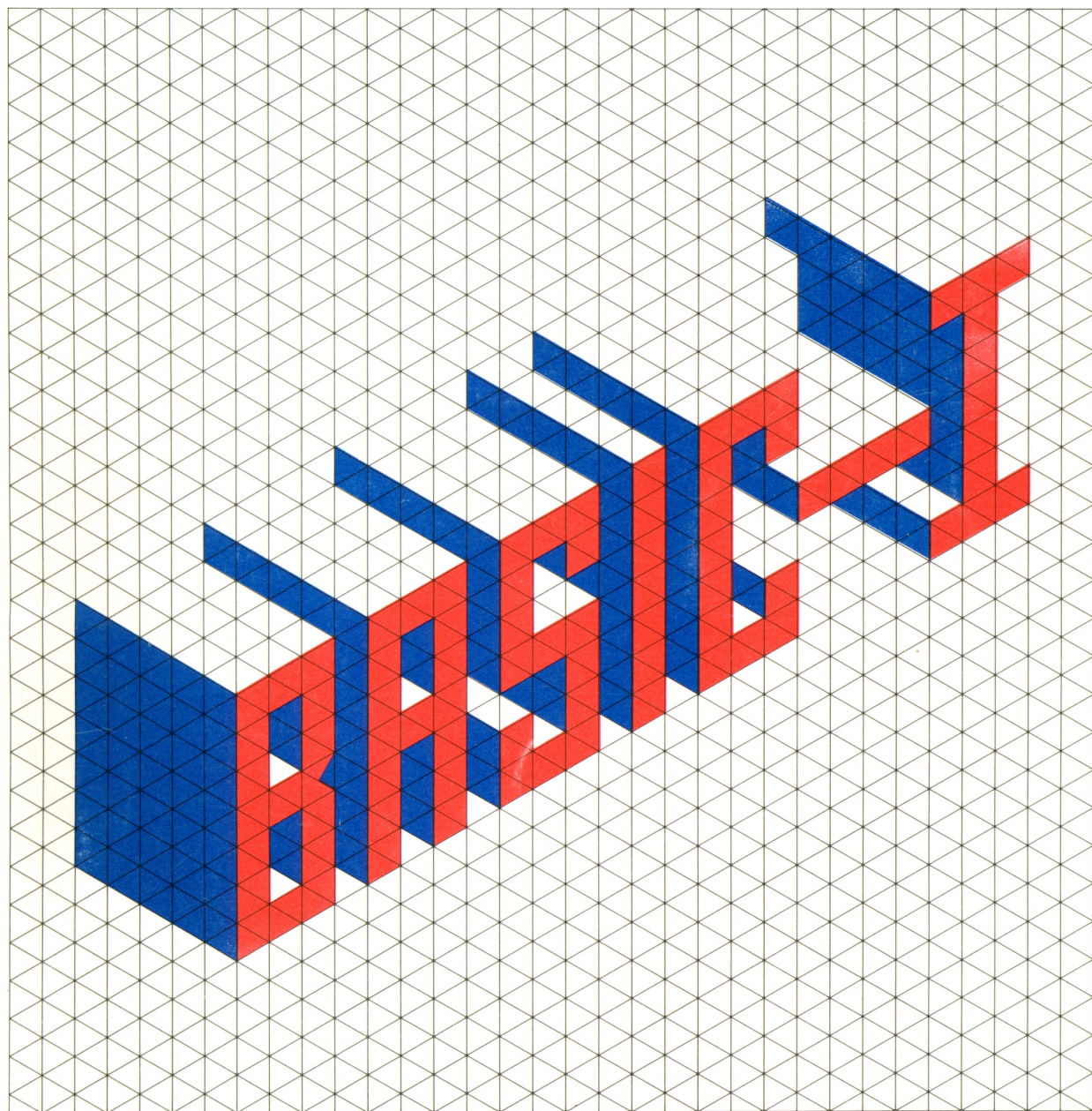


SORD

Creative Computer

m5 BASIC-I Manual



Easy BASIC for Beginners

NOTICE:

- Throughout this manual, the command prompt as it appears on the screen is shown as an "A", indicating the letter or alphabet mode. Due to a last minute design change, the command prompt for this mode has been changed to a "L". It will appear on your screen as shown below.



- Press the FUNC and "2" keys simultaneously for caps lock.

Copyright © 1983 by SORD COMPUTER CORPORATION

All rights reserved. Printed in Japan.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of SORD COMPUTER CORPORATION

INDEX

Chapter 1	Introduction	4
Chapter 2	Getting Started In BASIC	
	2-1 Getting Started in Basic.....	6
	2-2 The Keyboard	7
	2-3 Display Characters on the Screen.....	7
	2-4 Scrolling Function	9
	2-5 Repeat Function	10
	2-6 Delete Function	10
	2-7 Editing Function	12
	2-8 CTRL Key Function	13
	2-9 Drawing Graphics	16
Chapter 3	What Can I Do with BASIC?	
	3-1 Let's Do Some Calculations.....	18
	3-2 Important Notes about Calculations	19
	3-3 Error Messages	20
	3-4 Continuous Calculations.....	20
	3-5 Other Usages of the PRINT Command	21
Chapter 4	What Is a Variable?	
	4-1 What Is a Variable?.....	22
	4-2 Letter Variables.....	22
Chapter 5	What's in a Program?	
	5-1 Steps in Making a Program.....	24
	5-2 Let's Make a Program	24
	5-3 Let's Look at the Program Again	25
	5-4 Let's Make It Look More Like a Program.....	26
	5-5 Line Numbers	29
	5-6 Corrections.....	30
	5-7 Using the INPUT Command in Another Way	32
	5-8 Using Messages to Make Your Program More Understandable	35
	5-9 Other Usages of the LIST Command	37
Chapter 6	How to Save a Program	
	6-1 Saving Your Program.....	38
	6-2 Verifying Your Program	39
	6-3 Loading Your Program.....	41
Chapter 7	Various Commands	
	7-1 Various Commands.....	44
	7-2 Endless Loop	44
	7-3 Looping Three Times.....	46

Chapter 8

Dice Game

8-1 Game Program48
8-2 RND Function48
8-3 "Computer" and Dice Game49
8-4 Saving Frequently Used Code as
Subroutines51
8-5 How Many Times Did Each Number Come Up?.....54
8-6 Array Variables.....55
8-7 Reading Data in Your Program.....57

Chapter 9

What's a Character String?

9-1 What's a Character String?.....60
9-2 Playing with Character Strings60
9-3 Counting Characters62
9-4 Looking At Only a Portion of a Character String.....63
9-5 Jumping Around in Our Character String Program.....64
9-6 Hints to Understand Your Program Better.....65
9-7 An Easy Software Times66
9-8 Printing Characters on the Screen.....66
9-9 Using the TAB Function68
9-10 Various Character String Functions.....69

Chapter 10

Dice Graphics

10-1 Dice Graphics.....70
10-2 Rolling Dice74
10-3 Coloring the Die Face.....74

Chapter 11

Conclusion

.....76

Appendix

A UFO Game
B Color Codes
C Character Codes
D Commands
E Functions
F Control Codes
G ASCII Character Codes
H Error Messages
I Mathematics Tutor Program

CHAPTER
1

CHAPTER
2

CHAPTER
3

CHAPTER
4

CHAPTER
5

CHAPTER
6

CHAPTER
7

CHAPTER
8

CHAPTER
9

CHAPTER
10

CHAPTER
11

1. INTRODUCTION

BASIC is a computer language. Pretty simple. BASIC allows anyone to communicate with the M5 computer. Still simple. We're going to learn how to use BASIC. Still sound simple? It is.

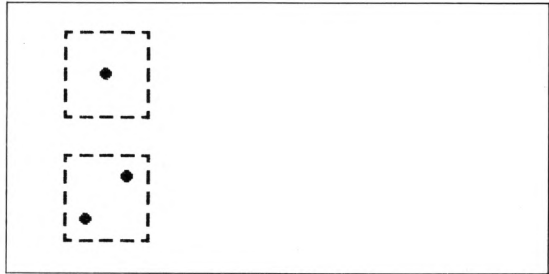
This version of BASIC, BASIC I, which leaves out some difficult and complicated commands, is designed for users who have never before used BASIC or who have just started to learn programming.

But that doesn't mean it isn't useful. Try it out! Many things can be done with BASIC I. Games. Math. Keeping track of calorie intake. It can be fun for the whole family!

BASIC I is basically a subset of a larger set of BASIC commands. When you master this subset, you can progress to the next step. Don't worry about having to know all kinds of commands or computer talk. Just relax. The only thing you should know at this point is that commands tell the M5 computer what to do.

Before you master BASIC, you'll need to learn how to use the keyboard and how to write programs. As the old proverb goes, "Practice makes perfect". We hope you will become accustomed to BASIC commands soon.

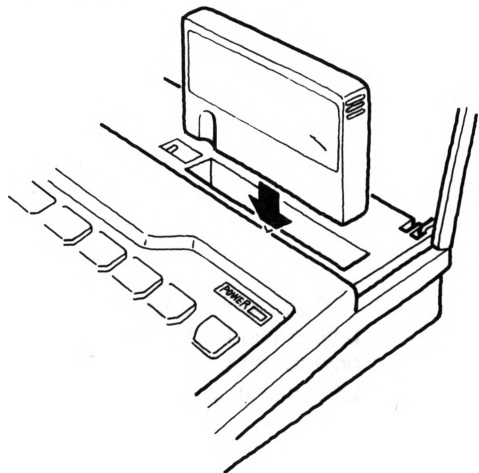
This manual is written in everyday English with clear examples so that even older school children can understand it easily. This makes it ideal for parents and kids to study together. OK! Let's get ready to have some fun with the M5! But what can we do with BASIC I? Well, we're going to explore some simple examples and have some fun with some games that'll help us to understand BASIC I better. One game we'll explore, add to and learn from is a dice game.



If you look in Appendix A, you'll find a UFO game. Wait just a little before you try to use it. First, go through this manual. It'll tell you exactly how to get started the right way. And by the time you get ready to enjoy the UFO game, you'll understand exactly what to do and why you're doing it.



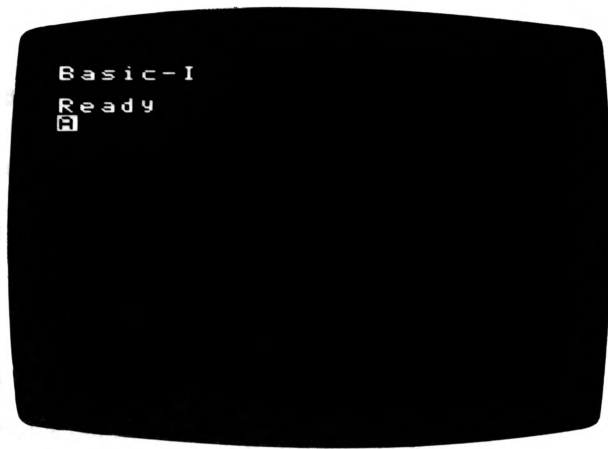
We're ready to get started. Or are we? Look at the type of cartridge inserted into the M5 console. In order to work with BASIC I, you must insert the BASIC I program cartridge into the console. But first turn off power to the computer. In fact, you should always turn power off before inserting or taking out a cartridge. Otherwise, you might have problems. For the first several times especially, be very careful when you insert or pull out the cartridge because the connector can be damaged. Be sure to insert the cartridge with the BASIC I label facing you. Push firmly.



After you master BASIC I, you can proceed to BASIC G which has advanced BASIC commands for games and graphics, or to BASIC F for scientific calculations.

2-1. Getting Started in Basic

After connecting the console to your television set and inserting the BASIC I program cartridge, turn the power unit ON. Doing this will supply power to the M5 console. At this point, the M5 system is ready to input and run programs. The "BASIC I Ready" message should be displayed at the upper left hand corner of the screen. But if the system has not been connected properly, obviously it will not function correctly. So if the ready message is not displayed, immediately turn off the power unit and check the connection between the television and M5 console. Also check that the BASIC cartridge is plugged in correctly.



The "Basic-I Ready" message is the first indication from the system that it is ready to run BASIC I programs.

If the "A" cursor displayed on the left-hand side of the screen is difficult to see, execute the following command (when power is on, the M5 is in the GI mode):

GI, GII Multicolor mode
VIEW 1 , 0 , 31 , 23 RETURN

Text mode

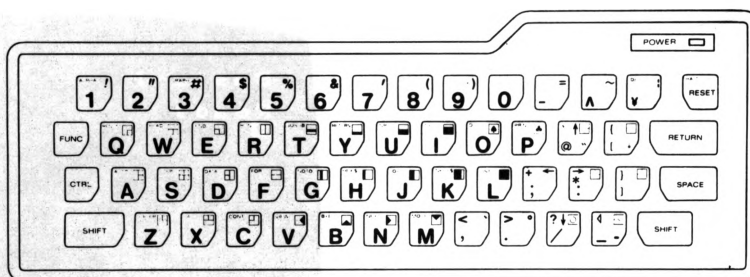
VIEW 1 , 0 , 39 , 23 , RETURN

Pushing the CTRL key and the F key simultaneously will cause the screen display area to shift to the right by one character space. If this is done after the VIEW command has been input, but before the return key is pressed, the entire display will be shifted to the right by one character, and no characters previously displayed in the "0" position will remain.

2-2. The Keyboard

Now look at the M5 console. What you'll notice right away is the typewriter-like keyboard. But don't assume it is exactly identical to a typewriter. Eventually, you'll learn that there are several keys that really increase the flexibility and power of the system.

Each key on the keyboard either displays a letter, number, character symbol, graphics symbol, function, or a combination of these. We'll go through and learn how all of these symbols and characters are used. Relax. We'll take it nice and easy. Before you realize it, you'll be composing your own custom made programs and taking advantage of the M5's processing power.



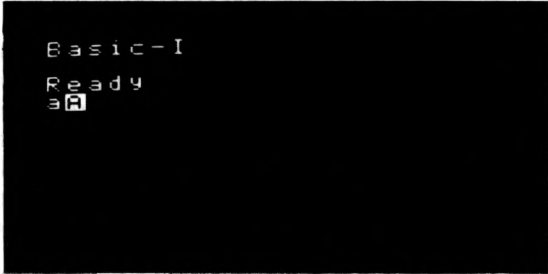
M5 Keyboard

2-3. Displaying Characters on the Screen

What's a cursor? Look on the screen and you'll see the character 'A' blinking. That's the cursor. It's used to indicate the place on the screen where new characters are input by you.

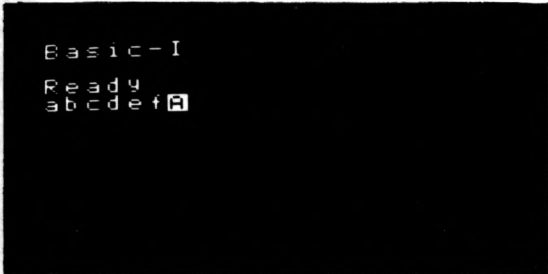
```
Basic-I
Ready
A
```

Try pressing a number or one of the letters on the keyboard. You'll then see it displayed on the screen. Notice it's displayed at the cursor's previous position. Also, the blinking cursor moves one position to the right and a "beep" is heard.



```
Basic-I  
Ready  
a
```

Try pressing a couple more letters and numbers. You'll see all the keys you pressed displayed on the screen one after another. You can see that the cursor stays ahead of your input by one position. It marks where your next input character will be displayed.



```
Basic-I  
Ready  
a b c d e f
```

The M5 system has many powerful functions. If you press a key we have not yet discussed, it may make it impossible for you to handle the next operation in this manual. If this warning comes a bit late and you can't make the system perform as you like, turn the switch on the power unit OFF and back ON. This resets the system and allows you to start over. Note that this is not always good practice since the system forgets what it was previously doing. But when you become familiar with the system, this won't be a problem. In the meantime, we recommend that you closely follow this manual.

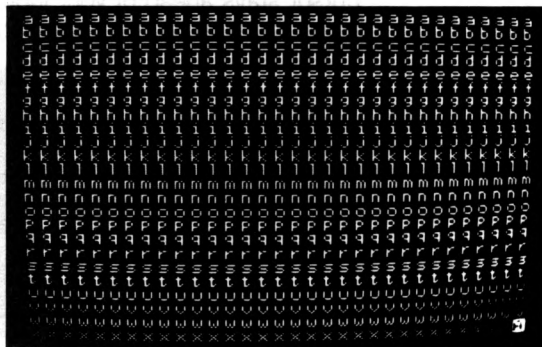
2-4. Scrolling Function

The numbers zero through nine are displayed on the uppermost line of the keyboard while the letters are arranged in the three lower rows, just like a typewriter. Press all of these keys. When enough keys have been pressed to fill up one line, the cursor automatically falls off the end of the current input line and ends up at the leftmost position of the next line. This happens for every new input line.

Obviously, since there are 24 input lines, the screen quickly runs out of space. In this case, the cursor never leaves the screen. Instead, the cursor falls off the end of the current input line as before. But the uppermost line moves up one line and disappears from the screen. Don't worry. The computer hasn't forgotten about it. It's retained in the M5's memory. You'll see the cursor displayed at the leftmost position of the new input line which is now the bottom line of the screen. The same thing will occur every time a new line is input.

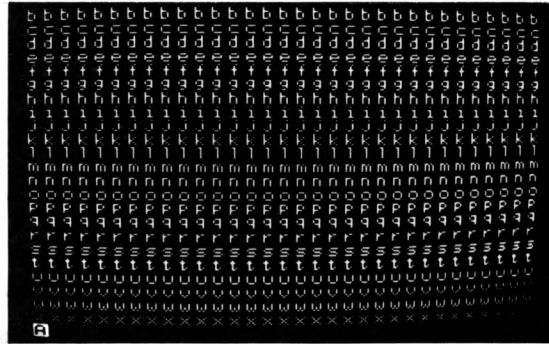
This function is called "scrolling."

The figure below shows a screen that has been completely filled with input. The cursor is sitting at the lower right hand corner of the screen.



Full screen of input

Now notice after inputting one more key, the topmost line scrolls off the screen and the cursor moves to the leftmost position of the next input line. The next input line then becomes the current input line.



Automatic scrolling

2-5. Repeat Function

Next, try pressing the same key again and again. Now keep pressing down on the same key. Notice the effect is the same. In other words, constant pressure on a key is the same as repeatedly pressing that key resulting in a constant stream of the same key displayed on the screen. It'll stop once you take pressure off the key.

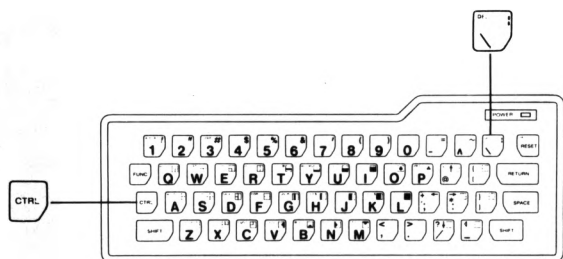
This function, for obvious reasons, is called the "repeat function." The repeat function, together with the CTRL key at the lefthand side of the keyboard provides you with another function. It'll be explained later.

2-6. Delete Function

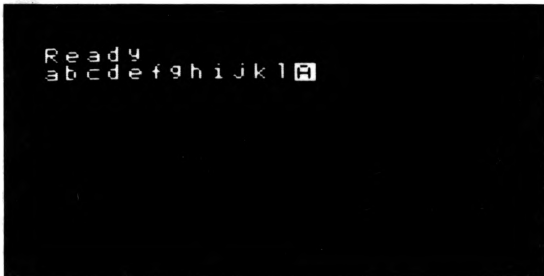
Up to this point, we've only discussed how to display letters and numbers on the screen. Now we'll describe how to fix incorrect letters and numbers already displayed on the screen.

For this, we'll conveniently invoke the "delete function." So press the CTRL key at the lefthand side of the keyboard with your left hand. Keeping the CTRL key pressed, press the DEL key situated at the upper right side of the keyboard with your right hand. (If you have trouble finding the DEL key, note DEL is indicated at the upper left of the key in small capital letters.) Pressing this combination of keys at the same time results in the cursor stepping one position to the left and erasing the character that was previously displayed there. To delete several characters, press the DEL key for each character you want to delete.

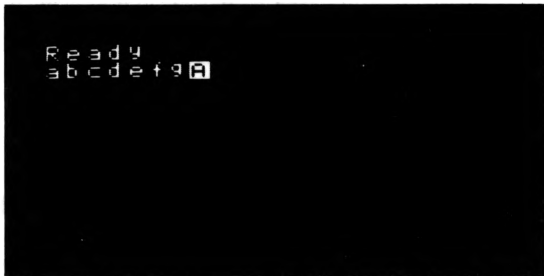
Now stop pressing the CTRL and DEL keys and type in the correct information. Each time you input a new character, the cursor will then move to the right as before and the new character is displayed on the screen.




Invoking the Delete function with the CTRL and DEL keys''



Deletion from end of text




The delete function can also be used in the middle of a line of characters. In this case, the cursor remains where it is and all characters to the right move one position left. The text is shortened.



```
Ready
abcde fghijkl
```

Deletion in the middle of the text



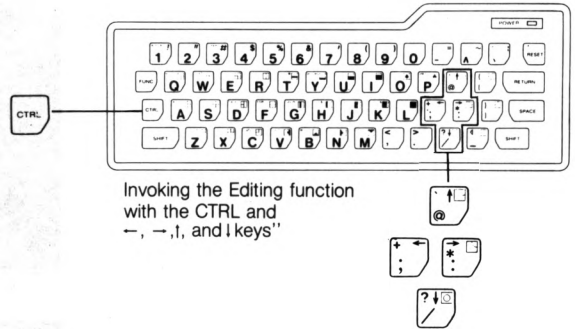
```
Ready
abcde hijkl
```

2-7. Editing Function

What do you do if you find a wrong character many positions back on the screen? It would be very tedious to backtrack using the CTRL and DEL keys. To introduce a new function, again press the CTRL key with your left hand. Now look to the right side of the keyboard. In the upper left corner of four keys are displayed ←, →, ↑, and ↓. These are known as cursor control keys. Try pressing the ↑ key. (Don't forget to leave the CTRL key pressed.) Looking at the cursor, notice it's moved one line up and remains on the same column. Next, try pressing the ← key while maintaining pressure on the CTRL key. The cursor moves to the left of the same line, but the character that sits where the cursor was previously displayed is still there. To sum up cursor control keys, they allow you to easily move the cursor in any direction while not changing any of the displayed information.

Take your left hand off the CTRL key and press a letter key. You'll see the letter displayed in place of the last displayed character. Using this combination, it's easy to see how any character displayed on the screen can be quickly and easily corrected.

This is known as the "editing function."



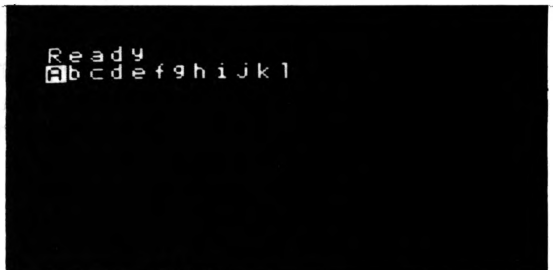
2-8. CTRL Key Functions

If you're itching to get at more powerful functions, relax. We'll describe several very useful functions that also use the CTRL key.

Before going on, move the cursor to the leftmost position of the current input line. Pretty boring if the cursor was near the right margin. But there is an easier way. For this exercise, now move the cursor somewhere near the middle of the current input line. While keeping the CTRL key pressed, press the B key. The cursor is automatically moved to the leftmost position of that input line. Now you can type in new characters for that line if you like. Try it.

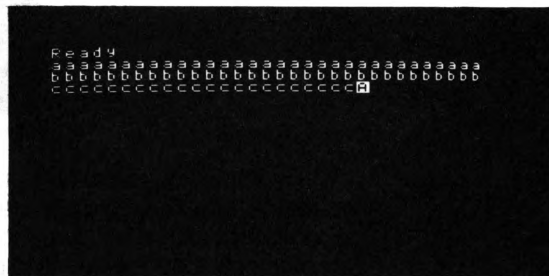


Before Using the CTRL B Function

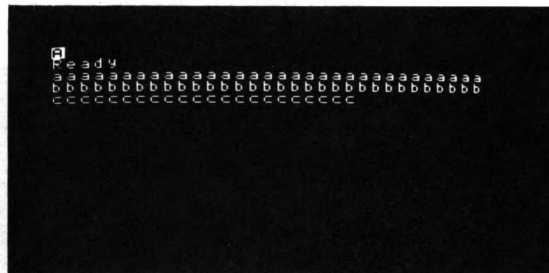


After Using the CTRL B Function

The upper left corner of the screen is also known as the “home” position. Similar to the CTRL and B key combination which moves the cursor to the leftmost position of the current input line, the CTRL and K key combination moves the cursor to the home position. Don’t forget to press both keys at the same time. Try it.

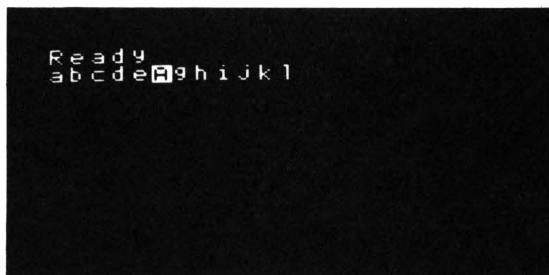


Before Using the CTRL K Function





After Using the CTRL K Function

You may sometimes want to delete many characters at one time. To see how this can be easily done, move the cursor to a position anywhere on the input line. We’ll practice how to delete all characters beginning at the cursor and to the right of it. Keep the CTRL key pressed and press the X key at the same time. Pretty easy. All these characters are deleted. Likewise, a complete line can be deleted by positioning the cursor to the leftmost position and using the CTRL and X key combination.



Before Deleting Characters



```
Ready
abcde
```

After Deleting Characters

Many times you will want to delete only a few characters somewhere in the middle of a line. This can be easily done by moving the cursor to the position from where you want to begin deleting. You can delete by one of two methods. One method is to push the SPACE key as many times as desired; however, this will leave spaces where the key was pressed. If you don't want the spaces, then use the DEL key along with CTRL key as many times as needed to delete the characters and close the text up.



```
Ready
abcde  ijkl
```

Using the SPACE key to delete characters

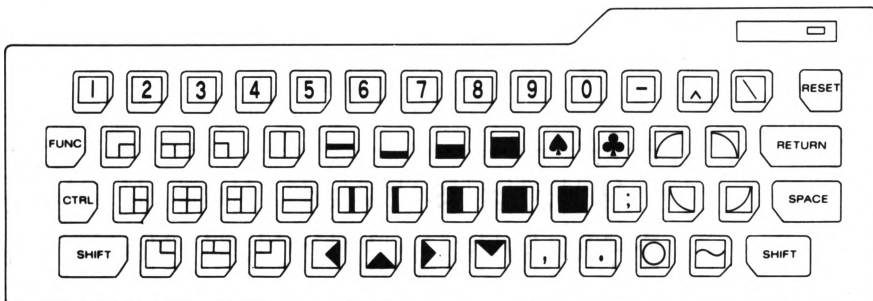
2-9. Drawing Graphics

Up to now, you've learned only how to input, correct and delete alphabet letters, numbers, symbols and signs. But the M5 gives you the capability to draw graphics. Let's learn how to do it.

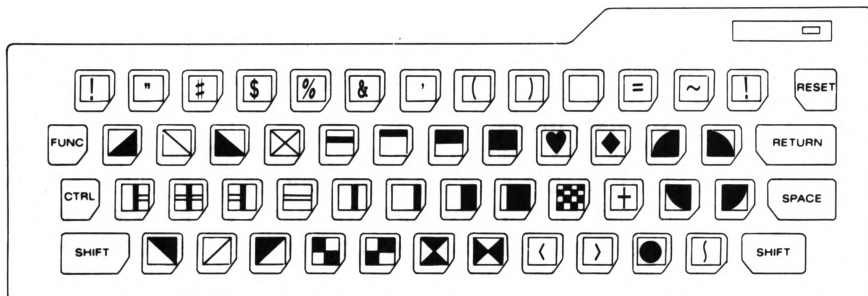
• Graphics Mode

To get into the graphics mode, push the FUNC key and the 3 key together. You will notice that GRAPH is written in small letters above the number 3. The cursor now blinks with the letter 'G'. In this mode, you can draw graphics using the symbols highlighted in light blue on the keyboard and also other symbols if the SHIFT key is pressed. Try drawing some pictures. It will probably be easier if you use the ←, →, ↑, and ↓ keys as described in the Editing function section.

Notice that the first row of characters in the shifted position is identical to that in the alphabet mode.



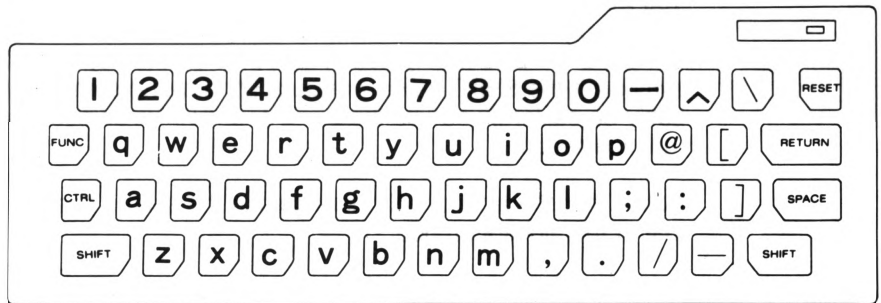
Graphic characters available with SHIFT key not pressed



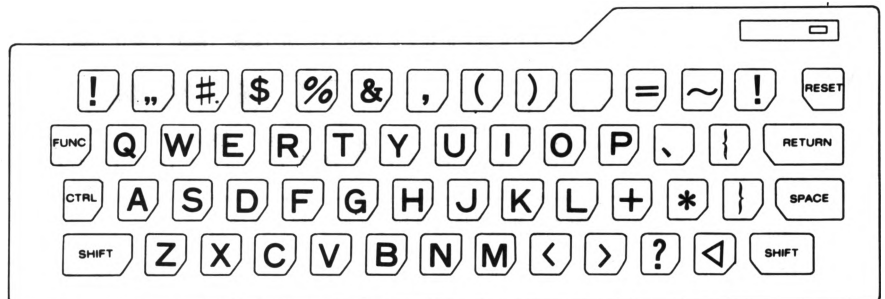
Graphic characters available with SHIFT key pressed

• Alphabet Mode

To get into the alphabet mode, press both the FUNC key and the 1 key together. You will notice that SMALL is written in small letters on the top part of the number 1 key. The cursor now blinks with the character A as you have seen earlier. Remember that this is the mode that you are in when you first power on the M5.



Alphabet characters with SHIFT key not pressed



Alphabet characters with SHIFT key pressed

• Capitals Mode

When you want to type using capital letters only, you could hold down the SHIFT key but it is more convenient to use Capitals mode. Press the FUNC key and the 2 key together. The cursor will change to the letter C, and all letters typed will appear in upper case. Press FUNC and 1 together to return to lower case mode.

Well, you've learned how to type alphabet letters, characters, graphic symbols, and signs easily on the console. You've also learned how to correct and delete them. These techniques are very important and basic to programming in BASIC I. Please be sure that you become familiar with all the operations that have been described so far.

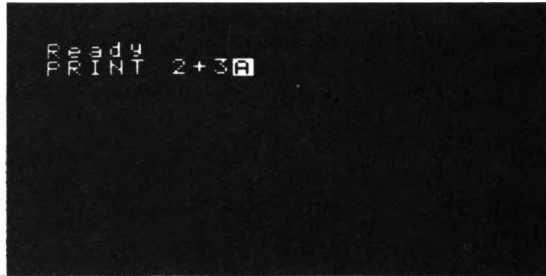
3-1. Let's Do Some Calculations

First of all, turn the power off and on to erase the screen.

•ADDITION

Let's add $2 + 3$. Normally, you would write $2 + 3 =$ but in BASIC I this is not the format in which to do a calculation and print a result. BASIC I has some special rules and commands that you must learn and adhere to. Don't worry, they're quite easy and pretty soon you'll be able to exploit the full capability of the M5.

For this addition, you'll need to use the PRINT command to obtain the result. Type PRINT $2 + 3$. The + symbol is located on the ; key, so, you'll need to press both the SHIFT and ; keys at the same time.



Why wasn't the = sign used? Well, in BASIC I, this will not cause the result to be printed. The cursor is positioned after the 3 and is still blinking. How do we get the answer to be printed out? Well, you need to press the RETURN key. Look what happened. The computer printed out the answer and the Ready message on a new line. This message indicates that the M5 is ready to accept input again.

Try adding some more numbers. Note: You may type PRINT in small letters.

•SUBTRACTION

Use the - sign to perform subtractions. Try doing some subtractions.

For example:

$23 - 14$
 $13 - 35$

•MULTIPLICATION

In BASIC I, the * symbol (press both the SHIFT and : keys together) is used instead of the \times sign.

Try some of these examples.

123×34
 57×67

•**DIVISION**

In BASIC I, the / symbol (press the / symbol located on the right side of the last row) is used instead of the \div sign.

Try these examples

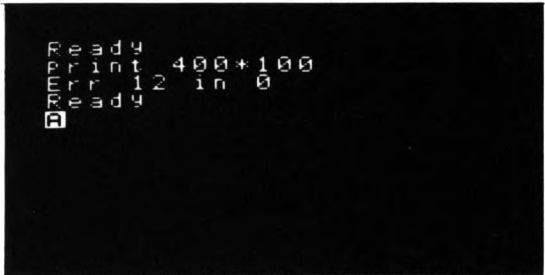
$100 \div 10$
 $1072 \div 4$

3-2. Important Notes about Calculations

•**OVERFLOW**

In BASIC I, only values in this range — 32767 to 32767 is recognized. Therefore, if you perform a calculation that yields a result outside this range, you will get an error message.

For example, try multiplying 400×100

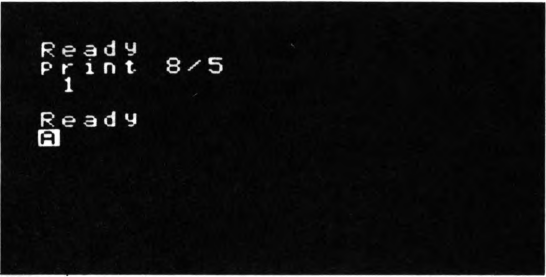


```
Ready  
print 400*100  
Err 12 in 0  
Ready  
A
```

The answer that should be printed is 40,000 but you get this message 'Err 6 in 0' instead. This indicates an overflow error.

•TRUNCATION

Only integer values are recognized. Therefore, if you try to divide two numbers, say, $8 \div 5$, the answer that will be printed is 1 and not 1.6. Essentially, all fractional results are truncated to the lower integer result.



```
Ready
Print 8/5
1
Ready

```

3-3. Error Messages

BASIC I checks the command syntax for errors. If there is one, an error message is printed. For a detailed description of the error messages 'Err1 ~ Err18', refer to page 97.

3-4. Continuous Calculations

In normal programming, you will do many continuous calculations. The order in which calculations are performed is the same as what you've learned in school. Basically, all operations within parentheses are performed first. Operations are performed left to right according to the following precedence levels: multiplication (highest), division, addition, then subtraction (lowest). Please remember that the final result must be within this range, -32767 to 32767 .

Try the following calculations and as many as you would like to try.

```
12 * 5 - 18 / 9 = PRINT 12 * 5 - 18 / 9 RETURN
(2 * 4) * (6 - 3) = PRINT (2 + 4) * (6 - 3) RETURN
3 + 2 ^ 4 = PRINT 3 + 2 ^ 4 RETURN
```

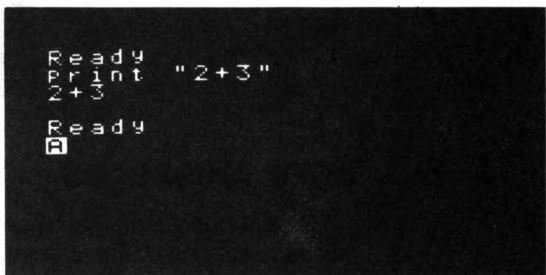
↑
The ^ symbol signifies exponentiation.

How did the calculations come out?
Were you able to use the PRINT command easily?

The PRINT command is one of the simplest BASIC I commands to know and use, especially, since it gives you your answer on the screen. If you omit the print command and just enter a calculation, say $2 + 3$ and the RETURN key, then nothing will happen. The PRINT command is necessary.

3-5. Other Usages of the PRINT Command

The PRINT command can also be used to print text. Just enclose your text within quotation marks. For example, if you had wanted to print $2 + 3$, you enter this `PRINT "2+3"` RETURN. Try it.



```
Ready  
Print "2+3"  
2+3  
Ready  
␣
```

Printing text

Try some of your own examples and, if you want, try these here

```
10 PRINT "BASIC-I"  
20 PRINT "♣♣"  
30 PRINT "How do you do?"
```

You will find this usage of the PRINT command to be very helpful when programming in BASIC I on the M5.

4-1. What Is a Variable?

The concept of a variable will probably be unfamiliar to many users. First, let's calculate the following calculation before giving a detailed explanation.

For example, in the three lines below, the sum of 15 and 9 is printed. The variable A has been assigned the number 15, and likewise, B the number 9. In this way, A and B, used in place of numbers, are called "variables". But more precisely, A and B since they store numbers are appropriately called numeric variables.

For example

```
10. A = 15
20. B = 9
30. PRINT A + B
```

```
a=15
b=9
Print a+b
```

A variable is a symbolic representation (name) that will assume a value. A variable can also take on different values besides numbers; for example, it can take on characters and symbols. In this case, they are called character string variables. They will be described in more detail with a sample program later in this manual.

The value of a variable will also change for different executions of the program or at different stages within the program. Don't worry, if you are a little confused, this will be explained in detail later.

4-2. Letter Variables

In the example above, the variables A and B are each one character. However, you can use symbols and more characters as BASIC-I variables. A variable can be at most 32 characters. The first should be non-numeric so that BASIC I does not confuse it with a number.

For example

```
10. AAAA1000 = 15
20. BBB44dd2 = 9
30. PRINT AAAA1000 +
    BBB44dd2
```

```
aaaa1000=15
bbb44dd2=9
Print aaaa1000+bbb44dd2
```

IMPORTANT NOTES:

Do not name a variable that has a special meaning in BASIC I; for example, do not use a variable named `PRINT`.

Also, do not start a variable with a number, otherwise, you will get an error and the computer will misinterpret your variable.

Changing one character in a variable indicates a different variable. For example, `LEN` and `LENG` are different variables.

In conclusion, we hope you understand the meaning of variables. The concept is very important and basic in order to program in BASIC I. In the latter half of this manual, variables are explained in more detail with a sample program.

5-1. Steps in Making a Program

A program is made up of many instructions that are each separated by the input of the Return key.

Now that you've learned how to input characters from the keyboard to the screen, and you understand the concept of a variable, let's make a program.

5-2. Let's Make a Program

First, we have to clean up the computer as follows:

- **Clearing the Screen (CLS)**

It is a good idea to start typing in a program from the top of the screen. To erase everything on the screen, you need to use the CLS command. Type in C L S , then press the RETURN key.

- **Cleaning Up Internal Memory (NEW)**

It is also necessary to erase internal memory, that is, all programs and data stored in the computer before entering your program. Otherwise, you may encounter some unknown problems. To clean up internal memory, type in N E W, press the Return key. This does not clear the screen, but everything displayed on the screen is cleared from internal memory.

NOTE:

Clearing internal memory and the screen can also be done by powering off and on the M5.

Okay, now let's make a program. Key in the following program as described below.

Line Number	Command	Operand
1	0 SPACE P R I N T SPACE	2 + 3 RETURN
2	0 SPACE E N D	RETURN

First, let's explain the meaning of 10 and 20. They are line numbers required by BASIC I so it will know how to distinguish and reference each instruction. The line number or identifier as you will discover later is very important for programming in BASIC I.

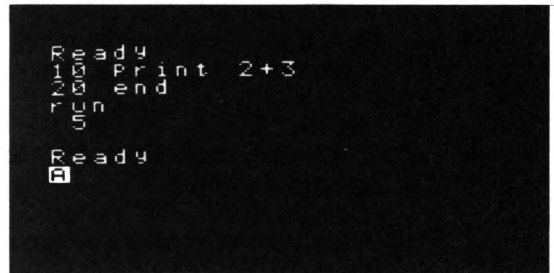
The next column is reserved for BASIC I commands. In this case, the PRINT command tells the M5 to print the sum of 2 plus 3. The END command just tells the M5 that this is the end of the program. This is always the last command or instruction of the program.

Next is the column for operands. In this example, only the PRINT command has an operand— $2 + 3$. Operands are the input for the command.

Well, you've entered the program but you want to know how to run it and see the answer.

•Executing a Program (RUN)

To execute a program, you need to use the RUN command. Key in RUN and press the Return key. You should see your answer now. The RUN command is always used to execute a program.



Executing a Program

5-3. Let's Look at the Program Again

•Listing the Program (LIST)

To list out your program at any time, use the LIST command. Try it! Enter LIST and then Return. Remember, commands may be typed in small or capital letters.

```
list
10 PRINT 2+3
20 END
```

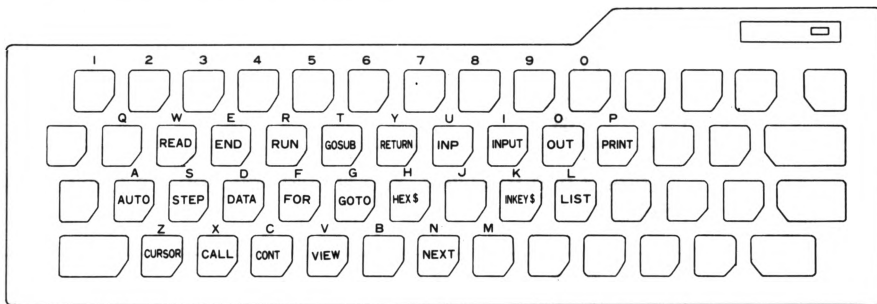
NOTE:

A space will automatically be inserted between the line number and the command.

•Convenient FUNC Key

You can display automatically many BASIC-I commands with the simple touch of the FUNC key and the appropriate key. These commands are written in small letters above the alphabet keys on the keyboard. Note that you can only display these keywords and not any other words.

For example, try displaying the PRINT command. While pressing the FUNC key with one hand, press the P key with the other hand. Wasn't that easy? Try some other commands.



BASIC I commands available with FUNC key

5-4. Let's Make It Look More Like a Program

•Entering Data (number or character string) from the Keyboard

Let's look at our previous sample program which added two numbers. That program only did one calculation but, now, we would like to do an unlimited number of calculations without having to enter the program over and over again. Let's enter the following example. First, clear your screen and internal memory with the CLS and NEW commands.

```

10 input a
20 input b
30 c=a+b
40 print c
50 end

```

Be sure there is a space here

Be sure there is a space here

Be sure there is a space here

Chapter 5 What's in a Program?

Now we're ready to execute the program. Enter RUN. You should get the following response on the keyboard.

```
run
? 
```

Notice that the blinking cursor is positioned after the '?'. This indicates that the program is expecting input. In line 10 of the program, you typed in 'input a'. Well, the INPUT command specifies that data is expected from you, the user. A '?' symbol is a prompt from the computer for you to input some data. The first example was to add 2 and 3. Enter 2 and Return. This value entered will now be assigned to the variable a. Now the screen will display another '?' to prompt you for the second value for the variable b (line 20—input b).

```
run
? 2
? 
```

Enter 3 and RETURN. You should now see the following display on your screen.

```
run
? 2
? 3
5 ← result printed

```

The answer 5 was printed. Now if you want to add two more numbers, just type in RUN and repeat the same procedure as described above. Try it.

- **Getting a Listing**

Let's get a listing of the above program.

```
list
10 INPUT A
20 INPUT B
30 LET C=A+B
40 PRINT C
50 END
```

Notice that it is slightly different from what you typed in. Specifically, look at line 30 and you see that the word LET has been inserted before $C = A + B$. This is automatically done because each instruction must have a BASIC I command. And since the LET command is the one used most, the M5 has made it easier for you to program so that you don't have to enter it all the time. All you had to do was to type the equation, in this case, $c = a + b$.

- **Errors**

If you were not able to get the answer, then you probably typed the program incorrectly. For example, your program may look like this.

```
list
10 LET INPUTA
20 LET INPUTB
30 LET C=A+B
40 PRINT C
50 END
```

The LET command was probably automatically inserted when no space was inserted between 'input' and 'a'. The computer misunderstood your command line as 'inputa' instead of 'input a', and thus, it inserted 'LET'.

```
10 inputa
      ↑
      Did not leave a space
```

Be sure that you leave a space between the command word and the operands. If you still get an error message after you run the program, refer to section 5-6 on Corrections.

5-5. Line Numbers

- **Importance of Entering Line Numbers**

If you don't enter line numbers, then the instructions will not be entered into the program. Instead the instruction will be executed when the RETURN key is pressed. Always enter line numbers when creating a program.

- **Why in Increments of 10?**

Well, you could number the lines as 1,2,3,... but what would happen if you decided that you wanted to add one or more lines between, say, 5 and 6. And your program was 50 lines long. You wouldn't want to renumber over 40 lines, would you? By numbering the lines in increments of 10, you can modify and update your programs easily and quickly.

- **Automatic Line Numbering (AUTO)**

BASIC I in the M5 provides a very useful command in which lines are automatically numbered when you enter a program. Everytime you enter an instruction, the line number would have already been printed on the screen. To invoke this function, use the AUTO command. This command requires two input values: the first specifies the line number to start at; and the second specifies the increment for subsequent line numbers.

This first example specifies automatic line numbering beginning with line number 100 in increments of 10.

```
AUTO 100,10
```

The second example starts automatic line numbering with line number 100 in increments of 50.

```
AUTO 100,50
```

Note that line numbering can only be within this range, 1-32767.

To cancel the Auto function, press the RETURN key only after the a line number is displayed.

5-6. Corrections

To correct the program below, compare it with the one in section 5-4.

```
10 LET INPUTA
20 LET INPUTB
30 LET C=A+B
40 LET PRINTC
50 END
A
```

- **Using the SPACE Key**

First, position the cursor on line 10 at the beginning of the word LET. Press the Space key three times to erase this word.

```
Ready
10 1st
20 LET INPUTA
30 LET INPUTB
40 LET C=A+B
50 LET PRINTC
50 END
```

Now position the cursor over to the letter A and press the SPACE key again.

```
Ready
10 1st
20 LET INPUTA
30 LET INPUTB
40 LET C=A+B
50 LET PRINTC
50 END
```

Type in A and press the Return key.

- **Using CTRL X and LIST for Editing**

You can list the program while you are correcting it without having to move the cursor to the end of the last line number.

For example, position the cursor at the beginning of line 20. Press the CTRL key and the X key together to erase this line. Then, type in LIST. What happened? The line number 20 reappeared again. The deletion using the CTRL and X keys only erased the instruction but not the line number.

- **Using the DEL Key**

You can use the DEL key to correct the errors. Verify your corrections with the LIST command.

- **Reentering One Line**

You can correct a line by simply reentering it without having to move the cursor to that line. For example to correct line 40, simply input the following on a new line.

```
40 PRINT C
```

Leave space

Now, enter the LIST command to verify the change.

```
10 INPUT A
20 INPUT B
30 LET C=A+B
40 PRINT C
50 END
```

- **Erasing One Line**

You can erase one line by simply typing in the line number immediately followed by pressing the Return key without having to move the cursor to that line. For example, to delete line 50, simply input the following.

```
50
```

RETURN Press RETURN key

No space and Return key pressed

- **Deleting Several Lines**

Of course, you can delete your program by using the NEW command but if you want only to delete certain lines without having to delete one line at a time, you can do the deletion easily and quickly with the DEL command. You also need to specify the first and last line numbers of the deletion. For example, lines 20—40 will be deleted as keyed below.

```
DEL 20,40 RETURN
```

Now, let's see what remains of your program. Type in LIST.

```
list
10 INPUT A
Ready
```

As you can see, lines 20—40 were deleted as well as line 50 from the previous method of deletion described before.

5-7. Using the INPUT Command in Another Way

The INPUT command can also be used in another way.

Enter the program on the right as shown below.

Previous program

```
10 INPUT A
20 INPUT B
30 LET C=A+B
40 PRINT C
50 END
```

new

```
10 input a,b
20 c=a+b
30 print c
40 end
```

Lines 10 and 20 of the program on the left have been merged into one line on the right. Essentially, the INPUT command is now just on one line instead of two lines. Let's execute the program.

```
run
?
```


Chapter 5 What's in a Program?

At this point, you can now enter both numbers separated by a comma on the same line. For example, enter 7 , 5 Return.

```
run
? 7,5
12 ← Comma
    ← Answer
```

The answer 12 is printed.

If you have many input values, you can also have your program accept more input on one line. Look at the example below.

```
list
10 input a,b,c
20 d=a+b+c
30 print d
40 end
```

Let's execute the program.

```
run
?
```

This time you will need to input three numbers separated by commas. For example, add 3, 4 and 5.

```
run
? 3,4,5
12 ← Answer
```

See how easy it is!

- **INPUT Command Error Messages**

For example, in the last program that required three inputs, if you had entered 'CD', this is what you'll see.

```
run
? cd
-??
```

The '-??' indicates that there is an error in your input. Specifically, letter characters have been input for a numeric variable. You must now enter numeric values. Let's enter 3 and 4, and see what happens.

```
run
? cd
-?? 3,4
??
```

The '??' indicates that there is still more input to be entered. Remember our program called for three inputs to be entered. Let's enter the last input 5.

```
run
? cd
-?? 3,4
?? 5
12
Ready
■
```

In summary, the error messages can be described as follows.

-??	Wrong input	Characters were entered when numbers were expected as input.
??	Insufficient input	Not enough input was entered. Enter only what is expected. The RETURN key was pressed without any input entered

5-8. Using Messages to Make Your Program More Understandable

As you noticed, how you know what to input or what the program does beforehand can be very confusing. Therefore, messages can be very helpful in executing a program.

- **How to Print Out a Message**

First of all, let's clear the screen with the CLS command. In the last program where three numbers are expected, line 10 can be changed as follows.

```
Ready
10 input "Addition of three numbe
r= a+b+c":a,b,c
```

Quotation Marks, Space

Enter the change and run the program.

```
Ready
run
Addition of three numbers a+b+c?
3 3
3 4
9
Ready
```

Notice that the INPUT statement printed the text within the quotation marks. Now, you can enter three numbers. Easy to understand, isn't it?

- **Better Messages**

You can even print out clearer messages. Look at the program below.

```
10 PRINT "Addition of three numbers a+b+c"
12 INPUT "a=";a
14 INPUT "b=";b
16 INPUT "c=";c
```

Be sure to leave
space here

Be sure to type
semicolon here

Enter this program and execute it.

```
run
Addition of three numbers a+b+c
a=?
```

Notice that the program is now asking for the input of a. You will now get prompts for the other two inputs.

```
run
Addition of three numbers a+b+c
a=? 3
b=? 4
c=? 5
12
```

Aren't these messages easier to understand? They're very helpful!

5-9. Other Usages of the LIST Command

Up to now, you've learned how to list your entire program with the LIST command. However, the LIST command is very versatile!

For example, if you had an error on line 30, this is how you can use this command.

```
run
Err 2 in 30
```

You only want to see line number 30 but not the whole program. Just enter the following.

```
list 30,30
      ↑
      comma
```

You want to see all lines up to line number 30.

```
list ,30
```

You want to see all lines from line number 30 onwards.

```
list 30
```

You only want to see lines 30 through 60.

```
list 30,60
```

The LIST command can be used in many different ways!

6-1. Saving Your Program

You're probably wondering, "If I typed in a really long program, how can I save it without having to retype it in each time?" As you know, each time you turn off the computer, all of internal memory is lost. Therefore, you'd like a way to save your program and use it again at any time. Well, that can be easily done in BASIC I.

First of all, you will need a cassette tape recorder and a blank cassette tape. Connect the tape recorder as described in the Hardware manual.

Let's save the program that has been used as the example so far.

Okay, let's learn how to save a program. Don't worry, it's easy!

```
10 PRINT "Additional of three numbers a+b+c"
12 INPUT "a=";A
14 INPUT "b=";B
16 INPUT "c=";C
20 LET D=A+B+C
30 PRINT D
40 END
```

• Attaching a Filename to Your Program

When you want to save a program, you need to assign a name to your program. This name or filename labels or identifies your program to the computer. If no filename is attached to your program, then it cannot be saved.

In the sample program above, let's attach the filename "add3". The SAVE command is used to save a program. Before you enter this command, you have to prepare your tape recorder for recording.

If you have hooked up your tape recorder for remote control, then make sure you have forwarded the tape past the leader portion before inserting it into the tape recorder. Press the appropriate buttons for recording. Notice that the tape does not move. That's because the computer is controlling it. Now let's enter the SAVE command.

```
save"add3"
```

Hit the Return key and now the tape should move. When the program has been saved, the ready message will be displayed and the tape recorder will automatically stop.

```
save"add3"
Ready
```

If your tape recorder is not hooked for remote control, then insert the tape and forward it past the leader portion. Now enter the command as shown above (without pressing Return). Next, press the appropriate buttons to begin recording and press the Return key to begin saving the program. The ready message will be displayed when the program has been saved (see the figure above). Notice that your tape recorder is still going. You have to manually stop it, otherwise, it'll keep on recording.

It wasn't that hard to record, was it?

- **More About Filenames**

Before we continue, let's explain a little more about filenames.

First, filenames are at most 9 characters long. If you try to use 10 or more characters, then your program will not be saved.

Second, try to use filenames that are meaningful and useful so that you will not forget them easily, especially if some of them will not be used often. For example, in the program above, if "nothing" was chosen, you'll probably be wondering what the program is all about. So, be sure to pick catchy, descriptive filenames.

Third, be sure you write down the filename somewhere—in a notebook or on the cassette label or box. Otherwise you'll forget.

Just pretend that you are recording beautiful programs that can be played over and over again to give you lasting pleasure.

6-2. Verifying Your Program

You saved your program, but aren't you a little worried that it might not have been saved correctly? Relax, the M5 provides a way for you to verify that.

- **Using the VERIFY Command**

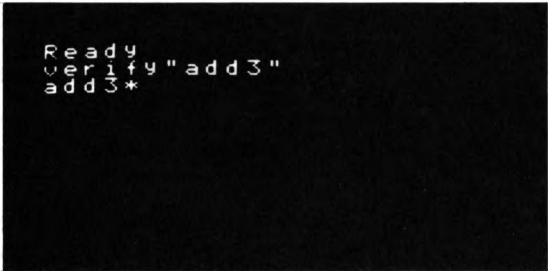
The VERIFY command will check to make sure that your program has been saved properly.

Before you enter the command, you need to rewind the tape (if your tape recorder is not remote-controlled).

Enter the command (rewind the tape now to the beginning of the tape if your tape recorder is connected for remote control) and press the playback button and the Return key.

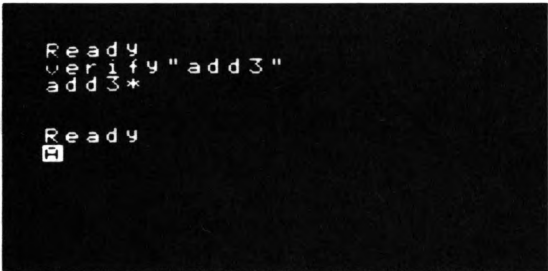
```
verify"add3"
```

The tape should now move. Pretty soon your filename followed by an asterisk will be displayed on the screen. This indicates that your program is being verified.



```
Ready  
Verify "add3"  
add3*
```

When your program has been verified, a ready message will be displayed. This means that everything went okay.



```
Ready  
Verify "add3"  
add3*  
  
Ready  
[ ]
```

At this point, the tape recorder stops if it is under remote control, otherwise, you'll have to press the stop button. There, wasn't that easy?

NOTE:

If the tape recorder does not work, make sure you have connected it properly. Are all connections secure? Do you need fresh batteries?

If your tape recorder is working okay, but you still were not able to verify your program even though you supposedly saved it, rewind the tape and try saving the program again.

- **Another Way to Use the VERIFY Command**

You can also use this command to find out all the filenames on a tape. If you forgot or are not exactly sure of the name, then this command will come in especially handy. Use a filename that you are pretty sure that is not on tape if you want to list all the filenames, otherwise, the command will stop when the filename is found.

For example enter the command as shown below.

```
verify "dummy" ← "The filename should be uncommon"
```

After you enter this command, the filenames on the tape will be displayed on the screen. If no match was found, then the tape will go to the end. However, the verify command is still in effect. Therefore, if you want to cancel it, then you need to press simultaneously the SHIFT and RESET keys.

```
verify "dummy"  
add3
```

6-3. Loading Your Program

Now that you saved and verified your program, you want to do one final check. You clear memory with the NEW command and now you want to load your program into the computer. The other times you will be loading in your program will be when you first start, when you want to load in a different program, and when you accidentally erased a program by mistake.

Let's see how easy it is to load your program!

- **Using the OLD Command**

The OLD command loads a program into the M5.

First of all, rewind the tape to the appropriate place.

Again, let's use the "add3" program. Clear memory with the NEW command. Enter the OLD command as shown below.

```
old "add3"
```

The tape plays and after a while, your filename followed by an asterisk will be displayed on the screen. This means that your program is being loaded.

```
Ready  
old"add3"  
add3*
```

If there are other programs located on the tape before "add3" and you had rewound the tape to the beginning, then their filenames will be displayed to the screen; however, they are not loaded. Only "add3" will be loaded.

```
Ready  
old"add3"  
amnor  
checkbook  
startrek  
add3*
```

- **Using OLD Without a Filename**

If you do not enter a filename with the OLD command, then the first program that is read will be loaded. In the example above, "checkbook" will be loaded. The program loaded also depends on where the tape is positioned prior to executing this command.

```
Ready  
old  
checkbook*
```

7-1. Various Commands

So far, you've learned some very simple commands. However, if you want to write more difficult programs, then you need to know some more powerful commands. In this chapter, we will introduce these commands. Let's go!

7-2. Endless Loop

Again, let's use the sample program from before to explain the meaning of an endless loop.

[illegible]

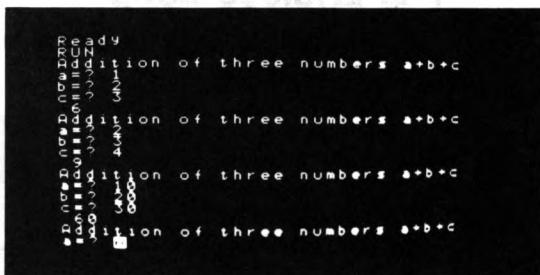
First of all, let's add the CLS command in line 5. This will clear the screen before the program is executed. This is a nice way to indicate the start of a program.

Next, let's add this command as shown below.

[illegible]

Enter the three numbers as before. The answer will be printed.

But notice what happens next! You get the prompt to input a again. Then for b and c. Another answer is printed and the program starts all over from the beginning again. This type of continuous looping is called an endless loop.



After the answer is printed (line 30), the next command which is on line 35 is executed. In this case, the GOTO command is executed. This command instructs the computer to go to line 10 in which the heading is printed. You could've easily specified line 5 in which case, the screen will be cleared each time before you enter the numbers, or specified line 12 in which case, the heading would be printed only once.

Imagine what you can do with the GOTO command. You can tell your program to go anywhere. Just be sure that you specify a line number that is in your program, otherwise, you'll get an error message.

I bet you're itching to learn some more commands? Okay, let's continue!

7-3. Looping Three Times

Suppose you only wanted to loop three times. What do you do? Well, you could've easily pressed together the SHIFT and RESET keys after the third time to stop the program but you must be thinking, "There's got to be another way!" Right you are, there is!

Let's look at the program below which has been modified to do three repetitions.

```

2 REM CALCULATION
5 CLS
7 FOR Z=1 TO 3 STEP 1
10 PRINT "Addition of three numbers a+b+c"
12 INPUT "a=";A
14 INPUT "b=";B
16 INPUT "c=";C
20 LET D=A+B+C
30 PRINT D
35 NEXT Z
40 END
  
```

Notice that lines 7 and 35 are different

```

      └─ a space is needed here
7 FOR Z=1 TO 3 STEP 1
35 NEXT Z
      └─ a space is needed here
  
```

Why don't you execute the program and see what happens.

```

Addition of three numbers a+b+c
a=1
b=2
c=3
6
Addition of three numbers a+b+c
a=1
b=2
c=3
6
Addition of three numbers a+b+c
a=1
b=2
c=3
6
Ready
  
```

Your screen probably looked a little like the above.

Well, this new command (look at line 7) told the computer to execute the next set of commands three times. The set of commands is bounded by line 35.

The FOR—TO—NEXT command is very powerful in that it lets you specify exactly how many times you want to do a certain set of commands.

Let's analyze the command in more detail. The FOR Z = 1 says to start the loop counter Z at 1. The TO 3 says to stop after Z > 3 and the STEP 1 says to add 1 to Z each time the NEXT Z command is executed.

So in this example, Z = 1 when the program starts execution. After the answer is printed, Z = 2. The program starts and the answer is printed again after you input the three numbers. Now Z = 3 and the program repeats itself. Then Z = 4, at this point, the computer knows to stop looping because Z is greater than 3 and this is what happens.

<pre>7 FOR Z=1 TO 3 STEP 1</pre> <div><div><u>Z = 1,</u></div><div><u>Until Z > 3,</u></div><div><u>Add 1 to Z</u></div></div>

In conclusion, this command allows you to execute a certain set of instructions a specified number of times.

This command is very powerful so several more usages will be explained in the next section.

Are you ready? If not, why don't you take a break. It's good to let new and exciting ideas sink in a little. If you go too fast, you'll become confused and discouraged. And we don't want that! Go back and review a little. And when you're ready, move on!

8-1. Game Program

Up to now, we've been familiarizing you with how to use BASIC I. From here on in, that won't be the case anymore. We will now describe the commands involved with games.

Our first game, the "dice game" has already been programmed. This is a very simple game program which will provide you with many basic and necessary commands and concepts for programming more difficult programs.

You will be introduced to many new words and complicated explanations.

Don't worry, if you don't understand exactly, just input the program and play and learn along.

Let's go!

8-2. RND Function

For the dice game, it's difficult to display the die on the screen right now, so let's just print the number of the die on the screen.

What's the RND function?

This function outputs random numbers. So, for example, when you throw a die, you can get a number between 1 and 6. In this case, the function simulates the throwing of a die. This function can generate any number and is appropriately called a random number generator.

In BASIC I, there are 23 other functions which are described on pages 89 to 92.

Now let's use the RND function to determine the pips on a die. Enter the following program. Be sure to clear memory first.

```
10 CLS
20 LET DIE=RND(5)+1
30 PRINT DIE
40 GOTO 20
```

Run the program.



You'll notice that the numbers on the screen are all between 1 and 6. Press both the SHIFT and RESET keys to stop the program.

Let's explain line 20. The RND function with the number 5 specified within parentheses will generate a number between 0 and 5. But the number on a die is between 1 and 6; therefore, 1 is added to the number generated by RND to get a die number.

Consequently, you can generate other numbers in larger ranges. For example, if you input 100 or 1000 within the parentheses, then you will get numbers between 1 and 101 or 1 and 1001.

This RND function will be especially useful later on.

8-3. "Computer" and Dice Game

Well, now you know how to generate the number on a roll of the die.

Let's play the dice game with the computer. The winner will be whoever throws the higher die.

Input the program as follows.

```
10 CLS
100 REM Computer rolls your die
110 RANDOMIZE
120 MYDIE=RND(5)+1
130 PRINT "Your die is a ";MYDIE
200 REM Computer rolls its die
210 RANDOMIZE
220 CDIE=RND(5)+1
230 PRINT "Computer rolls a ";CDIE
```

Run the program. You'll see what your die and the computer's die are.

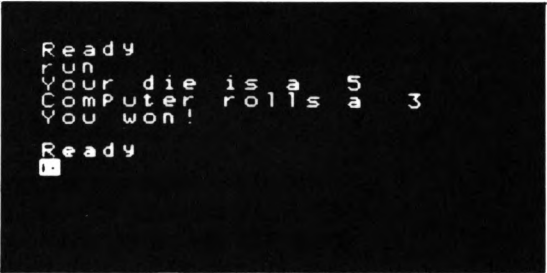
You know who won but let's have the computer tell you. A decision is needed so let's enter the following statements.

```
300 REM Who won?
310 LET STR$="The computer won"
320 IF MYDIE=CDIE THEN LET STR$="Draw"
330 IF MYDIE>CDIE THEN LET STR$="You won!"
340 PRINT STR$
```

Notice that the judgment was based upon the IF-THEN command which you saw before. Let's look at lines 310 to 330.

- Line 310 Store the character string " THE COMPUTER WON."
 into the string variable STR\$.
- Line 320 Store "DRAW" if MYDIE and CDIE are the same.
 Here, the string variable STR\$ changed from
 "THE COMPUTER WON." TO "DRAW".
 If the numbers are the same, then STR\$ remains
 at " THE COMPUTER WON.".
- Line 330 Store "YOU WON!" if MYDIE is greater than CDIE.

Hopefully, you understood what has been explained so far. If you are not completely sure, just run the program and let the computer display the results and winner.



```
Ready
Run
Your die is a 5
Computer rolls a 3
You won!
Ready
```

• Character String Variables

Characters can also be stored in variables.

In the above program at line 310, the string variable STR\$ contains " THE COMPUTER WON."

The variable name must end with a \$ sign and can be at most 16 characters.

The character string stored (at most 18 characters) must be enclosed within double quotation marks.

Character string variables behave just like numeric variables, in that their values can be changed at any time. Notice what STR\$ can be in the above program.

8-4. Saving Frequently Used Code as Subroutines

The program above used two dice, one for you and one for the computer. This time, let's try to use only one die between you and the computer because there is no need for two dice.

- **Subroutines**

A subroutine is essentially a group of common instructions used repeatedly in a program. For the dice game, let's make a subroutine out of the instructions used to generate the throw of a die. Change your program as follows.

```
110 GOSUB 400
120 LET MYDIE=DIE
...
210 GOSUB 400
220 LET CDIE=DIE
...
...
350 END
400 REM Randomly generated dice subroutine
410 RANDOMIZE
420 LET .DIE=RND(5)+1
430 RETURN
```

Notice that there are two new commands, the GOSUB command and the RETURN command. They are always used together. The GOSUB command above tells the computer to go to the subroutine, in this case, located on line 400. The RETURN command tells the computer to return to the next line after the GOSUB command line.

Let's list the program

```
10 CLS
100 REM computer rolls your die
110 GOSUB 400
120 PRINT "Your die is a ";DIE
130 LET MYDIE=DIE
200 REM computer rolls its die
210 GOSUB 400
220 PRINT "Computer rolls a ";DIE
230 LET CDIE=DIE
300 REM who won?
310 LET STR$="Computer won."
320 IF MYDIE=CDIE THEN LET STR$="Draw."
330 IF MYDIE>CDIE THEN LET STR$="You won!"
340 PRINT STR$
350 END
400 REM randomly generated die subroutine
410 RANDOMIZE
420 LET DIE=RND(5)+1
430 RETURN
```

Let's explain the program in detail

1. First, the screen is cleared (line 10).
2. Line 110 is executed in which control of the program jumps to line 400.
3. The die is thrown once in this subroutine and the value is stored in the variable DIE. The RETURN command passes control back to line 120.
4. Your die is printed and the value is stored in MYDIE.
5. Next, the computer's die is determined in the subroutine, printed and then saved in CDIE.
6. From here on in, we continue as described before; the winner is determined and printed.

The values of the dice are saved in CDIE and MYDIE after each subroutine call because they are needed later to determine the winner. If CDIE and MYDIE are not used, then the computer has no way of remembering what you rolled because on the second subroutine call, the old value of DIE (i.e. your roll) will have been updated by the computer's value. Consequently, no winner can be determined.

The subroutine has many usages.

You can have several subroutines in one program. You can even call another subroutine within one subroutine.

For the present, the dice program is completed, so let's save it on a cassette tape with the SAVE command.

8-5. How Many Times Did Each Number Come Up?

Let's determine how many times each number on a die comes up after a certain number of rolls. First, clear memory (NEW command) and enter the program below.

```

5 REM Dice Statistics
10 CLS
20 FOR I=1 TO 50
30 LET DIE=RND(5)+1
40 IF DIE=1 THEN LET D1=D1+1
50 IF DIE=2 THEN LET D2=D2+1
60 IF DIE=3 THEN LET D3=D3+1
70 IF DIE=4 THEN LET D4=D4+1
80 IF DIE=5 THEN LET D5=D5+1
90 IF DIE=6 THEN LET D6=D6+1
100 NEXT I
110 PRINT "ONE   -";D1
120 PRINT "TWO   -";D2
130 PRINT "THREE -";D3
140 PRINT "FOUR  -";D4
150 PRINT "FIVE  -";D5
160 PRINT "SIX   -";D6
170 END

```

Run the program.

```

ONE   - 8
TWO   - 10
THREE - 10
FOUR  - 10
SIX   - 10
Ready

```

Notice lines 40 through 90. Don't they look similar? You are probably wondering if there might be a simpler way. Right, you are. In the next section, we'll introduce the concept of arrays in which you can make those statements into one statement.

8-6. Array Variables

Let's simplify the dice statistics program of the previous section. Enter the following program into the computer. Clear memory first.

```

2 REM Dice Statistics Program
5 CLS
10 DIM HEADER$(6),TIMES(6)
12 REM header array has headers
14 REM times array has times rolled
20 FOR I=1 TO 50
30 LET DIE=RND(5)+1
40 LET TIMES(DIE)=TIMES(DIE)+1
50 NEXT I
60 FOR I=1 TO 6
70 READ HEADER$(I)
80 PRINT HEADER$(I); "-";TIMES(I)
90 NEXT I
100 DATA "ONE ", "TWO ", "THREE ",
        "FOUR ", "FIVE ", "SIX "
110 END

```

Run the program and see what happens. Similar output to the above, isn't it?

```

ONE      - 9
TWO      - 10
THREE    - 9
FOUR     - 8
FIVE     - 6
SIX      - 6
Ready

```

• Handling Lots of Data with Arrays

Arrays, as you may have noticed, handle lots of data. Imagine your report card grades as stored in an array. For example, you might have algebra, reading, writing, and spelling as your subjects. Then you could record your grades for each term (pretend, you receive grades three times a year). Well, you can have one array for each of your subjects. And since you know you will receive three grades, you can specify the size of the array to be three.

For example, your report card may look like this.

	1st term	2nd term	3rd term
Algebra	80	85	90
Reading	75	80	80
Writing	85	80	75
Spelling	90	95	95

Well, that can be stored in the computer as follows.

	1	2	3
Algebra	80	85	90
Reading	75	80	80
Writing	85	80	75
Spelling	90	95	95

Pretty easy to understand, isn't it?

To declare arrays, you need to use the DIM (dimension) command. You can use character or numeric arrays. Let's look at line 10.

```
10 DIM HEADER$(6), TIMES(6)
```

A character array has been declared of size 7 (yes, 7 because indices start from 0. So, 0 to 6 yields 7 elements). A little confusing, don't worry. You'll understand it soon enough. The character array is named HEADER\$. This is a character array because of the \$ sign at the end of the variable name. The TIMES array can also contain seven values, numbers, in this case.

• **How to Use Array Variables**

Let's look at line 40.

```
40 LET TIMES(DIE) = TIMES(DIE) + 1
```

Each time DIE is calculated from the RND function, DIE is used as an index into the TIMES array. The TIMES array basically stores the number of times a die is rolled. TIMES(1) contains how many times 1 was rolled, etc. As we pass through the FOR-TO-NEXT loop each time, the appropriate variable TIMES(DIE) is incremented. This variable means the same as D1,D2,...D6 of the program in section 8-5.

Remember it was mentioned earlier that each array contains 7 values. Well, whenever you declare the size of an array, the number of variables is always one more than the size. In this case, the array variables HEADER\$(0) and TIMES(0) are not used because a die does not have the number 0 on it. Unless, you're playing with some strange dice?

Logically, array variables let you group data together according to some common factor. For example, you can have arrays that store names, ages, addresses, salary, and in the instance of the die game, you can keep track of how many times a certain die face came up compared with the other values.

8-7. Reading Data in Your Program

Sometimes, you may want to read data in automatically without having to input it in manually. The READ command lets you perform that function along with the DATA command. Like the GOSUB and RETURN commands, these two must come together.

Your data may be character strings or numbers. By using the READ and DATA commands, you can conveniently input data. Also, you can easily change your data.

The READ command will read values from the DATA command line into the variables listed on the READ command line. The variables and values must correspond and be positioned correctly, otherwise, you will get an error.

You can input several different types of variables on the same command line. For example,

```

10 READ A,B,C$,D,E$
20 DATA 55,32,"ABC",63,"CDS"

```

Character string values must be enclosed within double quotation marks ("), and values must be separated by commas.

Let's look at lines 60 to 100 from the program earlier in this section.

```

60 FOR I=1 TO 6
70 READ HEADER$(I)
80 PRINT HEADER$(I); "-"; TIMES(I)
90 NEXT I
100 DATA "ONE ", "TWO ", "THREE", "FOUR ",
        "FIVE ", "SIX "

```

Each time the READ command is executed, the computer stores a value into the HEADER\$ array. For example, after reading is completed, the array will look like this.

HEADER\$(1)	HEADER\$(2)	HEADER\$(3)	HEADER\$(4)	
HEADER\$(5)	HEADER\$(6)	"ONE "		
"TWO "	"THREE "	"FOUR "	"FIVE "	"SIX "

9-1. What's a Character String?

All right. We're going to get down to brass tacks and find out how to use character strings. But first, get a glass of milk, or some coffee, and put yourself in a good state of mind. This chapter may take some time to get through. Be patient. There's some good information here. And it's not very difficult.

Put simply, a character string is a combination of characters or numbers. In this light, it is a very simple idea. But if you take this a little further and try to imagine applications for using character strings, you can see this is a very powerful tool. We'll go through some applications and examples.

9-2. Playing with Character Strings

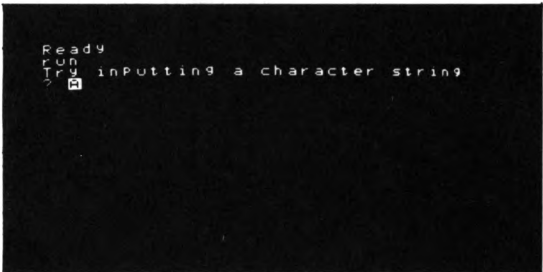
We're going to explore the usage of character strings with an example program. First input the program listed below. Don't worry about what the different statements do; we'll later go through and describe statements you haven't learned yet.

```
10 REM "LEARNING CHARACTER STRINGS"
20 CLS
30 INPUT "Input a string ";STR$
40 LET LNG=LEN(STR$)
50 FOR H=1 TO LNG
60 LET TEM$=LEFT$(STR$,H)
70 PRINT CURSOR(10,10);TEM$
80 FOR C=1 TO 1000:NEXT C
90 NEXT H
100 GOTO 20
```

Before describing what the various statements do, the following lists the variables used and what they mean. Notice the variable names describe their functions with an abbreviation; this makes programs much easier to understand and correct later.

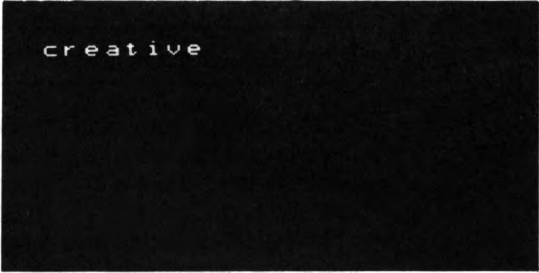
- STR\$A variable name followed by a '\$' indicates a variable that stores character strings. STR, in this case, is an abbreviation for STRing. It will contain the character string that you input to the program.
- TEM\$Like STR\$, TEM\$ is also a character string variable, but with a different function. What we're going to do later is play with the character string in STR\$ and store TEMPorary versions of it in TEM\$.
- LNGLNG is a variable containing the LeNGth of the character string that you input.
- H.....H is a variable used as a counter. Besides being a counter, it is also used to specify the length of a part of character string STR\$.
- C.....C is a variable that is also used as a counter. But its usage is different. We'll talk more about it later.

Run the program. Do you see the following information displayed on your screen? If not, use the LIST command to look over your program. Be sure everything is the same. The M5 computer can only do what you tell it to do; so you need to tell it exactly what to do.



```
Ready
Run
? inputting a character string
?
```

Now input a character string. Try inputting the character string 'CREATIVE'. Do you see the following information on your screen?



```
creative
```

Now input your name. Pretty easy, right?

9-3. Counting Characters

Look at line 40.

```
40 LET LNG=LEN(STR$)
```

You're familiar with assignment statements such as `A = 15`; line 40 is essentially the same. It's slightly different in that it uses a function to assign a value to the variable on the left of the equal sign. In other words, instead of using the value 15 (as in `A = 15`), line 40 uses a function built into BASIC I called `LEN` to assign a value to `LNG`. When `LEN` is specified as in line 40, BASIC I automatically takes the character string indicated by the variable inside the parentheses and counts the `LEN`gth of the character string.

In our sample program, line 30 asks you to input a character string which is then stored in `STR$`. Line 40 takes the character string stored in `STR$`, determines its length in characters and assigns this value to `LNG`. Let's summarize. At this point, we know that the character string that you input is stored in `STR$` and its length is stored in `LNG`.

9-4. Looking At Only a Portion of a Character String

Look at line 60.

```
60 LET TEM$=LEFT$(STR$,H)
```

So now let's see some of what we can do directly with character strings. It may not seem like much to begin with, but let's take a couple of characters from the left side of a character string, and leave the rest. LEFT\$, another function provided by BASIC I, does this easily and automatically.

In line 60 of our sample program, notice LEFT\$ has two variables listed inside the parentheses. These are called arguments. The first argument, named STR\$, is the character string from which we will grab some characters. H, the second argument, tells the LEFT\$ function how many characters to grab off the left side of character string STR\$.

For example, if STR\$ contains the character string 'ABCD' and H is assigned the value 3, the result of LEFT\$(STR\$,H) would be character string 'ABC', or the leftmost three characters of STR\$.

But, in our sample program, where do we assign H? Look in line 50.

```
50 FOR H=1 TO LNG
```

You've already been introduced to the FOR-TO-NEXT statement. Each time line 50 is executed, H is increased by 1. So when line 60 is first executed, H equals 1. This means we'll take the leftmost character from STR\$ using the LEFT\$ function in line 60. The next time line 60 is executed, H equals 2. So we'll take the two leftmost characters from STR\$. The next time, H equals 3, so the three leftmost characters of STR\$ are grabbed and stored in TEM\$... This goes on until every character in STR\$ has been examined.

You input the character string 'CREATIVE' earlier, a character string of eight characters. Let's see what actually happens to 'CREATIVE' in the software. In this case, H in line 50 varies from 1 to 8 since there are eight characters in 'CREATIVE'. Line 60 would then be executed 8 times with TEM\$ taking on a different value each time. Let's look at what TEM\$ will look like each time.

```
1st time ..... TEM$ = C
2nd time ..... TEM$ = CR
3rd time ..... TEM$ = CRE
4th time ..... TEM$ = CREA
5th time ..... TEM$ = CREAT
6th time ..... TEM$ = CREATI
7th time ..... TEM$ = CREATIV
8th time ..... TEM$ = CREATIVE
```

That's right. Every time H increases by one, the LEFT\$ function grabs one more character from STR\$. TEM\$ gets longer until it's grabbed the entire string stored in STR\$.

It's a little bit hard to tell by looking at the screen that TEM\$ changes each time line 60 is executed. What you'll see on the screen are all the characters you input displayed one by one from the left. It'll seem like the earlier characters have not changed at all. But we know better.

9-5. Jumping Around in Our Character String Program

By now it should be clear line 90 is the bottom of the loop started by line 50. In other words, when line 90 is executed, the M5 computer goes back to line 50 and checks whether H has exceeded LNG. When H becomes larger than LNG, the next statement executed is line 100.

When you were executing the program, the screen blanked out after displaying the last character of your input line. Look at line 100. It's executed after your character string was displayed character by character. In other words, it's executed after its job was done.

```
100 GOTO 20
```

Obviously, line 100 will cause the M5 computer to go to line 20. This is actually telling the M5 to start the job over again. Look at line 20.

```
20 CLS
```

Recognize it? That's right. We've executed this statement before. The CLS statement in line 20 will clear the screen. This prepares the screen for your next character string.

We've now come around full circle. You've also seen line 30 before. It'll ask you to enter another character string and display it for you one character at a time and then blank out the screen. You can keep doing this literally forever, or until you get bored of it.

9-6. Hints to Understand Your Program Better

So, you now know your practice software program goes to one line, then to another, and then yet to another that may not even be sequential. For larger programs, this can become a bit frustrating to understand unless you understand the flow of your program. Even then, since the screen only shows 24 lines, it's very difficult to understand how large programs jump from one place to another.

Before you even sit down at your computer, think about what you want your computer program to do. When this is clear in your mind and written down, compose your program on paper.

Let's think about our practice program again. Look at the lines to the left of the BASIC I statements below. They connect statements that are related to each other. Line 20 is connected to line 100 since line 100 will jump to line 20. Line 50 is related to line 90 since line 90 is the bottom of the FOR-TO-NEXT statement that was started in line 50. This will help you to see the flow of your programs.

```
10 REM "LEARNING CHARACTER STRINGS"  
20 CLS  
30 INPUT "Try inputting a character  
   string ";STR$  
40 LET LNG=LEN(STR$)  
50 FOR H=1 TO LNG  
60 LET TEM$=LEFT$(STR$,H)  
70 PRINT CURSOR(10,10);TEM$  
80 FOR C=1 TO 1000:NEXT C  
90 NEXT H  
100 GOTO 20
```

After you've typed your program in, the same technique can be used to help you correct your programs, especially if you have a printer.

You're not alone when you use this method; professional computer scientists also do it this way.

9-7. An Easy Software Timer

Let's skip line 70 for now and look at line 80.

```
80 FOR C=1 TO 1000:NEXT C
```

When you ran the program, did you notice a pause each time a new character was displayed? The pause was because of line 80. You know how a FOR-TO-NEXT statement works. Look at line 80. Confused? That's right. It seems line 80 does no work at all except loop around for 1000 times. You guessed it. It's stalling. It twiddles its thumb 1000 times and then goes on to line 90. If you want the M5 to stall a bit longer, increase 1000 to 3000, or more; or decrease 1000 to 500 so it'll stall less.

If you want to get into computer talk, call it a software timer. Try changing the timer value from 1000 and run the program again.

Notice there's a colon between the 1000 and the word NEXT in line 80. This is how you can put more than one statement on a line. Just insert a colon between the statements.

9-8. Printing Characters on the Screen

Look at line 70.

```
70 PRINT CURSOR(10,10);TEM$
```

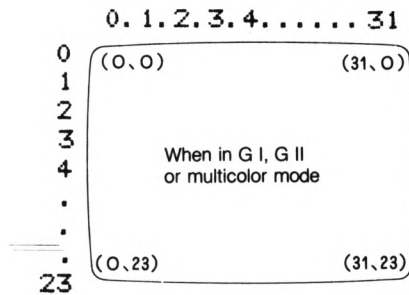
This PRINT statement is not like the others you've used. Remember line 60? It uses the LEFT\$ function built into BASIC I. The PRINT statement in line 70 is similar. Instead of simply displaying a variable, it uses a BASIC I function called CURSOR. Not surprisingly, this function, when used together with the PRINT statement, takes the cursor and puts it at a location known as (10,10). This is why your input line wasn't displayed starting from the leftmost column of your screen.

For example, the next PRINT statement will print the value stored in variable "A" at the center of the screen.

```
PRINT CURSOR(16,12);A
```

Chapter 9 What's a Character String?

As you can see, the CURSOR function has two arguments. These two arguments are the X (horizontal) and Y (vertical) coordinates of the screen. Look at the figures below.



Notice the numbers 0 through 31 listed above the screen. This means the screen has been divided into 32 vertical positions (0 is also used), otherwise called the X coordinate. Now look to the left of the screen. The numbers 0 through 23 show the screen is divided into 24 horizontal positions. If you combine both an X coordinate and a Y coordinate, you can display a character anywhere on the screen. Take an X coordinate and space over until you reach the vertical position of the X coordinate. Then go down the imaginary line of the X coordinate until you reach the horizontal position of a Y coordinate. There you are, at the point described by (X,Y). On the M5 computer, this is known as the G I mode. Now you see why the CURSOR function takes two arguments. The first argument is the X coordinate; the second is the Y coordinate. The following statement is the general case.

```
PRINT CURSOR(X,Y)
```

So why is (16,12) the center of the screen? Space over 16 positions across the top of the screen. It's the vertical center of the screen. Now space 12 positions down the screen. This is the horizontal center of the screen. Taken together, this is the center of the screen.

For instance, if you want to display a character at the top right of the screen, use the following statement. Do you see how it works?

```
PRINT CURSOR(31,0): "A"
```

Be careful when using the CURSOR function. If you point the cursor past 31 for the X coordinate or past 23 for the Y coordinate, the cursor will be displayed off the screen. That's right. You won't be able to see the cursor.

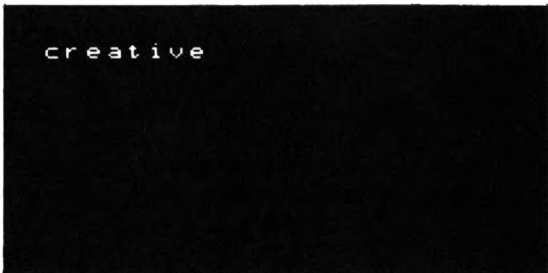
9-9. Using the TAB Function

Now you know how to display a character anywhere on the screen using X and Y coordinates. Another cursor function, called TAB, is also very handy to use. Unlike the CURSOR function, the TAB function can only space over horizontally. It only needs one argument.

To try the TAB function, let's replace line 70 of the character string program we were playing with earlier in this chapter with:

```
70 PRINT TAB(10);TEM$
```

Now run the program again. Let's use the character string 'CREATIVE' again. The result displayed on the screen is shown below.



creative

You cannot use the CURSOR function to force the screen to scroll. On the other hand, the TAB function can make the screen scroll. Try changing the value 10 in line 70 and run the program again. See how the cursor spaces over to the position indicated by the argument? Each time the end of a line is reached, the cursor moves to the first position of the next line.

9-10. Various Character String Functions

We're already gone over two character string functions. Namely, LEN and LEFT\$. Lo and behold, there are more. Among these are RIGHT\$ and MID\$. It's pretty obvious what these functions do. But let's try them and see. Go back to the program we were working on earlier in this chapter and replace line 60 with:

```
60 LET TEM$=RIGHT$(STR$,H)
```

Was the result what you expected?

Now let's try the MID\$ function. This time, replace line 60 in our practice program with:

```
60 LET TEM$=MID$(STR$,1,3)
```

Unlike the LEFT\$ and RIGHT\$ functions, MID\$ does not take characters off either end of a character string. Rather surprisingly, it grabs characters from the MIDDLE of a character string. What you need to do is tell it where to start taking characters and how many to take.

The first argument, STR\$, is the character string to look at.

The second argument, the number 1, tells the M5 to start from the first character of STR\$ (starting from the leftmost character). Yes, it designates the leftmost character of STR\$. If you change the second argument to 2, it'll start taking characters from the second character.

The third argument tells the M5 how many characters to take. In our example, line 60 will take three characters from STR\$ starting from the leftmost character.

Say we've stored the character string 'CREATIVE' in STR\$.

Character string	C	R	E	A	T	I	V	E
Character number	1	2	3	4	5	6	7	8

If we execute line 60, characters numbered 1 through 3 will be assigned to TEM\$. TEM\$ will contain the character string 'CRE'. But if we change the second argument from 1 to 3, what do you think will happen?

```
60 LET TEM$=MID$(STR$,3,3)
```

Yes. We'll take three characters starting from character number three and put them into TEM\$. TEM\$ will then contain 'EAT'.

Got the hang of it?

10-1. Dice Graphics

Remember the "dice game" in chapter 8? Let's make it more interesting and learn something at the same time. Sound good?

In this chapter, we're going to get into some graphics. But what are graphics? Put simply, they're pictures. And the way we're going to use them is just like the alphabetic and numeric characters we've used before. Sound easy? It is.

•Drawing Dice

Review the dice program you played with in chapter 8. If you didn't save it on tape yet, type it in again and add the part enclosed in in the following program list. One pointer; when you're typing in lines 1000 to 1540, use the graphics mode on your console to type in the graphics characters that are between the double quotes. Remember, you don't have to use the SHIFT or CTRL keys when using the graphics mode. Just press the key that you want displayed. After that, go back to the alphabetic mode to type the alphabetic characters, then back to the graphics mode again to type the graphics characters.

Hang in there. It takes a little practice and a little patience.

```

5 REM Dice Graphics Program
10 CLS
100 REM Computer rolls your die
110 GOSUB 400
120 PRINT "Your die is a ";DIE
130 LET MYDIE=DIE
140 GOSUB 500
200 REM Computer rolls its die
210 GOSUB 400
220 PRINT "Computer rolls a ";DIE
230 LET CDIE=DIE
240 GOSUB 500
300 REM Who won?
310 LET STR$="The computer won"
320 IF MYDIE=CDIE THEN LET STR$="Draw"
330 IF MYDIE>CDIE THEN LET STR$="You won!"
340 PRINT STR$
350 IF INKEY$=" " THEN GOTO 10
360 GOTO 350
370 END
400 REM Randomly generated dice subroutine
410 RANDOMIZE
420 LET DIE=RND(5)+1
430 RETURN

```

Wait for key
to be pressed

Roll die

```
500 REM Test dice/draw dice subroutines
510 IF DIE=1 THEN GOSUB 1000
520 IF DIE=2 THEN GOSUB 1100
530 IF DIE=3 THEN GOSUB 1200
540 IF DIE=4 THEN GOSUB 1300
550 IF DIE=5 THEN GOSUB 1400
560 IF DIE=6 THEN GOSUB 1500
580 RETURN
```

```
1000 PRINT " [---] "
1010 PRINT " [---] "
1020 PRINT " [---] "
1030 PRINT " [---] "
1040 PRINT " [---] "
1050 RETURN
1100 PRINT " [---] "
1110 PRINT " [---] "
1120 PRINT " [---] "
1130 PRINT " [---] "
1140 PRINT " [---] "
1150 RETURN
1200 PRINT " [---] "
1210 PRINT " [---] "
1220 PRINT " [---] "
1230 PRINT " [---] "
1240 PRINT " [---] "
1250 RETURN
1300 PRINT " [---] "
1310 PRINT " [---] "
1320 PRINT " [---] "
1330 PRINT " [---] "
1340 PRINT " [---] "
1350 RETURN
1400 PRINT " [---] "
1410 PRINT " [---] "
1420 PRINT " [---] "
1430 PRINT " [---] "
1440 PRINT " [---] "
1450 RETURN
1500 PRINT " [---] "
1510 PRINT " [---] "
1520 PRINT " [---] "
1530 PRINT " [---] "
1540 PRINT " [---] "
1550 RETURN
```

If you've followed the flow of the program and read the REMark statements, you'll know what this program is doing. For those of you who are not so sure, let's go over some important points.

Look at line 350.

```
350 IF INKEY$=" " THEN GOTO 10
```

indicates a SPACE key

We haven't gone over the INKEY\$ function yet. But like LEFT\$ and RIGHT\$, INKEY\$ is also a function built into BASIC I that is used for character strings. Its function is to read keys typed in from the typewriter-like console.

Line 350 will take the character you type in while you're playing the game and check whether it's a space. A space is indicated by a blank enclosed between two double quotation marks. An example. If you enclose an 'A' in quotation marks, the M5 would be expecting you to press the A key.

Keep in mind when you use the INKEY\$ function, it doesn't wait for you to type in a character. If you haven't typed anything in, the M5 will go onto the next BASIC I statement.

Try it.

```
350 IF INKEY$="A" THEN GOTO 10
```

indicates an 'A' key

Now look at line 350 and 360 as a set.

```
350 IF INKEY$=" " THEN GOTO 10
360 GOTO 350
```

Line 360 will always jump to line 350. In essence, it's an infinite loop that'll go on forever until a space is typed in from the console. When that happens, the M5 computer will jump to line 10 which starts the dice program over again. Remember, if you haven't typed anything in, the IF test in statement 350 is not satisfied and the GOTO 10 is not executed. In this case, line 360 is executed next which jumps back to line 350. This continues until you type in a space, " ".

Notice that if another character (which is not a space) is typed in, the dice game ignores it and continues to wait for a space.

Now look at lines 500 to 580.


```
500 REM Test dice/draw dice subroutines
510 IF DIE=1 THEN GOSUB 1000
520 IF DIE=2 THEN GOSUB 1100
530 IF DIE=3 THEN GOSUB 1200
540 IF DIE=4 THEN GOSUB 1300
550 IF DIE=5 THEN GOSUB 1400
560 IF DIE=6 THEN GOSUB 1500
580 RETURN
```

Let's look at line 580 first. Since it contains a RETURN statement, we know a number of statements preceding it make up a subroutine. In fact, since we wrote the program ourselves, we know the subroutine starts from line 500 and ends on line 580. Basically, this subroutine checks the value of the rolled die and calls another subroutine to draw the die.

Line 510 checks whether the rolled die is a '1'; if so, the M5 calls the subroutine beginning on line 1000. If the die is not a '1', the M5 checks whether the die is a '2' in line 520. If so, it calls the subroutine starting on line 1100. This goes on until the right subroutine is called.

Going onto lines 1000 to 1550, we see these lines actually consist of six subroutines, one subroutine per die. After drawing a die, a RETURN statement causes the M5 to return to the line after the calling statement. For instance, if the subroutine beginning on line 1100 is called to draw a die with two pips (using line 520), the RETURN statement in line 1150 causes the M5 to jump back to line 530 (the line following line 520). In other words, even if a '1' is rolled and the subroutine beginning on line 1000 is called, lines 520 to 560 are still executed.

Remember, if you want to play the game once again, press the SPACE key or the A key, depending on which one you indicated in line 350. When you want to stop playing the game, press the SHIFT key and RESET key at the same time.

10-2. Rolling Dice

Let's get a little more advanced and roll the dice on the screen. Add the next few program statements. But first, one pointer. When you type in the characters between the quote marks in line 160, press the SHIFT, CTRL and ↑ keys simultaneously. This generates a special character that prevents the screen from scrolling. Look in Appendix F for other special characters and what they're used for. They can also be enclosed in double quotation marks.

```
150 IF INKEY$=" " THEN GOTO $ROLL
160 PRINT "↑↑↑↑↑↑↑↑" ← SHIFT CTRL ↑
170 GOTO 100
205 $ROLL
```

Line 150 is expecting you to input a SPACE key. It's the same function you used before in line 350. This time, instead of jumping to line 10, we'll jump to a statement labeled with \$ROLL. No, it's not a character string variable, it's used in the same way as line numbers. You can make the M5 jump to a statement simply by specifying a GOTO or a GOSUB keyword followed by the statement label. (Don't confuse the GOTO and GOSUB statements; they function differently.) In our example program, if a SPACE key is typed in, line 205 is the next line executed. Do you understand why? Yes. When the INKEY\$=" " test is satisfied, the THEN GOTO \$ROLL statement is executed which brings us to the \$ROLL label (line 205).

Within the scope of our program, what do the four statements above do? They allow you to roll your own die. What actually happens is the M5 computer will continue to roll your die for you until you tell it to stop. You can stop it by pressing the SPACE key. Line 160 prevents the screen from scrolling while the M5 computer is rolling your die. Afterwards, the computer rolls its own die and the result of the game is displayed.

10-3. Coloring the Die Face

Let's color the die pips, so add the next couple of statements.

```
20 PRINT "■"
30 FOR I=4 TO 6
40 STCHR "40404040404040" TO &E1,I
50 NEXT I
```

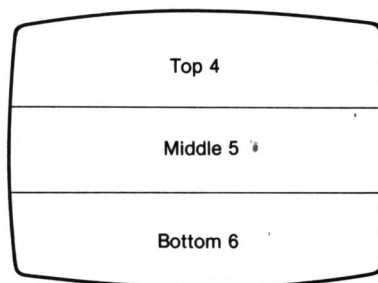
Line 20 changes the screen to the graphics mode, also known as the GII mode. The STCHR statement in line 40 is used to specify colors on the screen.

<pre>40 STCHR "40404040404040" TO &E1, I</pre>
<div style="display: flex; justify-content: space-around; align-items: center;"> <div style="text-align: center;"> Color choice </div> <div style="text-align: center;"> What is colored </div> <div style="text-align: center;"> Screen position </div> </div>

Color choice—Refer to the color code table in Appendix B to choose the color you want to display.

What is colored—Specify the ASCII code of the characters you want to color.

Screen position—The GII mode divides the screen into three parts, the upper, middle and bottom portions. You can choose any color for each of these parts. Look at the figure below. The upper part is assigned the value four, while the middle and bottom parts are given five and six respectively.



In this case, if you make I equal to 4, only the upper part of the screen will be blue. The middle and bottom portions of the screen will not be colored. Line 40 uses this feature to color the pips on the die.

So after building onto our small program we started with in chapter 8, we now have a program that allows you to roll your own die that competes with a die rolled by the computer. We can also display the die as it rolls with colored pips.

When you venture from this manual and begin composing your own programs, keep this technique in mind. Start with an idea. Think in broad terms. What do I actually want my computer program to do? Will I want to add onto it in the future? What kind of user input do I want? Then think about how to break it down to smaller functions until you can write the BASIC I to perform your function. But before you actually write the BASIC, ask yourself these questions. Am I going to be able to build onto this later? Have I made this program flexible enough? Have I put in enough REMark statements? Can someone else understand my program? This last question is often a good test to find out if you'll also be able to understand your program later. You may be surprised to find out how many times you'll be referring to your old BASIC programs.

11. Conclusion

How was that? Not so bad?

We hope this was a good introduction to the BASIC computer language. Without being too wordy, this manual should have given you a good feeling about what you can and cannot do with computers, especially using BASIC.

For beginners, this may have been slightly difficult. Don't worry. Keep plugging away. Take old programs and modify them. Then write some small programs that interest you. Before long, you'll be composing some pretty elaborate software. This is the probably the best way to improve your understanding of BASIC I.

For those of you who feel adventuresome, look at the appendices. They have some very useful information that'll allow you to take advantage of some advanced BASIC features. You can also enjoy the UFO program in Appendix A that uses some of these high-level BASIC I techniques. Type it in. Play with it. Look at the software. See if you can think of ways to make it more interesting. Modify it. Play it again. Write your own games. Your M5 can work better for you if you know what it can do. You're the boss.

The next time you're sitting under the stars thinking about new applications for your M5 computer, try to be free. Let your imagination wander. The current age of high-speed information processing is still a fairly new innovation. It's a baby-science guided and motivated by people like you. Learn what a computer can do and demand the computer manufacturers of today know and meet your needs for today and tomorrow. And talk to us. We're tuned into your needs.

Appendix A—UFO Game

All right! Now we're ready for the big time. This game tests your skill in shooting down invaders while avoiding meteors. Try it!

After you key it in, it's a good idea to save it on tape before you do anything else. Call it 'UFO'.

This is a fairly long program. If you accidentally turn off power or input a NEW command, you'll have to type in the whole program again. Save it on tape, then you can retrieve it any time you want. Then type in the RUN command and you're on your way.

When you've become proficient at this game, try to think of ways to make it more exciting. Modify it. Play it. Create your own game.

If the UFO game doesn't work properly, use the LIST command to look at your program. Check that it's identical to the following listing.

How to play

- To move the flying saucer up ↑
- To move the flying saucer down ↓

Score

- You'll get 10 points each time you knock out an invader.
- After you get 1200 points, the number of meteors will increase and you'll get 100 points for each invader you knock out.

To execute the UFO Game program again, input the command RUN followed by RETURN. Pushing FUNC R will not execute the game. After the UFO Game is finished, turn the power off and on again before saving or loading another program.

Note

If you don't knock out any invaders for a long time, the Game will automatically end and a READY message will be displayed.

```
5 REM ufo game
10 READ SC
20 OUT &20, &F6
30 LET X=5
40 LET A=PEEK(&701A)
50 POKE &701A, A AND &EF
60 PRINT "UFO!";:LET SL=100
70 VIEW 0,5,31,23:LET STC=&40
100 LET UX=32:LET UY=0:LET LV=3
110 GOSUB 10000
120 FOR LP=0 TO 1
140 PRINT CURSOR(30,RND(18));"●"
```

```

141 IF RND(9)<4 THEN PRINT CURSOR(30,RND(18));"4";
142 FOR I=1 TO LV
150 PRINT "0";
154 FOR J=1 TO 2
160 GOSUB 1000:GOSUB 2000
164 NEXT J:NEXT I
170 LET SL=SL-1 *
END 180 IF SL<0 THEN END *
190 IF LP=0 THEN GOTO 220
200 STCHR "183c7edbdb7e8142" TO &7F,1
210 GOTO 300
220 STCHR "183c7edbdb7e4281" TO &7F,1
300 NEXT
400 GOTO 120
POHYB 1000 LET S=PEEK(&702B)
1010 IF S=51 THEN GOSUB 1100
1030 IF S=46 THEN GOSUB 1200
1060 LOC 0 TO UX,UY
1070 RETURN
1100 IF UY>=2 THEN LET UY=UY-2
1110 RETURN
1200 IF UY<=140 THEN LET UY=UY+2
1210 RETURN
TRLEFA 2000 LET UAD=UY/8*32+UX/8*&3800+32*5
2010 LET UAD=UAD+(UY AND 4)*8+(UX AND 4)/4
2016 LET BCHR=VPEEK(UAD)
2020 IF BCHR=225 THEN GOTO 3100
2030 IF BCHR=127 THEN GOTO 2200
2040 RETURN
ZASAH UFO 2200 VIEW
2210 LET O=O+SC:LET SL=SL+10 *
2215 PRINT "0"
2220 PRINT CURSOR(9,2);O;
2230 VPOKE UAD,32
2240 LET UX=UX+1
2250 IF UX<160 THEN GOTO 2400
2260 IF UX<=1 THEN GOTO 2400
2270 LET LV=LV-1
2280 LET UX=32
2290 READ SC
2300 LET STC=STC*&40
2310 VPOKE &3B9C,STC
2400 VIEW 0,5,31,23

```

```

EXPLOSE 2410 RETURN
3100 VPOKE UAD,32
3110 LOC 0 TO 256,0
3130 FOR BL=1 TO 3
3140 FOR BB=1 TO 2
3150 LOC BB TO UX,UY
3160 LOC 3-BB TO 256,0
3170 LOC BB TO UX,UY
3200 GOSUB 30000
3210 NEXT BB
3220 NEXT BL
3230 LOC 2 TO 256,0
3236 VIEW
3240 LET X=X-1
3260 PRINT CURSOR(8+X,3); "L";
3280 IF X=0 THEN GOTO 4000
3300 GOTO 2240
4000 PRINT CURSOR(12,10); "Game Over"
4010 OUT &20,&FF
4020 END
10000 REM
10050 VIEW
10110 VPOKE &3B9C,STC
10130 STCHR "182442427ffffff66" TO 0
10136 STCHR "182442427ffffff66" TO &82,1
10140 STCHR "3c66e79999e7663c" TO 1
10150 STCHR "c3991866661899c3" TO 2
10160 STCHR "183c7edb7e4281" TO &7F,1
10200 FOR I=0 TO 2
10210 SCOD I,I
10220 SCOL I,2^(I+1)
10250 NEXT I
10260 PRINT CURSOR(3,2); "Score: ";
10266 PRINT CURSOR(3,3); "Left : ♣♣♣♣♣";
10270 PRINT " ♠";
10280 FOR I=1 TO 30
10290 PRINT "=";
10300 NEXT I
10310 PRINT " ";
10320 FOR I=1 TO 3
10330 PRINT "|";CURSOR(31,I); "|";
10340 NEXT I
10350 PRINT "L";

```



```
10360 FOR I=1 TO 30
10370 PRINT "─";
10380 NEXT I
10390 PRINT "┘"
10510 VIEW 0,5,31,23
11000 RETURN
30000 OUT &20,&9F:OUT &20,&BF
30010 OUT &20,&E7:OUT &20,&F0
30020 FOR FRQ=1 TO 20
30030 OUT &20,&C0
30040 OUT &20,FRQ
30045 FOR T=1 TO 60:NEXT
30050 NEXT FRQ
30060 OUT &20,&F6
30070 RETURN
31000 DATA 10,100,200
31010 END
```


Appendix B—Color Codes

Remember the dice program in Chapter 10? Remember using the STCHR statement? We colored the screen with this command. This appendix lists all the colors you can choose.

For example if you use:

```
STCHR "80 80 80 80 80 80 80 80" TO &E1,4
```

Color code
Character code

the &E1 character code will be colored red. Look down at the table for color code 8. You can see that it's red. But how do you know what the &E1 character code is? Refer to the chart in Appendix C. Look across the top row and find 'E', then go down until you reach the '1' row. See the shaded circle? That's the graphics character for &E1.

If you would rather color around the circle, in other words, generate a ☐ character, change the '80 80 ... 80' to '08 08 ... 08', like this (numbers are reversed).

```
STCHR "08 08 08 08 08 08 08 08" TO &E1,4
```

That's right. Changing the 80's to 08's leaves the character indicated by the character code not colored. Rather, the space around it is colored. This is true for any of the characters and colors. If you want the character colored, specify the color in the first character. If, not specify the color in the second character.

For this function, we're going to use the GII mode.

COLOR CODE TABLE

Color code	0	1	2	3	4	
Color	No color	Black	Green	Light Green	Deep Blue	
Color code	5	6	7	8	9	
Color	Light Blue	Deep Red	Cyan	Red	Light Red	
Color code	A	B	C	D	E	F
Color	Deep Yellow	Light Yellow	Deep Green	Purple	Gray	White

Appendix C—Character Codes

This appendix lists what is known as the ASCII representation of all characters that can be stored and displayed by the M5 computer.

To use this appendix, find the character you want to display. Then look up at the row of numbers and letters across the top and find the one that lines up with your character. Now look to the left at the leftmost column of numbers and characters for the corresponding number or character. Then combine these two characters. The one you found first is followed by the second.

Let's look at three examples. Verify that they're correct in the table below.

Character to display	Character code
\$	&24
H	&48
+	&2B

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0																
1																
2																
3																
4																
5																
6																
7																
8																
9																
A																
B																
C																
D																
E																
F																

Appendix D—Commands

The commands in this appendix will prove themselves to be useful in utilizing your M5 better. Try them out. Become familiar with them.

Direct Commands

	Command	Usage	Examples
1	AUTO	AUTO M,N	AUTO 10 . AUTO 100,10
2	CLEAR	CLEAR	CLEAR
3	CLS	CLS	CLS
4	CONT	CONT	CONT
5	DEL	DEL M,N	DEL 10,50
6	LIST	LIST M,N	LIST 10 LIST 10,50 LIST,50
7	LIST #2,	LIST #2,M,N	LIST #2,LIST #2,10,100
8	NEW	NEW	NEW
9	RUN	RUN RUN N	RUN RUN 100

Input/Output Commands

	Command	Usage	Examples
1	CHAIN	CHAIN"file name"	CHAIN"UFO"
2	DATA	DATA N,M, ---	DATA 1,34, "CD"
3	INPUT	INPUT N,M\$,--	INPUT A,B\$
4	OLD	OLD"file name"	OLD"UFO" OLD"UFO",100
5	OUT	OUT N,M	OUT &20,&3F
6	PRINT	PRINT N,M\$,---	PRINT A,B\$, PRINT"UFO",33
7	PRINT #2,	PRINT #2,	PRINT #2,
8	READ	READ N,M,---	READ A,B,C\$
9	RESTORE	RESTORE N	RESTORE 100
10	SAVE	SAVE"file name"	SAVE"UFO"
11	SAVE	SAVE"file name",&M,1	SAVE"UFO",&2000,&3FFF,1
12	TAPE	TAPE	TAPE
13	VERIFY	VERIFY"file name"	VERIFY"UFO"

	Comments
1	Automatically assigns you line numbers for program input
2	Clears all variables
3	Clears the screen
4	Resume execution of the program that was stopped by the STOP command. Execution resumes from where it stopped.
5	Delete part of a program
6	Display a program or part of a program on the screen
7	Print a program on the printer
8	Erase the program stored in memory
9	Begin execution of a program either from the beginning or from a specified line number

	Comments
1	Retrieve a program saved on tape. Specify the file name of the program. Remember each time you execute a program again, all the variables start fresh, they're not retained.
2	Specify data used by the READ statement
3	Use the M5 keyboard to input data and assign it to a variable(s)
4	Read a program or data (or part of a program or data) stored on tape
5	Output the value M to the I/O board (input/output equipment) specified by N
6	Display a character string and/or variable(s) on the screen
7	Print a character string and/or variable(s) on the printer
8	Read the data specified by DATA sequentially
9	Restore the data specified in the DATA statement and read by the READ statement. A RESTORE 100 reads DATA on line 100.
10	Save a program named "file name" on tape
11	The data displayed on the screen is saved on tape. (It takes about one minute.)
12	Refer to the application program (refer to the M5 User's Guide)
13	Confirm whether a file has been saved correctly. Display the file names of programs stored on tape.

Program Commands

	Command	Usage	Examples
1	CALL	CALL N	CALL
2	DIM	DIM N(M) DIM N(M)\$	DIM A(10) DIM A(10)\$
3	END	END	END
4	FOR-TO-STEP	FOR N=M TO A STEP B	FOR N=1 TO 100 STEP 2
5	GOSUB	GOSUB N GOSUB \$N	GOSUB 100 GOSUB \$LAB
6	GOTO	GOTO N GOTO \$N	GOTO 100 GOTO \$LAB
7	IF-THEN-ELSE	IF N=M THEN C ELSE C	IF A=1 THEN GOTO 50 ELSE B=1
8	LET	LET N=X	LET M=N+1
9	NEXT	NEXT N	NEXT N
10	POKE	POKE N,M	POKE &2000,&3F
11	RANDMIZE	RANDMIZE	RANDMIZE
12	REM	REM	REM your comments
13	RETURN	RETURN	RETURN
14	STOP	STOP	STOP

Note—A complete discussion of the POKE command is not appropriate here. Do not try to use it without more knowledge of the M5 machine language. There's a possibility you could cause your M5 to malfunction.

	Comments
1	Call a program specified by N
2	Sets up a group of M related data fields referred to by N(M)
3	Stops program execution
4	Sets up a program loop. The example loops from N = 1 until it reaches 100 in steps of 2. Once N exceeds 100, the loop stops.
5	Jump to the desired subroutine
6	Jump to the desired line number
7	Condition statement. In the example, if A is 1, a jump is made to line 100, otherwise B = 1.
8	Assignment statement
9	Bottom statement of a loop started by a FOR-TO-STEP statement
10	Write the value specified by M into the address specified by N
11	Initialize the random number generator
12	Comments to help you understand your program better
13	Jump to the statement following the statement that called this subroutine
14	Stop the executing program for just a moment. Use CONT to resume execution.

Screen Control Commands

	Command	Usage	Examples
1	LOC	LOC N TO X,Y	LOC 0 TO 256,192
2	MAG	MAG N	MAG 3
3	SCOD	SCOD N,C	SCOD 0,255
4	SCOL	SCOL N,C	SCOL 0,4
5	STCHR	STCHR " " TO N,M	STCHR"4040404040404040" TO &E1,6
6	VIEW	VIEW XD,YD,X1,Y1	VIEW 0,0,32,24
7	VPOKE	VPOKE N,M	VPOKE &3B80+C/8,4*16+0

	Comments
1	Specify the position of a sprite on the screen N = 0 to 31 (Sprite number) X = -32768 to 32768 (Dot number) Y = -32768 to 32768 (Dot number) Note that you can only see the range X = 0 to 256 and Y = 0 to 192, on the screen.
2	Change the size of a sprite (N = 0 to 3)
3	Specify a sprite in the graphics mode N = 0 to 31 (Sprite number) C = 0 to 255 (character code)
4	Assign colors to a sprite N = 0 to 31 (Sprite number) C = 0 to 15 (Color code)
5	Manage characters by their assigned character codes " " = character pattern or color code N = character code (Refer to Appendix C) M = 1 to 3 (Character pattern management) = 4 to 6 (Color management)
6	Divide the screen into squares specified by (X0,Y0) and (X1,X1)
7	Write the value specified by M into the address specified by N

Appendix E—Functions

Functions

	Command	Usage	Examples
1	ASCII	ASCII (X\$)	A=ASCII("A")
2	CHR\$	CHR\$ (N)	A=CHR\$(97)
3	HEX\$	HEX\$ (N)	A\$=HEX\$(16)
4	INKEY\$	INKEY\$	A\$=INKEY\$
5	LEFT\$	LEFT\$ (X\$, N)	A\$=LEFT\$("ABCD", 2)
6	LEN	LEN (X\$)	A=LEN("ABCD")
7	MID\$	MID\$ (X\$, N, M)	A\$=MID("ABCD", 2, 2)
8	RIGHT\$	RIGHT\$ (X\$, N)	A\$=RIGHT\$("ABCD", 3)
9	VAL	VAL (X\$)	A=VAL("5")
10	RPT\$	RPT\$ (N, X\$)	A\$=RPT\$(3, X\$)

Operating Commands

	Command	Usage	Examples
1	CURSOR	CURSOR (X, Y)	PRINT CURSOR(16, 12); A
2	ERR	ERR	PRINT ERR
3	ERRL	ERRL	PRINT ERRL
4	ERRL\$	ERRL\$	PRINT ERRL\$
5	PEEK	PEEK (N)	A=PEEK(&3800)
6	TAB	TAB (X)	PRINT TAB(10); A
7	VPEEK	VPEEK (N)	A=VPEEK(&3800+Y*32+X)

	Comments
1	Change the character enclosed by (" ") into its corresponding ASCII code
2	Change the ASCII code enclosed by () into its corresponding character. This function is the opposite of ASCII(" "). Example: ASCII("a") = 97 CHR\$(97) = a Note these numbers are the decimal equivalents of the hexadecimal numbers in Appendix C.
3	Change the value enclosed in () into its hexadecimal (base 16) value.
4	Read one character input from the keyboard.
5	Take N characters off the left end of character string X\$
6	Counts the number of characters in the character string specified between ()
7	Take M characters from character string X\$, starting from character number N (starting from the left).
8	Take N characters off the right end of character string X\$
9	Returns the numeric equivalent of a character string
10	Returns N repetitions of character string

	Comments
1	Set the cursor at the position designated by (X,Y) X = 0 to 31 (GI and GII modes) X = TO 39 (Text mode) Y = 0 to 23
2	Print the last error number encountered
3	Print the last erroneous line number encountered
4	Print the last erroneous label encountered
5	Allows you to look at the RAM address specified by ()
6	Tabs the cursor by printing spaces to position X
7	Allows you to look at the VRAM address specified by ()

Value Functions

	Command	Usage	Examples
1	ABS	ABS (N)	A=ABS (-5)
2	FRE	FRE (N)	A=FRE (0)
3	INP	INP (N)	A=INP (0)
4	NUM\$	NUM\$ (X\$)	A\$=NUM\$ (5)
5	MOD	M MOD N	A=8 MOD 5
6	RND	RND (N)	A=RND (10)
7	SGN	SGN (N)	A=SGN (3)
8	TIME	TIME	PRINT TIME

	Comments
1	Returns the absolute value of the variable enclosed by ()
2	Tells you memory statistics FRE(0) ... Amount of memory already used FRE(1) ... Amount of memory left
3	Read one byte of data from the I/O board (input/output equipment specified by the value enclosed in ()
4	Change the numeric value enclosed in () into its character equivalent
5	Divide 8 by 5 and returns the remainders
6	Return a random number between 0 and the value enclosed in ()
7	Return the sign of the number enclosed in (). SGN(negative number) = -1 SGN(0) = 0 SGN(positive number) = 1
8	Returns the time the M5 computer has been powered on, up to 65535 seconds.

Appendix F—Control Codes

These are functions you can activate by pressing the CTRL key and the key below. But note that there is a difference between directly inputting them after a READY prompt and using them in a program.

When using control functions directly after a READY prompt, press the key below while keeping the CTRL key down.

When using control functions in a program (for example, in a PRINT statement), first press the CTRL and SHIFT keys down before pressing the keys indicated below. Also enclose this character in double quotes. You can also specify character codes in the CHR\$ function. For example, PRINT CHR\$(&0). The '&' signifies a character code follows.

Keyboard Key	Base 10	Base 16	Function	Program Usage Display
	0	00	Ignore	
A	1	01	Ignore	
B	2	02	Return cursor to beginning of current line	
C	3	03	Scroll screen display down	
D	4	04	Shift screen display left	
E	5	05	Scroll screen display up	
F	6	06	Shift screen display right	
G	7	07	Bell	
H	8	08	Backspace	
I	9	09	Tab the cursor eight spaces	
J	10	0A	Move cursor down one line	
K	11	0B	Move cursor to home position	
L	12	0C	Clear screen display (Cannot be used as a direct command)	
M	13	0D	Same as RETURN key	
N	14	0E	Move cursor to beginning of next line	
O	15	0F	Change to standard mode	
P	16	10	Change to insert mode	
Q	17	11	Change to multi-color mode	
R	18	12	Change to GII graphics mode	
S	19	13	Change to GI graphics mode	
T	20	14	Return to text mode	
U	21	15	Change to visible screen	
V	22	16	Alternates between the visible and invisible screens, input is sent to the displayed screen	
W	23	17	Same as RETURN key	
X	24	18	Delete characters to the right of cursor	
Y	25	19	Alternates between the visible and invisible screens only	
Z	26	1A	Writes input to the alternate screen	
[27	1B	Ignore	
:	28	1C	Right arrow	
];	29	1D	Left arrow	
^	39	1E	Up arrow	
./	40	1F	Down arrow	

Appendix G—ASCII Character Codes

The following table lists all characters that your M5 can store and display. There are ways to specify the ASCII equivalents using the CHR\$ function. One way is to use the base 10 numbering system and the other is base 16. When using base 16, precede the character code with a '&'

DEC	(HEX)	CODE	DEC	(HEX)	CODE	DEC	(HEX)	CODE
32	(20)		71	(47)	G	110	(6E)	n
33	(21)	!	72	(48)	H	111	(6F)	o
34	(22)	"	73	(49)	I	112	(70)	p
35	(23)	#	74	(4A)	J	113	(71)	q
36	(24)	\$	75	(4B)	K	114	(72)	r
37	(25)	%	76	(4C)	L	115	(73)	s
38	(26)	&	77	(4D)	M	116	(74)	t
39	(27)	'	78	(4E)	N	117	(75)	u
40	(28)	(79	(4F)	O	118	(76)	v
41	(29))	80	(50)	P	119	(77)	w
42	(2A)	*	81	(51)	Q	120	(78)	x
43	(2B)	+	82	(52)	R	121	(79)	y
44	(2C)	,	83	(53)	S	122	(7A)	z
45	(2D)	-	84	(54)	T	123	(7B)	{
46	(2E)	.	85	(55)	U	124	(7C)	
47	(2F)	/	86	(56)	V	125	(7D)	}
48	(30)	0	87	(57)	W	126	(7E)	~
49	(31)	1	88	(58)	X	127	(7F)	4
50	(32)	2	89	(59)	Y	128	(80)	■
51	(33)	3	90	(5A)	Z	129	(81)	○
52	(34)	4	91	(5B)	[130	(82)	♠
53	(35)	5	92	(5C)	\	131	(83)	♣
54	(36)	6	93	(5D)]	132	(84)	~
55	(37)	7	94	(5E)	^	133	(85)	⬆
56	(38)	8	95	(5F)	_	134	(86)	⬆
57	(39)	9	96	(60)	`	135	(87)	⬆
58	(3A)	:	97	(61)	a	136	(88)	⬆
59	(3B)	;	98	(62)	b	137	(89)	⬆
60	(3C)	<	99	(63)	c	138	(8A)	⬆
61	(3D)	=	100	(64)	d	139	(8B)	⬆
62	(3E)	>	101	(65)	e	140	(8C)	⬆
63	(3F)	?	102	(66)	f	141	(8D)	⬆
64	(40)	@	103	(67)	g	142	(8E)	⬆
65	(41)	A	104	(68)	h	143	(8F)	⬆
66	(42)	B	105	(69)	i	144	(90)	⬆
67	(43)	C	106	(6A)	j	145	(91)	⬆
68	(44)	D	107	(6B)	k	146	(92)	■
69	(45)	E	108	(6C)	l	147	(93)	■
70	(46)	F	109	(6D)	m	148	(94)	⬆

DEC	(HEX)	CODE	DEC	(HEX)	CODE	DEC	(HEX)	CODE
149	(95)	!	192	(C0)		235	(EB)	+
150	(96)	■	193	(C1)	⌘	236	(EC)	▲
151	(97)	■	194	(C2)	⌘	237	(ED)	▼
152	(98)	◀	195	(C3)	⌘	238	(EE)	▶
153	(99)	▼	196	(C4)	⌘	239	(EF)	▶
154	(9A)	▲	197	(C5)	⌘	240	(F0)	-
155	(9B)	▶	198	(C6)	⌘	241	(F1)	-
156	(9C)	⌘	199	(C7)	⌘	242	(F2)	■
157	(9D)	⌘	200	(C8)	⌘	243	(F3)	■
158	(9E)	⌘	201	(C9)	⌘	244	(F4)	!
159	(9F)	⌘	202	(CA)	⌘	245	(F5)	!
160	(A0)	!	203	(CB)	⌘	246	(F6)	■
161	(A1)	⌘	204	(CC)	⌘	247	(F7)	■
162	(A2)	⌘	205	(CD)	⌘	248	(F8)	■
163	(A3)	⌘	206	(CE)	⌘	249	(F9)	■
164	(A4)	⌘	207	(CF)	⌘	250	(FA)	⌘
165	(A5)	⌘	208	(D0)	⌘	251	(FB)	⌘
166	(A6)	⌘	209	(D1)	⌘	252	(FC)	▲
167	(A7)	⌘	210	(D2)	⌘	253	(FD)	▲
168	(A8)	⌘	211	(D3)	⌘	254	(FE)	▲
169	(A9)	⌘	212	(D4)	⌘	255	(FF)	▲
170	(AA)	⌘	213	(D5)	⌘			
171	(AB)	⌘	214	(D6)	⌘			
172	(AC)	⌘	215	(D7)	⌘			
173	(AD)	⌘	216	(D8)	⌘			
174	(AE)	⌘	217	(D9)	⌘			
175	(AF)	■	218	(DA)	⌘			
176	(B0)	!	219	(DB)	⌘			
177	(B1)	⌘	220	(DC)	⌘			
178	(B2)	⌘	221	(DD)	⌘			
179	(B3)	⌘	222	(DE)	⌘			
180	(B4)	⌘	223	(DF)	⌘			
181	(B5)	⌘	224	(E0)	⌘			
182	(B6)	⌘	225	(E1)	⌘			
183	(B7)	⌘	226	(E2)	⌘			
184	(B8)	⌘	227	(E3)	⌘			
185	(B9)	⌘	228	(E4)	⌘			
186	(BA)	⌘	229	(E5)	⌘			
187	(BB)	⌘	230	(E6)	⌘			
188	(BC)	⌘	231	(E7)	⌘			
189	(BD)	⌘	232	(E8)	⌘			
190	(BE)	⌘	233	(E9)	⌘			
191	(BF)	⌘	234	(EA)	⌘			

DEC → denotes base 10
 HEX → denotes base 16

Appendix H—Error Messages

Message	Error Type	Reason	Comments/Erroneous Examples
Err 1	FOR-TO-NEXT statement error	<ul style="list-style-type: none"> Missing NEXT statement No matching NEXT statement 	<pre>10 FOR A=1 TO 10 20 PRINT A 30 NEXT B 40 END</pre> <p>* line 30 should be NEXT A to match line 10</p>
Err 2	Statement error	Statement doesn't make sense	<pre>PRINT "C"</pre> <p>↑ * PRINT is misspelled</p>
Err 3	Subroutine error	<ul style="list-style-type: none"> CLEAR command was used in subroutine GOTO jumped into a subroutine RETURN statement found without a matching GOSUB 	<pre>10 GOTO 30 20 END 30 PRINT "A"} 40 RETURN }</pre> <p>* Line 10 should be GOSUB 30</p>
Err 4	READ, DATA statement error	<ul style="list-style-type: none"> Insufficient data Missing DATA statement 	<pre>10 READ A,B 20 DATA 44</pre> <p>* Needs one more data value</p>
Err 5	Variable type error	Unexpected variable type	<pre>10 A=RND(AB)</pre> <p>* RND expects an integer, not a character string</p>
Err 6	Overflow	The result is too large to store in the computer	<pre>PRINT 678*789</pre>
Err 7	Memory overflow	<ul style="list-style-type: none"> Too many subroutines Memory for variables exceeds limit 	<pre>DIM A(10000)</pre> <p>↑ * Too many elements in array A</p>
Err 8	Missing line number	GOTO or GOSUB is missing a matching line number	<pre>10 GOTO 100</pre> <p>* Line 100 is missing</p>
Err 9	Variable error	Unexpected variable value (should not be negative or zero)	<pre>DIM A(0) DIM B(-1)</pre> <p>* Bad arguments in both arrays</p>
Err 10	Conflicting specification	A variable attribute has been specified twice	<pre>10 DIM A(6),A(3)</pre>
Err 11	Division by zero	Attempt to divide by 0	<pre>PRINT 5/0</pre>
Err 12	Incorrect statement	<ul style="list-style-type: none"> Statement not executable Use CONT command if you want to execute past bad statement 	<pre>PRINT 5/0</pre> <p>* If you want to print a character string, enclose in parentheses; PRINT "5/0"</p>
Err 13	Conflicting data types	Two different types of data are used together	* Resolve the conflict

Err 14	String overflow	Insufficient memory for a character string	* Reduce the number of characters
Err 15	String error	<ul style="list-style-type: none"> • Character string too long • More than 256 characters passed to a string manipulation function • Too many character strings specified • Incorrect string assignment statement 	<pre>10 A\$="abcdefghij" 20 B\$="klmnopqrst" 30 C\$=A\$+B\$</pre> <p>* Resultant string in C\$ exceeds eighteen characters</p>
Err 16	Variable error	The variable order is incorrect	* Correct the order
Err 17	Duplicate label	More than one label with the same string	<pre>10 GOTO \$A 50 \$A 70 \$A</pre> <p>* Delete incorrect label</p>
Err 18	Cassette Read Error	<ul style="list-style-type: none"> • During the reading of a cassette file data was incorrectly read. 	Check volume level, cassette tape and retry.

Appendix I—Mathematics Tutor Program

The following program can be simply typed into your M5 and enjoyably entertain you for hours. It's meant to drill you in the basic math skills of addition, subtraction, multiplication and division. For those of you who are learning these skills in elementary school, this program can help you to become faster. And for those of you who have been out of school for awhile, it might tax your ability as well. Likewise, the developer of this software was stretched to give correct answers quickly for many of the problems. Try it. You may be surprised at the results.

When you first run the program, a message will be displayed on the screen.

Operation (+, -, x, /, mix)?

This message is asking you what type of functions you want to be drilled on. So input one of the following:

Input Meaning

- + Addition problems
- Subtraction problems
- x Multiplication problems
- / Division problems
- mix MIXture of addition, subtraction,
 multiplication and division problems

The program will then ask you what skill level you want.

Level 0, 1 or 2?

The meanings of the levels are shown below.

Level	Meaning
0	Easy. The numbers in the problems will always be positive numbers less than 10. The answers though, may be greater than 9. Great for learning multiplication tables.
1	More difficult. The numbers in the problems will be less than 1300. An exception is multiplication, which is limited to 181. The answers are correspondingly long. All numbers are positive. How well can you do without scratch paper?
2	Most difficult. It is similar to level 1 except that negative numbers are introduced. Are you as good as you thought?

When you want to stop the game to go into another level or try another set of functions, press the SHIFT and RESET keys simultaneously, then type in R U N and the RETURN key.

After you type this program into the M5 memory, immediately save it. This will save you from having to type it in again if the power is cut off accidentally.

If you type it in and it doesn't work, check that it is identical to the following listing. Be sure all line numbers are correct. Verify that the GOTO line numbers are correct. Make sure P1 is typed in as P1, and not P2, etc. The graphics characters in lines 866 and 900 are the heart and the circle characters. To type the heart, get into the graphics mode and press the SHIFT and 'O' keys simultaneously. Then to type the circle, also get back into the graphics mode and press the SHIFT and '/' keys at the same time. Go back to the alphabet mode to type in the other text.

Try to understand this program. It's actually a review of everything you learned in this manual except for lines 872, 878, 930, 960, 980 and 1010. These lines contain OUT statements which send sounds to your M5 console. If you turn up your television while playing this game, you'll hear what it's all about. If you don't like it, turn down the sound on your television or even change the program. Modify it. Make it better. Share it with your friends. Work on it together. Write your own. It's all possible with your M5 computer!

Listing of Mathematics Teacher Software

```

10 REM mathematics teacher
20 CLS
40 INPUT "Operation(+,-,x,/,mix)";OPR$
50 REM check type of operation
51 REM -----
60 IF OPR$="+" THEN LET OP=1:GOTO 140
70 IF OPR$="-" THEN LET OP=2:GOTO 140
80 IF OPR$="x" THEN LET OP=3:GOTO 140
90 IF OPR$="/" THEN LET OP=4:GOTO 140
100 IF OPR$="mix" THEN LET OP=5:GOTO 140
110 CLS
130 GOTO 20
140 REM input skill level
150 INPUT "Level 0,1 or 2";LEVEL
160 IF LEVEL<0 THEN GOTO 140
170 IF LEVEL>2 THEN GOTO 140
190 REM errflg is an error flag
200 REM used when user misses
210 LET ERRFLG=0:LET MISSED=0
220 CLS
230 IF OPR$<>"mix" THEN GOTO 270
240 RANDOMIZE
250 LET OP=RND(3)+1
270 IF LEVEL>0 THEN GOTO 340
280 RANDOMIZE
290 LET P1=RND(9)
300 RANDOMIZE
310 REM prevent division by 0
312 IF OP=4 THEN LET P2=RND(9)+1:GOTO 320
314 LET P2=RND(9)
320 GOTO 530
340 REM level 1/2 random number
350 REM generator
360 RANDOMIZE
370 REM prevent overflow
380 IF OP=3 THEN LET P1=RND(181):GOTO 410
390 LET P1=RND(1300)
406 REM prevent division by 0
410 RANDOMIZE
415 IF OP=4 THEN LET P2=RND(1300)+1:GOTO 450
420 IF OP=3 THEN LET P2=RND(181):GOTO 450
430 LET P2=RND(1300)
450 IF LEVEL=1 THEN GOTO 530
460 REM level 2 number alteration

```

```
470 RANDOMIZE
480 LET FLAG=RND(1)
490 IF FLAG=1 THEN LET P1=P1*-1
500 RANDOMIZE
510 LET FLAG=RND(1)
520 IF FLAG=1 THEN LET P2=P2*-1
530 REM perform calculations
550 IF OP=1 THEN GOTO 580
560 IF OP=2 THEN GOTO 610
570 IF OP=3 THEN GOTO 650
575 IF OP=4 THEN GOTO 680
580 REM --- addition
590 LET ANSW=P1+P2
600 GOTO 710
610 REM --- subtraction
620 IF P2>P1 THEN LET C=P2:LET P2=P1:LET P1=C
630 LET ANSW=P1-P2
640 GOTO 710
650 REM --- multiplication
660 LET ANSW=P1*P2
670 GOTO 710
680 REM --- division
690 LET ANSW=P1/P2
700 LET TP=P1-P1/P2*P2
710 REM user input
720 CLS
730 IF ERRFLG=1 THEN PRINT CURSOR(15,15);
    "Try again ..."
740 LET ERRFLG=0
745 IF OP=4 THEN PRINT CURSOR(13,7);
    "quotient, remainder"
750 PRINT CURSOR(5,10);P1;
760 PRINT " ";
770 IF OP=1 THEN PRINT "+ ";
780 IF OP=2 THEN PRINT "- ";
790 IF OP=3 THEN PRINT "x ";
800 IF OP=4 THEN PRINT "/ ";
820 PRINT P2;
830 PRINT " = ";
840 IF OP=4 THEN INPUT USRINP,USRREM:GOTO 860
850 INPUT USRINP
860 IF USRINP<>ANSW THEN GOTO 880
862 IF OP<>4 THEN GOTO 866
864 IF USRREM<>TP THEN GOTO 880
```

```
866 PRINT CURSOR(13,15):"♥ Right you are!";
870 FOR C=207 TO 230
872 OUT &20,C
874 FOR T=1 TO 80:NEXT T
876 NEXT C
878 OUT &20,&FF
879 GOTO 210
880 LET MISSED=MISSED+1:LET ERRFLG=1
890 IF MISSED<3 THEN GOTO 710
900 PRINT CURSOR(12,15):"● Answer is ";ANSW;
910 IF OP=4 THEN PRINT ", ";TP
920 FOR C=175 TO 200
930 OUT &20,C
940 FOR T=1 TO 100:NEXT T
950 NEXT C
960 OUT &20,&FF
970 FOR C=239 TO 254
980 OUT &20,C
990 FOR T=1 TO 100:NEXT T
1000 NEXT C
1010 OUT &20,&FF
1020 GOTO 210
```



SORD COMPUTER CORPORATION

KYOBASHI K-1 BLDG., 7-12, YAESU 2-CHOME,
CHUO-KU, TOKYO 104, JAPAN

PHONE: (03)281-8118

TELEX: 2224225 (SORDIN J)