, and the result of  $5324.7 + 78625.9$  are to be printed by each PRINT statement respectively.

```
PRINT 7
```

```
PRINT 23.8
```

```
PRINT 5324.7 + 78625.9
```

If you enter each of the above statements, MINC prints the results on the screen as follows:

```
PRINT 7
7
```