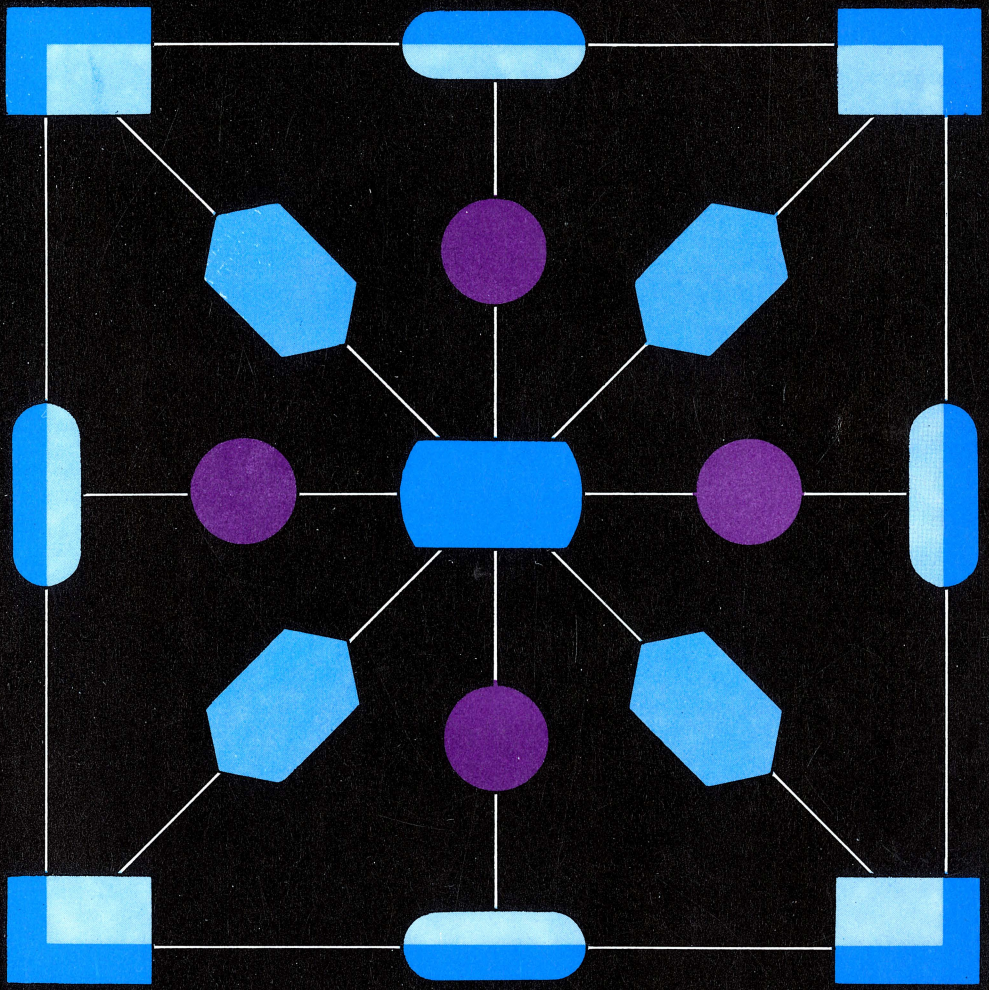
# Micro-Professor™

# THE INTRODUCTION TO BASIC PROGRAMMING



MULTITECH INDUSTRIAL CORPORATION

O-279, O-191



# Micro-Professor™

# THE INTRODUCTION TO BASIC PROGRAMMING



MULTITECH INDUSTRIAL CORPORATION

Copyright© 1982 by Multitech Industrial Corp.
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 the publisher

**Second Version**

**MULTITECH INDUSTRIAL CORPORATION**
OFFICE/ 977 MIN SHEN E. ROAD, TAIPEI, 105,
TAIWAN, R.O.C.
TEL:(02)769-1225(10 LINES)
TLX:23756 MULTIIC, 19162 MULTIIC.
FACTORY/5, TECHNOLOGY ROAD III,
HSINCHU SCIENCE-BASED INDUSTRIAL PARK
HSINCHU, TAIWAN, 300. R.O.C.
TEL:(035)775102(3 LINES)

# Micro-Professor™


**CHAPTER 1. COMPUTER** ..... 5

1 • 1 Computer ..... 7

1 • 2 Human Brain Vs Electronic Brain ..... 10

1 • 3 Hardware ..... 16

1 • 4 Computer Language ..... 19

**CHAPTER 2. START WITH THE MICROCOMPUTER** ..... 23

2 • 1 The Learning of Microcomputer ..... 25

2 • 2 Binary Number System ..... 27

2 • 3 Memory ..... 31

**CHAPTER 3. PRESENTING THE MPF-II** ..... 35

3 • 1 MPF-II Main Board ..... 38

3 • 2 Video Display ..... 39

3 • 3 Power Supply ..... 40

3 • 4 Cassette ..... 41

3 • 5 Printer

3 • 6 Remote Control Box

3 • 7 Software Cartridge

3 • 8 Inside the MPF-II ..... 42

3 • 9 Memory

**CHAPTER 4. HOW TO OPERATE THE MPF-II** ..... 45

**4.1 Power On** ..... 47

**4.2 Study the Keyboard** ..... 48

**4.3 The Cassette Recorder** ..... 54

**4.4 Loading** ..... 56

**4.5 A First Look at the Print Statement** ..... 59

**4.6 Abbreviated Print Statement** ..... 61

**CHAPTER 5. THE MPF-II AS A CALCULATOR** ..... 63

**5.1 Immediate Mode** ..... 65

**5.2 Addition and Subtraction**

**5.3 Multiplication and Division** ..... 66

**5.4 Exponentation** ..... 67

**5.5 Math Function** ..... 68

**5.6 MPF-II's Format for Numbers**

**CHAPTER 6. ELEMENTARY PROGRAMMING** ..... 71

**6.1 Statements** ..... 73

**6.2 Programmed Mode** ..... 75

**6.3 Line Numbers** ..... 76

**6.4 New, List, End and Home** ..... 80

**6.5 More Basic Statements** ..... 83

**6.6 Data** ..... 86

**6.7 Let and Variables** ..... 91

6・8 Input ..... 98
6・9 For····· Next Loops ..... 105
6・10 Advanced Editing Techniques ..... 118
6・11 Remarks ..... 121
6・12 Plotting and Developing in Color ..... 122
6・13 Handling String ..... 129
6・14 Random ..... 136
6・15 Array DIM ..... 140

CHAPTER 7. COMMANDS RELATING TO THE FLOW OF CONTROL ..... 143

7・1 Instruction Analysis Related to System ..... 145
7・2 Instructions Related to Execution ..... 147
7・3 Instructions Related to Editing and Format ..... 157
7・4 Control Instructions of Printer ..... 164

CHAPTER 8. VARIABLES, OPERATORS AND STRINGS ..... 165

8・1 Value Variable and its Operation ..... 168
8・2 String ..... 176
8・3 Array ..... 185

CHAPTER 9. INPUT AND OUTPUT INSTRUCTIONS ..... 191

9・1 Input and Output Instructions ..... 193

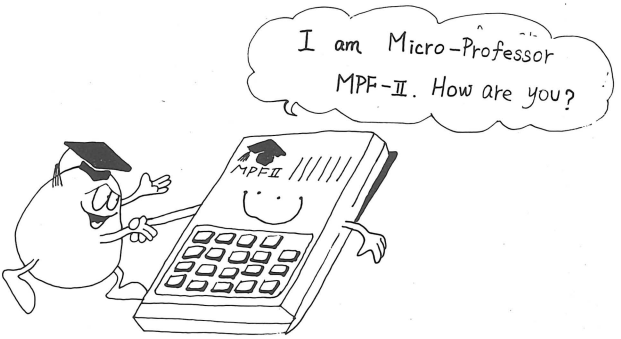
9.2 Plotting ..... 204
9.3 Low-resolution Instructions ..... 205
9.4 High-resolution Instructions ..... 208

CHAPTER 10. LOOP AND ARRAY ..... 215

10.1 ON.....GOTO ..... 218
10.2 GOSUB and RETURN ..... 222
10.3 ON.....GOSUB ..... 228
10.4 ONERR GOTO and RESUME ..... 230

CHAPTER 11. MATHEMATICAL FUNCTIONS ..... 233

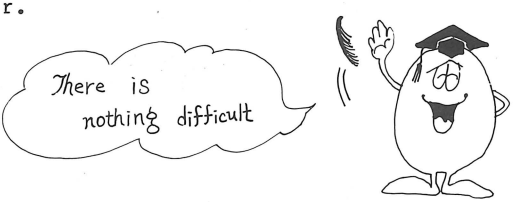
11.1 Trigonometric Functions ..... 236
11.2 The Plotting of Functions ..... 240
11.3 Other Functions ..... 245



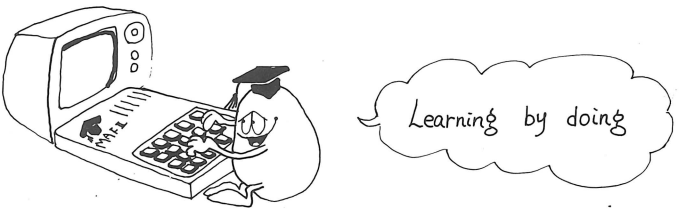

# WELCOME

The MPF-II (Micro-Professor II) is a low-cost R6502 based microcomputer system which provides an interesting and inexpensive way to enter into the computer world. Its built-in BASIC language makes it easy to create user programs. The color and sound capabilities help keep the user interested in programming.

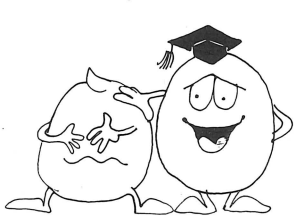
This manual is designed for people who want to learn to program in MPF-II BASIC. With this manual, and a MPF-II computer, and a bit of your time and attention, you will find that there is nothing difficult about learning how to program a computer.

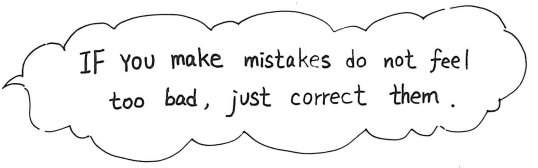



At first, as with anything new, programming will be unfamiliar, but this manual was designed to alleviate any apprehension you might have. First of all, there are no hidden secrets that you have to know before you can read this manual. Everything is revealed, and only one thing at a time is explained. If you start at the beginning, try everything as it comes along, and make up your mind to take your time, it is pretty much guaranteed that you will learn how to program.



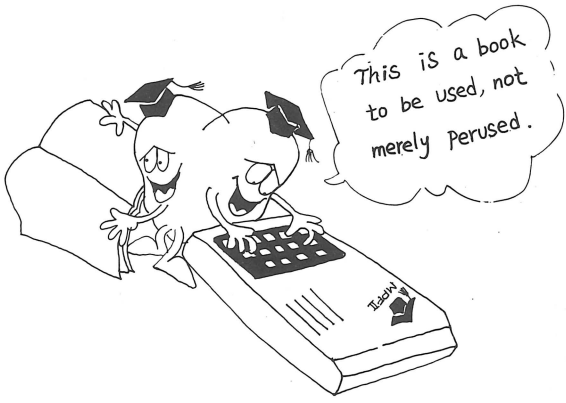
The real secret is taking your time and trying everything. You cannot learn programming by merely reading this or any other book. Like learning to ride a bicycle or play the fiddle, you must learn by doing. You must make mistakes and correct them, and not feel too bad when you do make mistakes.






If you already know how to program, a quick run through this book will make you familiar with the features of MPF-II BASIC. We suspect that you will be quite impressed with the ease of doing high resolution graphics and using the other features that make the MPF-II a fine computer for a wide variety of applications.

This book is a tutorial manual. We hope you enjoy using this manual as much as we have enjoyed writing it.




# INTRODUCTION

The Installation Manual will show you how to plug in your MPF-II (easy) and will be a guide as you learn to program it (also easy). If you are an Old Hand at programming, you will find some new features and conveniences in MPF-II BASIC that make programming a lot more enjoyable. If you are a Newcomer to programming, you will find many features and conveniences in MPF-II BASIC that make programing a lot of fun. But, if you are a Newcomer, be warned that programming, though not difficult, can only be learned by doing. More will be said on this topic later, but remember--this is a book to be used, not merely perused.

This manual will help you learn to program even if you have never touched a computer before. It describes the MPF-II itself and then covers the common peripheral devices.

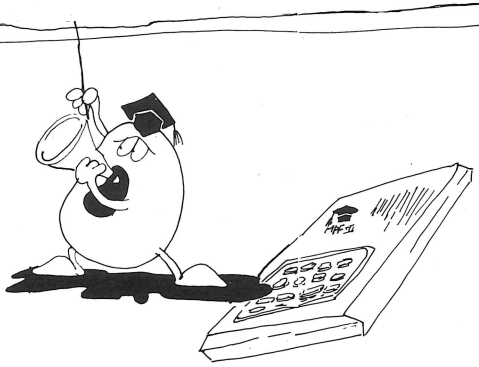
What is an MPF-II? How do you make it work? The first two chapters of this book answer those questions. You have probably noticed that the MPF-II system is made up of several pieces of equipment all connected together with wires and cables. The first chapter tells you what all the pieces are and what they are used for. The second chapter tells you how to operate each computer part. With this knowledge you are ready to use any of the ready-to-run programs that are widely available for word processing, financial analysis, bookkeeping, computer-aided instruction, and entertainment.

The next two chapters of the book teach you how to write your own BASIC programs on the MPF-II. Chapter 3 introduces MPF-II as a powerful calculator. Chapter 4 starts things off with a tutorial approach to the fundamentals of BASIC that are available on the MPF-II.

The rest of the chapters continue with a coverage of advanced programming topics and BASIC features.

# CHAPTER


1. Computer
2. Starting With the Micro-computer
3. Presenting the MPF-II
4. How to Operate the MPF-II
5. The MPF-II As a Calculator
6. Elementary Programming
7. Commands Relating to Flow of Control
8. Variables, Operators
9. Input, Output, and Graphics
10. Loop and Array
11. Some Math Functions

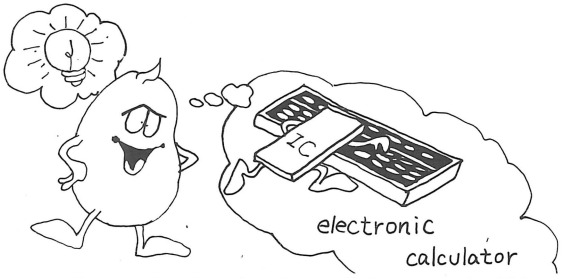



# 1

# COMPUTER

## 1 • 1 Computer

A modern computer is composed of electronic circuits. When we mention computer, we should first think about the word "computation".

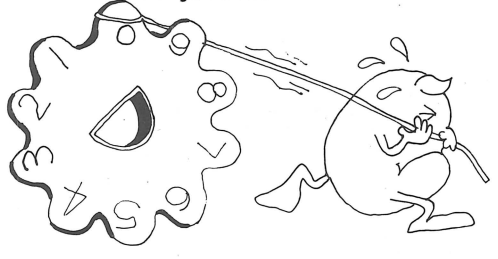



Since the beginning of human civilization, "computation" has become a kind of brain-racking and inevitable business. Therefore, mankind is striving for all kinds of computing devices to help solve the complicated tasks of computation. Yet, the Chinese abacus, which was invented three thousand years ago, can be regarded as the earliest ancestor in the history of computer.



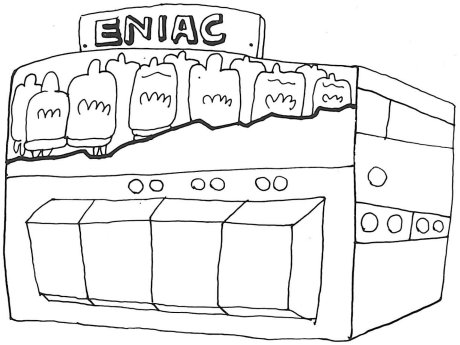
Earliest ancestor — chinese abacus

The development of computer parallels technological advances. Computer technology made little progress until in 17th century. In 1642, the French scientist Blaise Pascal invented the decimal cogwheeled mechanical calculator. The computer was then transferred from a manual device to mechanical one. Even so, the importance of abacus was still not down graded.

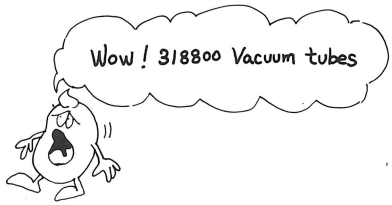


Pascal's ten — cogwheeled calculator

The real breakthrough of computer science should be attributed to the progress of elctronic technology. In 1945, the University of Pennsylvania produced the first compuer equipped with vacuum tube to solve complex computation. This is the first generation computer, which was called ENIAC, the acronym stands for Electronic Numerical Integrator and Calculator.




First generation computer

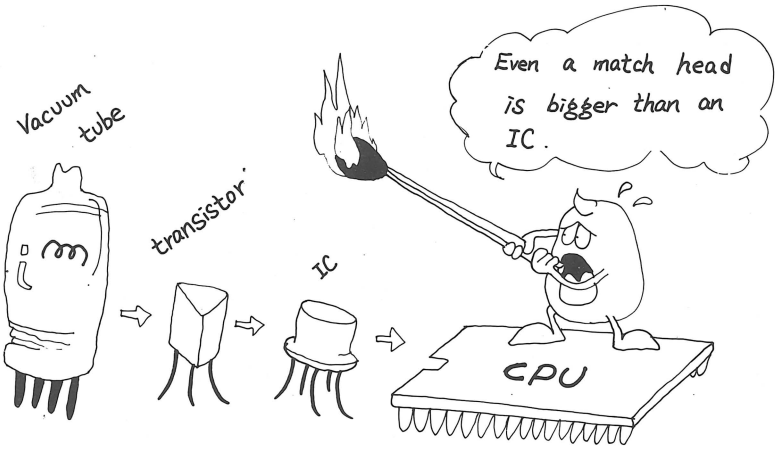



Aided by advancement in electronic technology, electronic elements developed from vacuum tubes to transistors, and then to integrated circuits (IC). As a result, computer science also advanced.

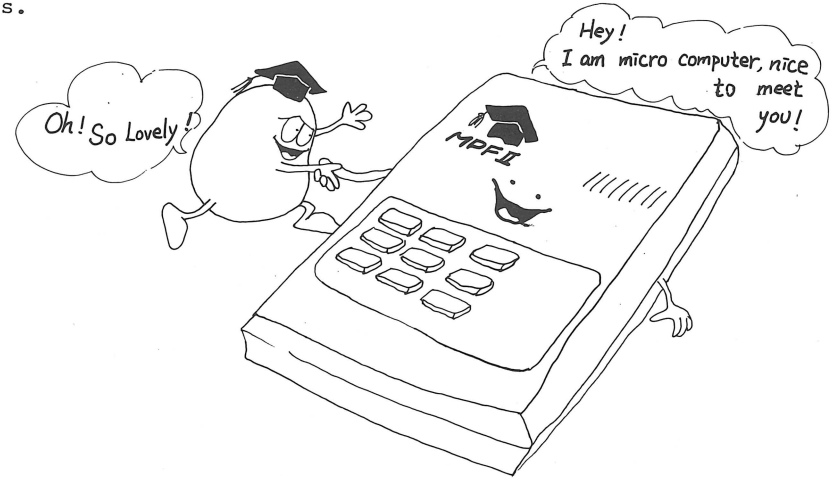
In 1950's, the second generation computers came to the world. They were equipped with transistors, each one able to replace a vacuum tube, but the size of transisitor is much smaller.

In 1960's, the third generation computers came into existance. They used integrated circuits (ICs). The size of each integrated circuit is about the size of a thumb. Each IC equals tens up to hundreds of transistors. Thus, the size of computer decreases, the speed increases, and the consumption of electricity lessens.

In 1970's, IBM company produced the IBM 370 computer which can only be regarded as a three-and-a-half generation computer, but not the fourth generation. The fourth generation refers to the microprocessor-centered microcomputers. Microprocessor, as are large as a thumb, consists of thousands of transistors (or more). Consequently, the size of microcomputer decreases, the consumption of electricity lessens, the power increases and the computer costs less.




Since the microprocessor came to the world, it decreases immensely the size and cost of computer, quickens the introduction and application of computer to the daily lives, and enlarges the opportunities of learning about computers.




A computer is an essential device for processing information, and we usually associate the term "information" with "computer". Although the term computer does not necessarily mean information, after a computer processes data, we can then comprehend the meaning of the information.

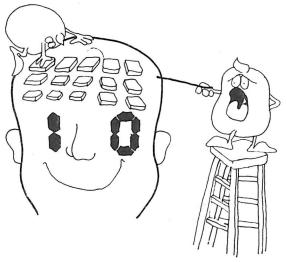
## 1 • 2 Human Brain Vs Electronic Brain

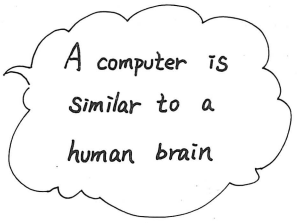
Now, we would like to explain why computer is also referred to as "electronic brain".

In addition to computation, we are frequently required to judge, to analyze and to compare in our routine lives. When we are performing these tasks, we need lots of data as input. In order to acquire this data, we have to collect, compute and classify it first. Yet, in all these tasks, the human brain plays the control part.

As a matter of fact, the computer can perform not only computing, but also judging, comparing and analyzing. And because it can help our brains work by being an "artificial brain", we may call it "electronic brain" as well.

Now that "electronic brain" is a machine to help human brain work, so an understanding of the workings/of a human brain can be the basis of the understanding of computer.

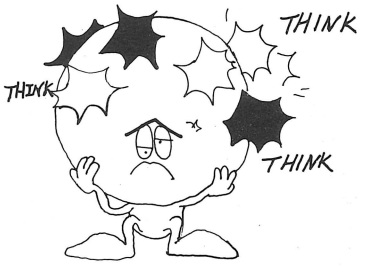





Mankind is the master of all the creatures in the world, because mankind knows how to think. From the ancient times, human beings have been thinking how to improve their lives, this has resulted in various efforts.

Mankind knows how to think, and computer is the mechanical device to aid mankind's thinking.

What makes mankind different is our power to think.



We have known that the brain power enables human beings to invent tools and machinery and thus makes human beings more versatile. Because the computer was invented, human beings have accelerated the pace of life.

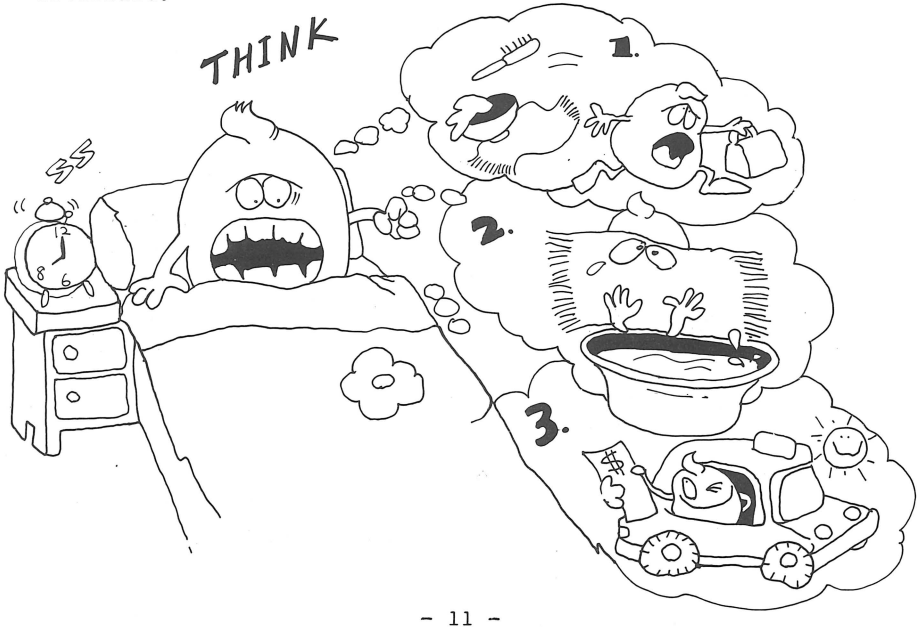
Since computers were invented to help human brains, once we understand the thinking process of human brains, it will be easier for us to understand the theory of computers.

Since the computer is the machine which helps human beings think, we must discuss further the detailed process of how men think. First, let's review how we think:

For example, if one day you overslept, when you woke up, your first reaction must be: "Augh, it is too late for the school!" Suppose you did everything step by step as usual, you would certainly be late. Therefore, you would probably give up brushing your teeth, washing your face, or even your breakfast, and go out in a hurry.

Perhaps, after thinking, you realized that there was still time to wash face. You would probably wash your face first and then go out, although you didn't have time for breakfast.

Perhaps, after thinking, you thought you could afford to take a taxi to reach school on time, thus to save time for you to wash your face, to brush your teeth and to have breakfast.



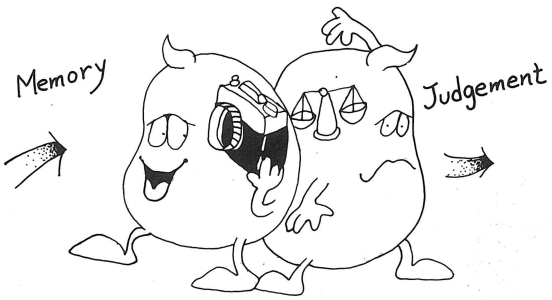
Facing the above situation, you should first think about "Can I go to school on time?", and then come to a certain conclusion, for example, "I will skip everything.", "I will still wash my face", or "I'll go by taxi", etc. But, how are these conclusions reached? Of course, it is not without resources. These conclusions arrived by thinking would probably be the product of using the "memory" area of your brain.

You were late for school, because one day you overslept, but you still had your breakfast and washed your face as usual. Or perhaps, you yourself didn't have such an experience, but you did see that your classmate was reprimanded for being late, so this fact was rooted in your memory. Of course, in order to arrive at a similar conclusion mentioned above, you not only have to have a memory as a source, but also apply judgement. For example, you overslept last time and went to school without washing your face and brushing your teeth, and you were stared at by your classmates. This time, in order not to be involved in the same situation, you choose to go by taxi. So, we know that these conditions must be judged each time according to your memory patterns.

From the above, we realize that human thinking is composed of "memory" and "judgement". Our various daily thinking activities call all be regarded as the judgements made according to our memory. Therefore, we have to study "memory" and "judgement" in order to research the human thinking process.

*To research human thinking, we have to*

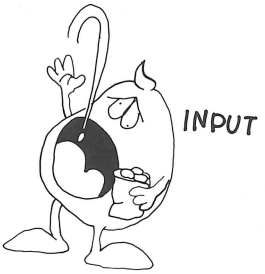
*Study "memory" and "judgement"*



"Memory" is produced by experience. And so-called experience can be regarded as the external things gained by our sense organs.

The sense organs refer to the five senses, the eyes, ears, nose, tongue and skin. We see with eyes, hear with ears, smell with nose, taste with tongue, and feel with skin. Even thougn human beings have intelligent brains, if without the five senses to feel the external things, there is no way to accumulate experience. That is, without the five senses, human beings can't receive the external stimula.

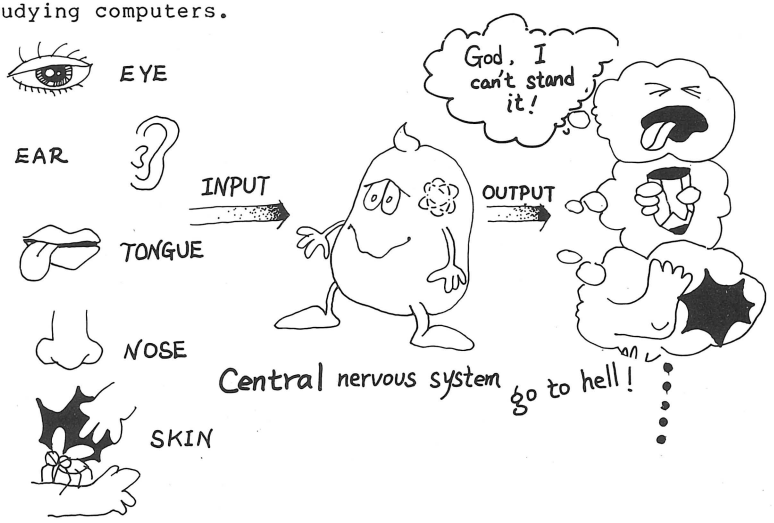
The so-called external stimula here means input. All the stimula received by five senses are referred to as input.




Our thinking capability starts from input. The five senses first receive some external stimula and then transfer them to the central nervous system where the stimula are processed by judging and memorizing to reach a conclusion, finally this conclusion is expressed by means of mouth, face, hands, feet and body. The reaction of such expression is output.

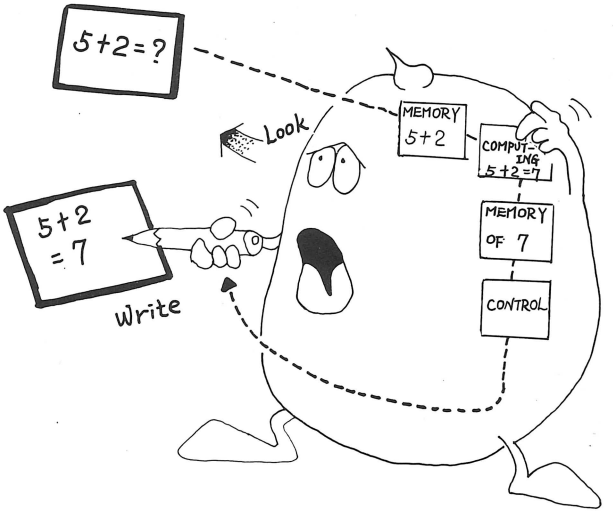


In order to comprehend human behavior, please look at the following picture. This picture is the first step in studying computers.



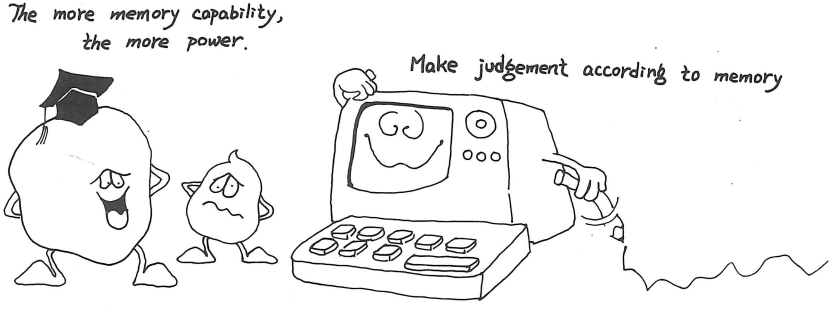

We shall first focus on "memory". When we are thinking we have to revive past input from memory and then apply it.

Although the above thinking process is interesting, it is still a little difficult to apply this to explain a computer. Let's now have another simple example:



As shown in the picture, when a pupil is performing a mathematical addition: he first sees the computing pattern 5 + 2 in his memory, therefore he loads this pattern into the processing area of his brain; and then the pattern 5 + 2 = 7 comes into his mind, therefore he concludes that the answer for this computation is 7. Then, the central nervous system commands the hands to produce the answer.

From above example, we see that computer has also an input section. This input section transfers messages into the core of computer which has already had prior information memory as means of judgement, and then it transfers a result to the outer world through an output section.

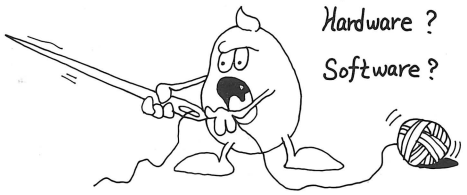



We have mentioned over and over that our daily thinking activities are altered by what is in our "memory". Therefore, good thinking must have an abundant memory as its source. Certainly, a man with a large memory cannot necessarily make a good judgement; but we can be assure that, without "memory" as source of judgement, there must be lots of mistakes. These mistakes are what we usually call mere conjecture.

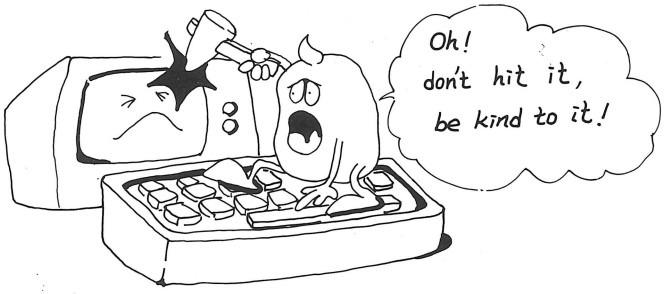
In fact, the capacity of computer is determined by the amount of its memory, thus the capacity of a computer is in varies with the size of its memory. The more larger memory a computer has, the more capacity it can afford.

## 1 • 3 Hardware

There are two important terms in computer terminology: hardware and software.

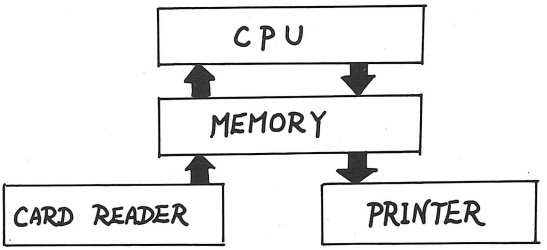



Computer hardware refers to the parts which can be seen and touched, and is composed of electronic circuits.



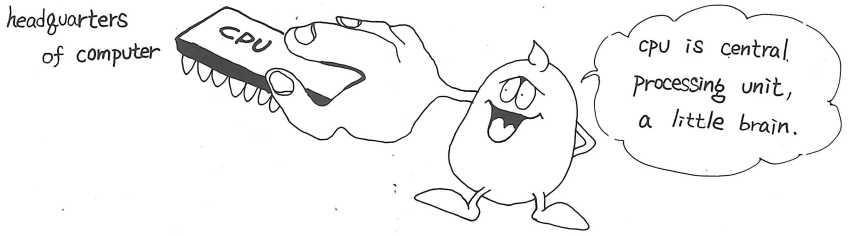

In last section we have learned that human activities involve memorizing and judging, and that memory is an essential part in computer. The Central Processing Unit (CPU) is responsible for the tasks of controlling and judging in computer. Just as human beings do, a computer needs input and output too. Card reader has been the main device for input. Printers are still one of the main devices for output.

Now we will explain the three hardware parts shown below individually:



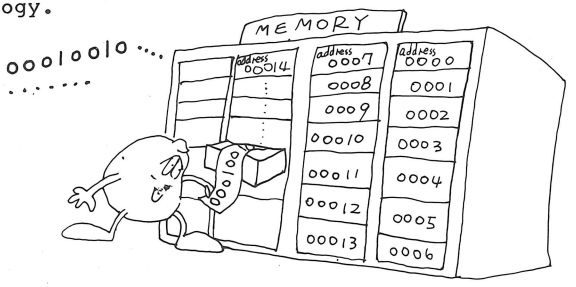
**1. Central processing unit (CPU)**

The CPU is the headquarters of a computer, it deals with all kinds of computing and processing. The letters CPU is the acronym for Central Processing Unit.



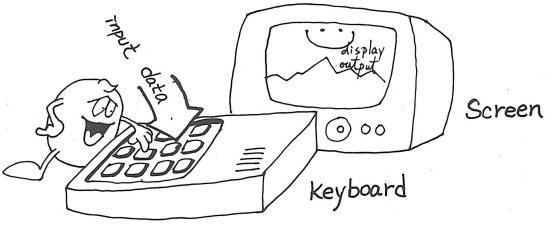

**2. Memory**

The existance of a memory is the reason why a computer possesses the capability of memorizing recording information. All the data waiting to be processed exists in this area. The arrangement of a memory is just like the picture shown below, which has each location numbered. These numbered locations are called addresses in computer terminology.

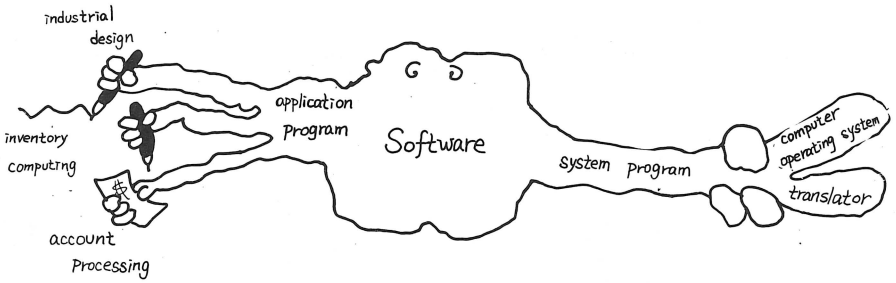



**3. Input and output**

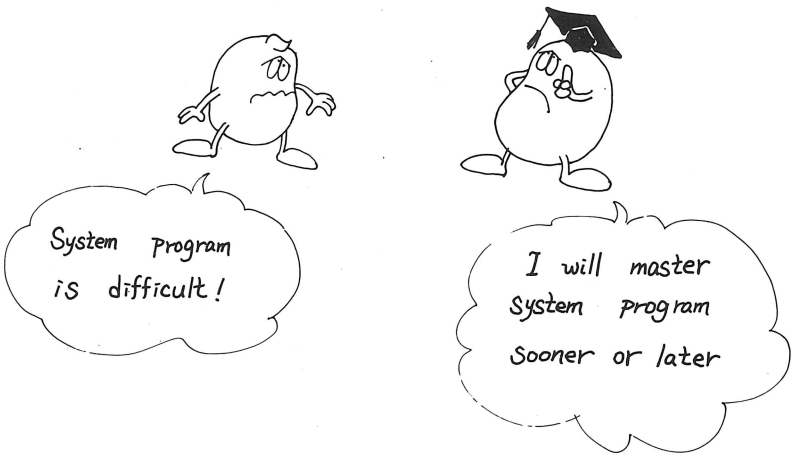
The input and output are the media by which the computer communicates with the outer world. The Card reader with reads in the data is the input device. The printer which recordes on paper the results is the output device. In addition, a keyboard is a very important input device, and screen an important output device.



The main function of a system program is to make it easier for you to use the computer. Two classes of system programs that are particularly usefull are: translators and computer operating systems.




A translator helps us to write application program, which on operating system helps us operate a computer. These programs are provided by computer manufacturers and specialists, thus we only have to comprehend the functions of these programs.

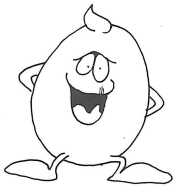


## 1 • 4 Computer Language

In using the computer to solve problems, we must use a computer language to communicate with computer.

A computer language can be called a machine language to distinguish it from natural languages such as English, Chinese, Japanese, etc.

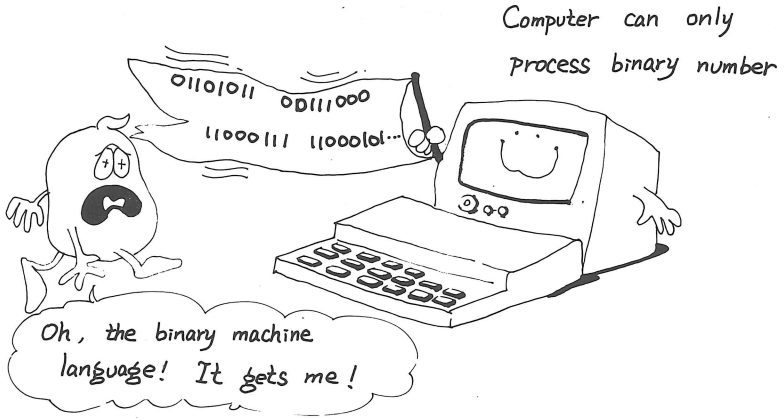
Software refers to the program which operates the computer. In writing programs, we have to use a machine language acceptable by computer. Also, data have to be loaded into the computer in a way which is understandable by the computer.



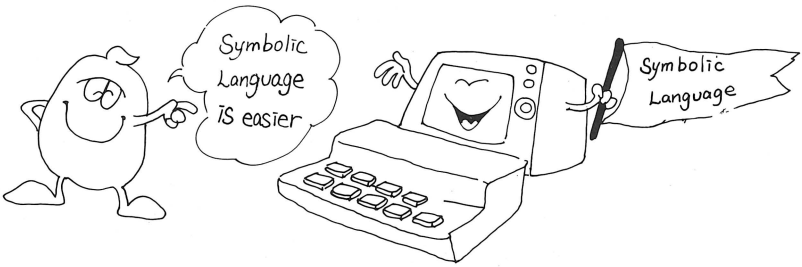
Computer language is to communicate with computer.

A modern computer understands only two digits: 0 and 1. This simple number system (binary) uses binary numbers. That is, a modern computer understands and processes only binary numbers.

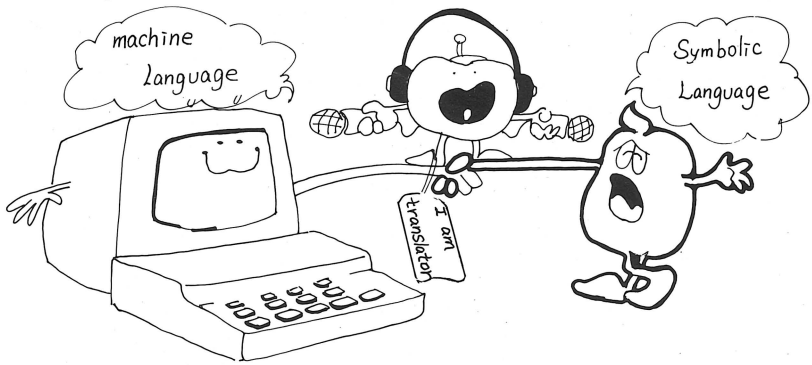
Obviously, it is impractical to communicate with computer in binary machine language. It is also rather difficult and time-consuming for programmers to write programs in binary language.



The first step in simplifying programming is to use more easily understandable symbols to represent the binary language which is meant to communicate with computer. As a result, symbolic language which is similar to English came into existance.

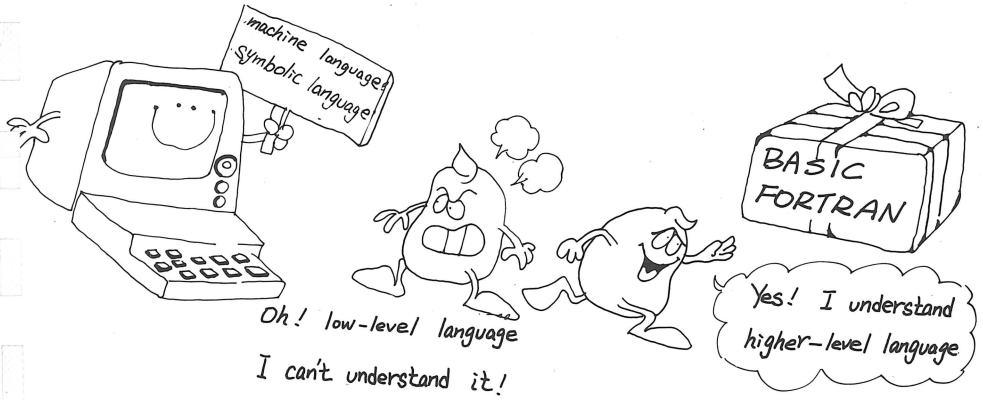



A symbolic language makes it easier for us to write programs, but we have to add a translator program in the computer. Translators are provided by manufacturers and computer specialists. A translator makes it easier for us to communicate with computer. In a computer system, a symbolic language is translated by translator into a machine language understandable to computer. Therefore, in this way computer is able to work for us.




A symbolic language is still quite different from the way we express ourselves, even though symbolic language makes it easier to use a computer, thus, some languages more similar to English have been produced.

FORTRAN and BASIC are the examples of these language. Machine language and symbolic language which bear a resemblance to computer language are referred to as low-level language. While FORTRAN and BASIC which bear a resemblance to human language are referred to as higher-level language.




The main purpose of this manual is to instruct you how to use the computer, therefore, we will use a higher-level language to communicate with computer. How a higher-level language is translated into machine language will not be discussed here.

Now, let us discuss some advantages of applying a higher-level language:

1. The computation pattern used by higher-level language is rather similar to the computation pattern used in mathematics, so it is easy for us to learn higher-level language.

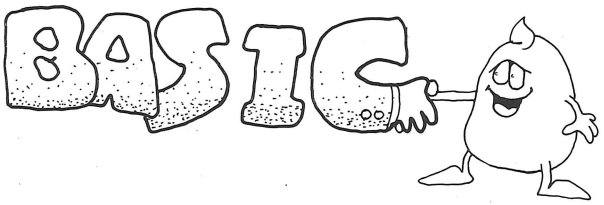
2. The user does not need to know the actual memory locations used by the program in the computer. We only need to know the rules of a higher-level language, and use some variables as labels, then computer itself will arrange it for us.

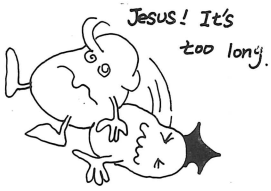
3. A Program is composed of some English sentences which are simpler than actual English sentences, so it is easier for us to use higher-level language.

Of course, there are many other higher-level languages, we can surely learn more as time and interest permit. But in reality, if we master one, it is quite simple for us to learn another.

This manual uses the easy and simple BASIC language to instruct the beginners. BASIC is the acronym of Beginner's All-purpose Symbolic Instruction Code.

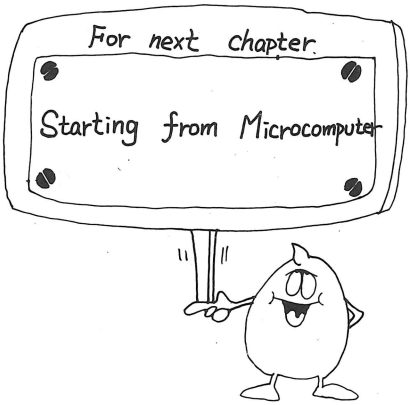
BEGINNER'S ALL-PURPOSE SYMBOLIC INSTRUCTION CODE




The advantages of BASIC are the following:

1. Easy to learn and with powerful functions.
2. Most inexpensive and popular microcomputers use BASIC language, thus your knowledge of BASIC can we used just about everywhere.



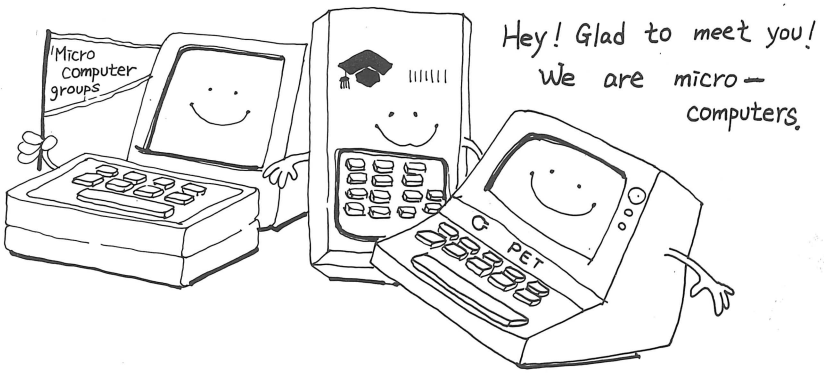
# 2

# STARTING FROM MICRO-COMPUTER

## 2・1 The Learning of Microcomputer

Before microcomputers came into existance, computers were quite expensive. Even if I were presented with a computer as a gift. I would be troubled as to whether I had enough room in my study to for it. But this condition changed greatly since microcomputers came into the world. Nowadays the situation becomes that almost everybody can buy a set of a microcomputer to use at home. And the youngsters are even more interested in assembling sets of microcomputer by themselves. If you have the chance to drive around, stores selling microcomputers are everywhere. You surely can sense that the era of microcomputer is on its way.

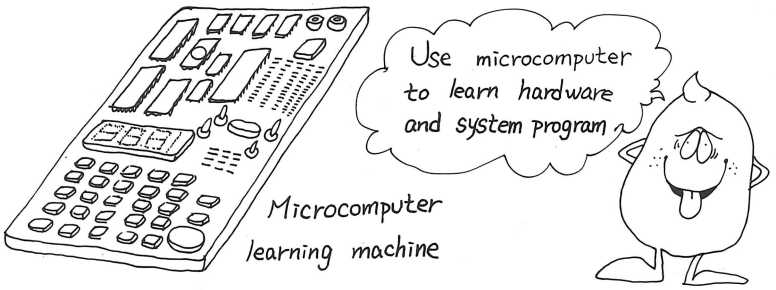
This is a good chance for those who are interested in computing to learn with the microcomputer as an learning aid instead of starting with a mini-computer at school.




As for the learning of computer, there are two different ways that should be noted:

1. One of them is to use a home computer or personal computer as the learning aids. Those microcomputer that are available worldwide including MPF-II, PET, TRS-80, APPLE-II etc.

2. Another one is the using of microcomputer learning machine as the learning aids, MPF-I etc.




This book emphasizes the learning computing using a home computer and personal computer. And the main emphasis is put on the learning of application programs by using the BASIC as the higher-level language. This is an easy approach for everybody to learn computing.

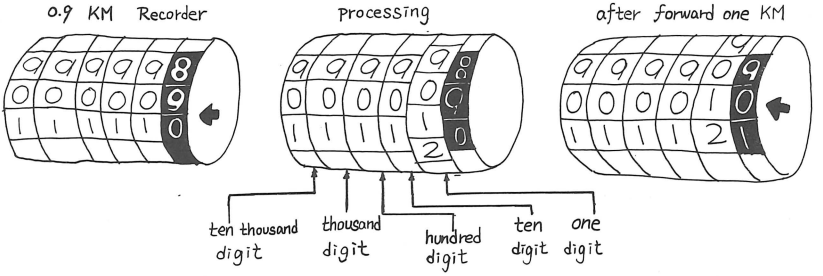
The Microcomputer learning machine lays more stress on the hardware and system programs by using a low-level language, which is closely connected with machines, such as symbol language and machine language. For those who want to go more deeply into hardware - how a computer works - a microcomputer learning machine such as the MPF-I can help you reach your goal.

This manual introduces you briefly to microcomputer application programs by using the BASIC language.

## 2 • 2 Binary Number System

Let's imagine that there is meter whose wheel only consists of two digits, Ø and 1. When turning the wheel, Ø appears first and then 1 and then Ø appears agian. Since the meter has only binary digits, this meter is called a binary meter. In order to get a concrete idea of the binary meter, let's look at a distance meter first (odometer).

The meter shown below consists of six drum gears. When the first gear turns to 9, if you turn a little bit more, it will then turn back to Ø, and at the same time advance the neighboring gear on the left side to 1.




The concept of return-to-zero and advance the higher wheel can be applied to the binary meter. When 1 turns back to Ø, the next higher digit unit place will advance.

The phenomena of the calculating of a binary speed-meter on an automobile is shown below. In a new automobile, all the digits on the binary speedmeter are all Ø.

ØØØØØ

After driving for one mile, the scale will be

ØØØØ1 (1)

After driving for one mile more, the digit on the first unit place will return to Ø and advance the next position. The scale will be

ØØØ1Ø (2)

After driving for three miles, the scale will be

ØØØ11 (3)

After driving for four miles, what will the condition be? The first gear returns to Ø and advances the next digits; the second gear returns to Ø and advances the next digit; the third digit is 1. It will be

ØØ1ØØ (4)

After driving for more miles, the scale will be

ØØ1Ø1 (5)
ØØ11Ø (6)
ØØ111 (7)

After driving for eight miles, the first gear returns to Ø and advances the next digit. The same with the second and the third one. And the fourth gear becomes 1. The result will be

Ø1ØØØ (8)

After nine miles, it will be

Ø1ØØ1 (9)

After ten, it will be

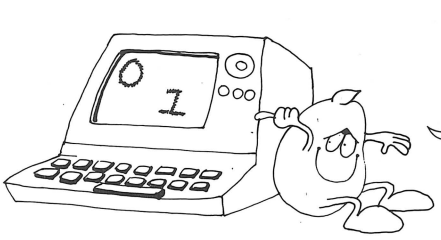
Ø1Ø1Ø (1Ø)

(Do the following by yourself.)

Now you form the concept that each mile adds 1 to the first gear. After revolving for one circle, the digit will return to Ø and advance the next gear (digit). Similarly, if the second gear revolves a circle, it will return to Ø and advance the next gear, and so will the others.

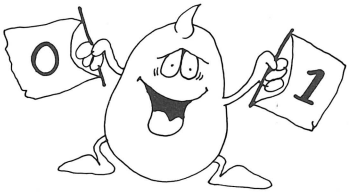
A binary speedmeter can show the binary number consisting of a set of digits, 1 and Ø. ØØØØ1 represents 1; ØØØ1Ø represents 2; ØØØ11 represents 3 and so forth.

When dealing with great numbers, it is inconvenient to use binary numbers. For example, Ø1ØØØ represents 8, Ø1ØØ1 represents 9; Ø1Ø1Ø represents 1Ø. Therefore when it is not really necessary, we will not use a binary number. But when analyzing the performance of computer, it is frequently necessary to use binary numbers. Why? Just as with the limitations of a binary speedmeter, a computer circuit can only handle binary numbers.




The last thing to mention is that when the display of the decimal speedmeter is 00314, the two zeroes can be omitted and read as 314. Similarly, when the display of a binary meter is 00111, zeroes can also be omitted and read as 111. Omitting the zero, binary counting will be 0, 1, 10, 11, 100, 101, .....

From what is mentioned above, we know that any numbers that are used in the binary system can be expressed only by a combination of zeroes and ones.

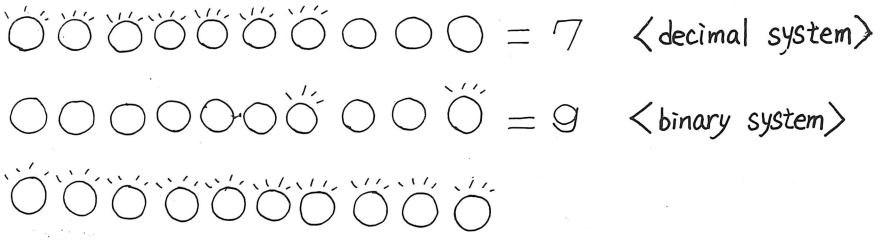



As a matter of fact, it is frequently more convenient to use the binary system instead of decimal system. We can see from the following example:

Suppose there are ten bulbs put side by side and we hope to make use of the ons and offs to represent a certain number.

1. When, in the decimal system, 10 lighted bulbs represent a certain number 10; 6 bulbs represent 6; 3 bulbs represent 3. This means that only eleven numbers, from 0 to 10, can be shown.

2. When we apply the binary system, 1 means lighting up and 0 means turning off. When all the bulbs are lighted up, it will represent 1111111111 (binary) which means number 1023 (decimal) after some calculation. There are 1024 numbers from zero to 1023. Number 9 in the decimal system can be represented as 0000001001 in a binary system.




512+256+128+64+32+16+8+4+2+1=1023 <binary system>

So, you can make a conclusion:

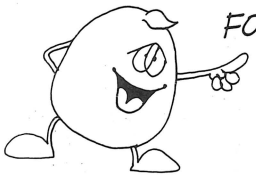
A bit can represent 0 and 1, only two values in.
Two bits can represent 4 values, that is, $2^2 = 4$
Three bits can represent 8 values, that is, $2^3 = 8$
Four bits can represent 16 values, that is, $2^4 = 16$

.
.
.
.
.
.

8 bits can represent 256 values, that is, $2^8 = 256$
16 bits can represent 65536 values, that is, $2^{16} = 65536$

From above example, we know that the CPU of a 6502 has only the data of 8-bits which can only show 256 different numbers. It means that only the numbers from 0 to 255 can be shown.

The 6502 uses 16-bit addresses so that there are 65536 digits from 0 to 65535 which can be used as addresses.



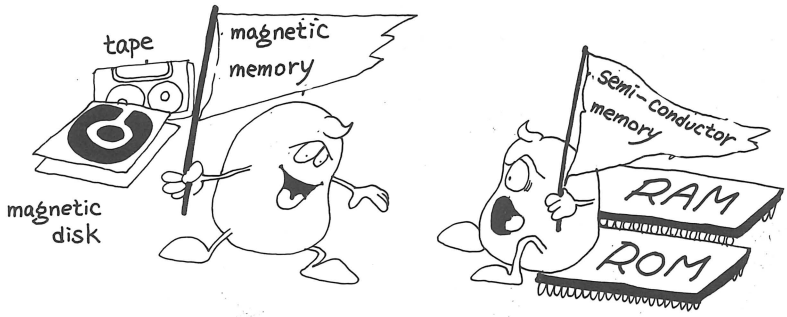
FOR CPU 6502
Data : 0~255 ($2^8$)
Address : 0~65535 ($2^{16}$)

## 2 • 3 Memory

There are two kinds of memories that are used in a computer:

a. Magnetic memory: it includes magnetic tapes and magnetic disks. They store data using magnetic recording theory.

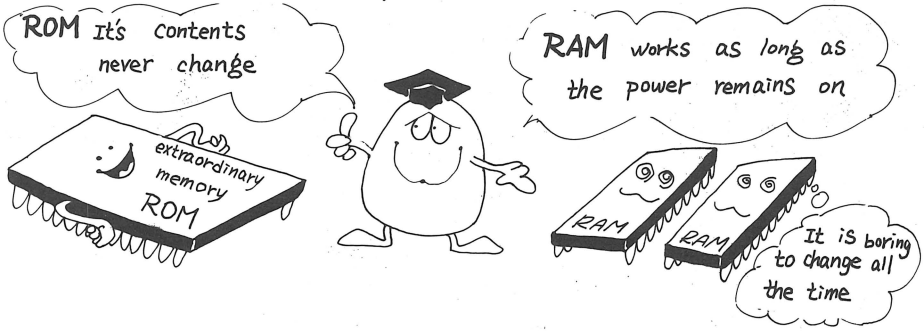
b. Semi-conductor memory: data is stored using solid state characteristics (semi-conductor theory).



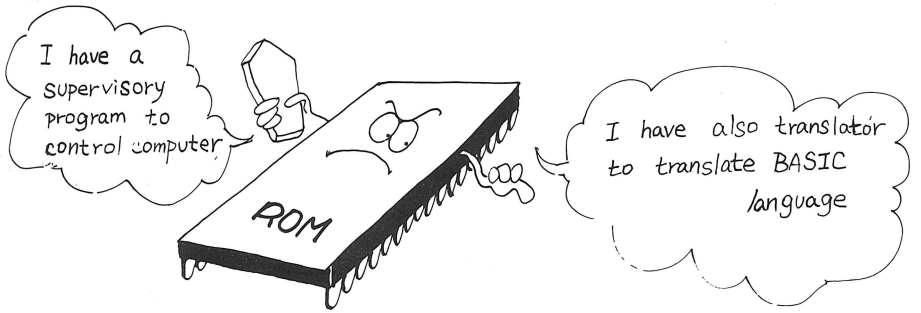

Owing to the progress that the semi-conductor technology has made, the semi-conductor memories have become the main memory in a computer while the magnetic tape and magnetic disk function as the peripheral memories. In this section, only semi-conductor memory will be discussed.

There are two kinds of semi-conductor memories: ROM and RAM. ROM is the acronym for Read Only Memory. The data or commands stored in it will not disappear when power is removed, but ROM can not be altered (corrected).

RAM is the acronym for Random Access Memory. The data and commands in a RAM can be corrected at random, but when the machine is switched off, the RAM contents are lost.

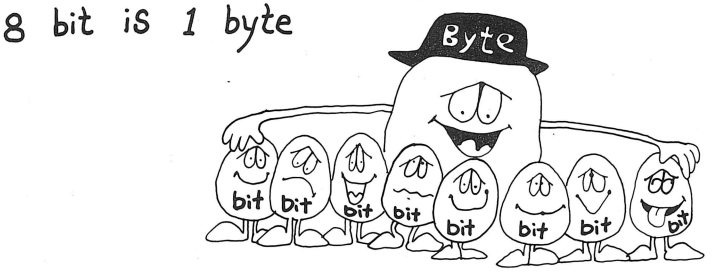


When the switch is on, the computer is ready for you to give it commands. The computer already contains a supervisory program and a translator program in ROM. Turn off the computer and these programs are still there. Because the translating program is always there you can program in BASIC at any time.

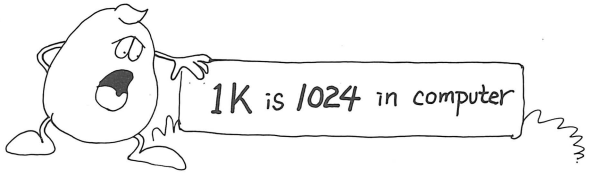



RAM is the memory that can be read and written at random. It is available to the user. The user puts the written program the input data, into the RAM. The larger the RAM memory, the more memory space available to the user.

We have used the binary number system and the smallest unit in the system is a bit. Because the bit unit is too small, we are in need of a larger unit. You may know that most CPUs can process 8 bits at a time, so 8 bits are combined into a larger unit. This unit is called a byte.



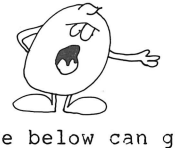
But byte is still too small to describe the capacity of a computer memory. We need a much larger unit. In decimal number system, we use thousand as multiple and represent it by K. In a computer, we can also use K. The K of a computer memory does not represent 1000, but 1024. Because $2^{10}$ = 1024 and it is close to 1000, we can use it as unit.




1K BYTES = 1 x 1024 BYTES = 1024 BYTES
4K BYTES = 4 x 1024 BYTES = 4096 BYTES
8K BYTES = 8 x 1024 BYTES = 8192 BYTES
16K BYTES = 16 x 1024 BYTES = 16384 BYTES

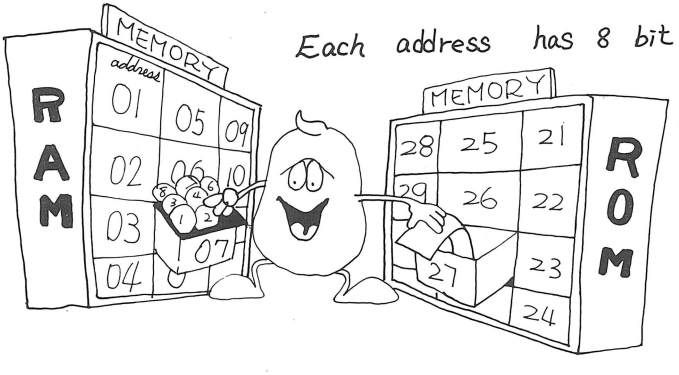
Now we know that a computer with 64K byte memory can store 1024 x 64 = 65536 bytes.

If you want to buy a microcomputer, you have to pay heed to the amount of ROM and RAM, and you should know that the programs put in a ROM are either supervisory programs or translating programs; RAM is availible for you to use.



MPF-II-ROM : 16K Bytes
RAM: 16K Bytes
64K Bytes

The picture below can give you a synthesis explanation: Memory can be divided into ROM and RAM. Those put in the ROM are supervisory programs and translating programs; RAM is the area for you to use. Each address in ROM or RAM is a byte consisting of 8 bits.



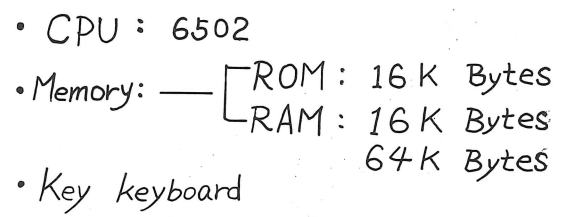
For general family computing, 16K RAM is sufficient. But in consideration of the needs of future expansion, a model with 64K RAM is provided. Users can choose the model that best suits a user's needs.

Up to now, let's have a review:
The main board in MPF-II consists of three parts:

. CPU: 6502

. Memory: ROM 16K bytes
RAM 16K bytes

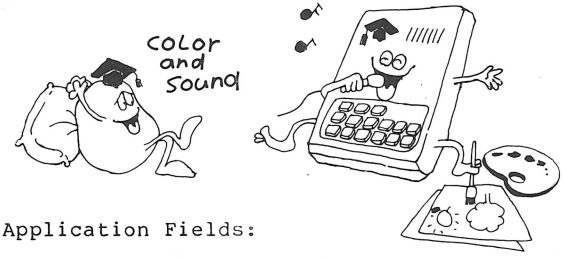
. Keyboard



# 3

# PRESENTING THE MPF-II

MPF-II (Micro-Professor II) is a low-cost R6502 based microcomputer system. Its built-in BASIC language makes it easy to create user programs. The color and sound capabilities help keep the user interested in programming.




MPF-II Application Fields:

1. Learning computer languages such as BASIC, PASCAL, and FORTH.

2. Games

3. Education: such as spelling, music, Chinese, mathematics, bridge,...etc.

4. Home and business management.

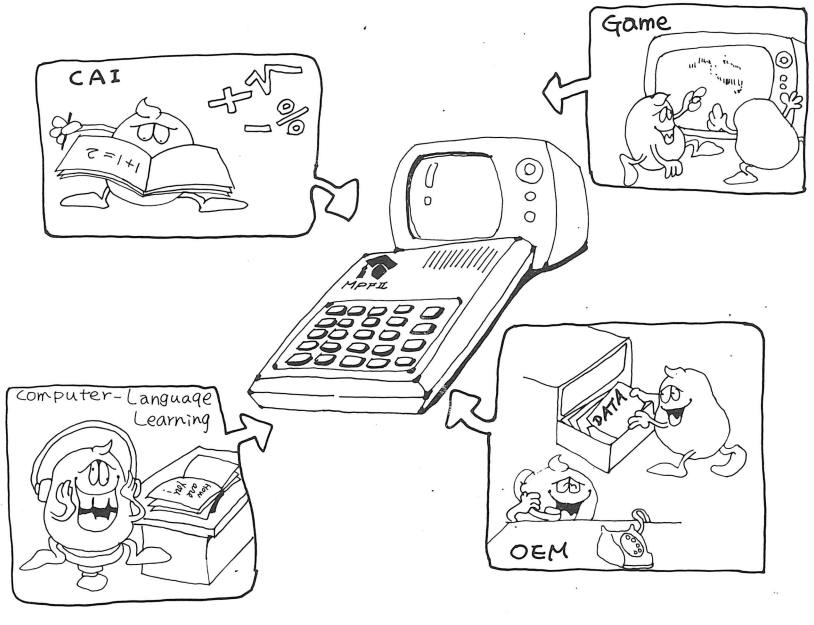


Figure 3-1 is a figure of a typical MPF-II computer system. Notice that it takes several separate pieces of equipment to make up a complete system.

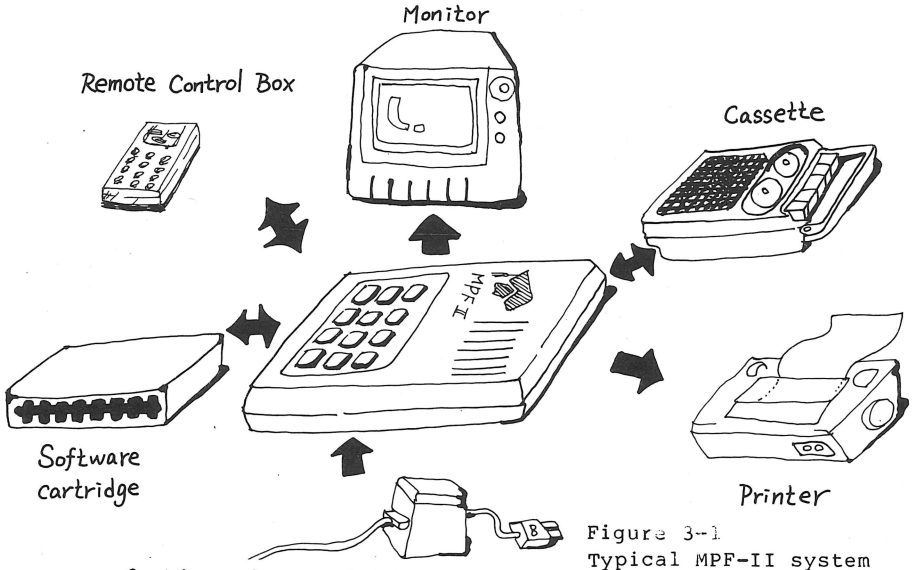
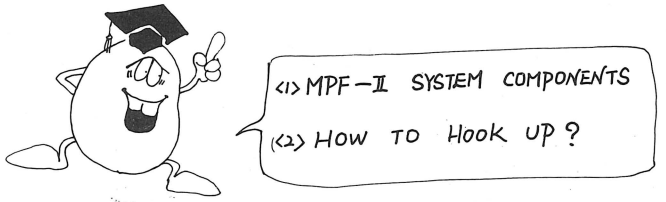



Figure 3-1
Typical MPF-II system

Your system may not look exactly like the one pictured in Figure 3-1. Many system components come from a list of optional equipment. But there are three components that every system has in common:

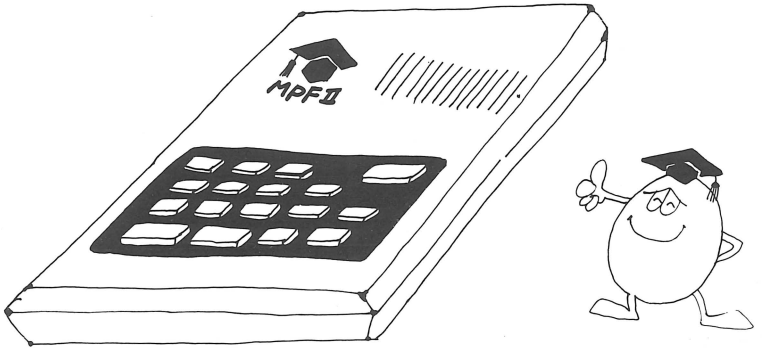
MPF-II itself, the built-in keyboard, and a television. Let's take a closer look at each of these and at some of the more common pieces of optional equipment. We will also describe how to hook up these components to the MPF-II.




## 3 • 1 MPF-II Main Board

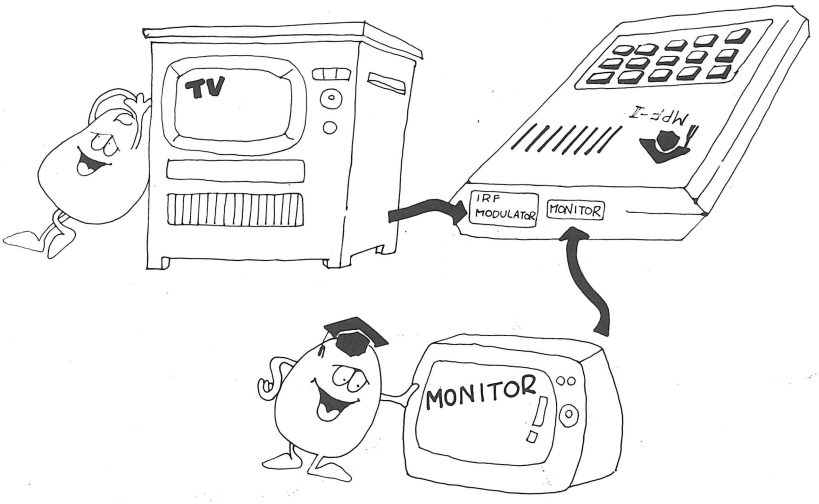
The MPF-II Main Board provides the major functions of the system and provides interfaces to the peripherals.

The keyboard and TV screen allow the user to communicate with the MPF-II. A standard typewriter-style keyboard comes with the MPF-II. It transfers instructions from your fingertips into the MPF-II.

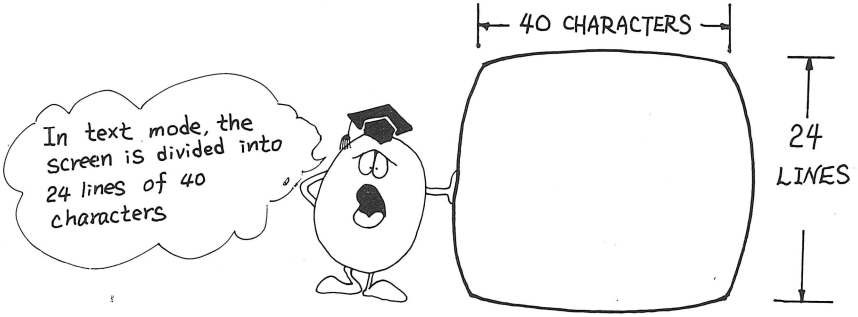



## 3 • 2 Video Display

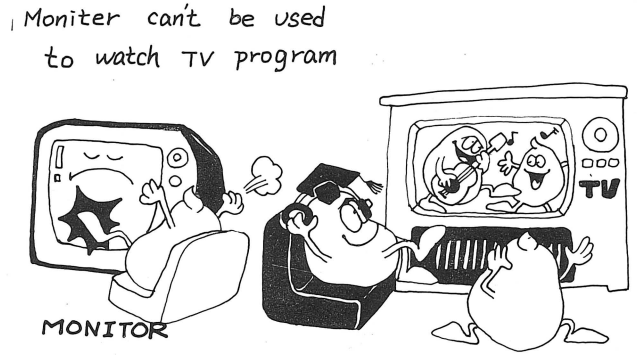
The display screen can either be an ordinary color television set or a color television monitor. A black-and-white TV works fine too, but of course color displays will show up in black-and-white. The screen not only echoes everything you type so that you can visually verify its accuracy, it also displays the reactions of the MPF-II to your instructions.



The standard display screen has three different modes of operation. One is for black-and-white text characters only and the other two are chiefly for graphics. In the text mode, the standard screen is divided into 24 lines of 40 characters each. The graphics modes deal with points and lines, not characters, and subdivide the screen more finely (graphics are discussed further in Chapter 8.)




An MPF-II user may use a television set for his display screen either because he has one or because it's a good excuse to get one. The televison monitor produces a sharper picture than a TV set in the computer environment, but you can't use it to watch Sea Hunt, Highway Patrol, or the 6 o'clock News.

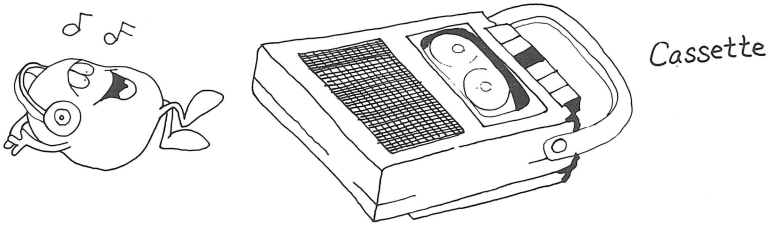



## 3 • 3 Power Supply

The power supply provides the power used in MPF-II, external memory, and I/O expansion.

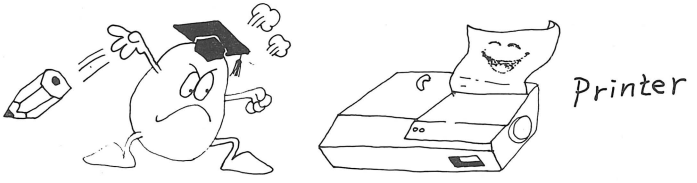
## 3 • 4 Cassette

A cassette provides the auxiliary storage for saving information. The MPF-II can save program on audio cassette tape and read it back again.




## 3 • 5 Printer

A printer with a Centronics interface may be connected to the MPF-II. The printer provides the capability of listing programs, memory contents and makes hard copy of the video display.

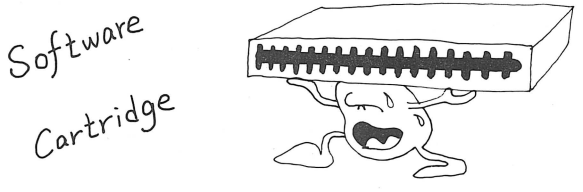



## 3 • 6 Remote Control Box

Two Remote Control Boxes may be connected to the MPF-II for game applications.

## 3 • 7 Software Cartridge

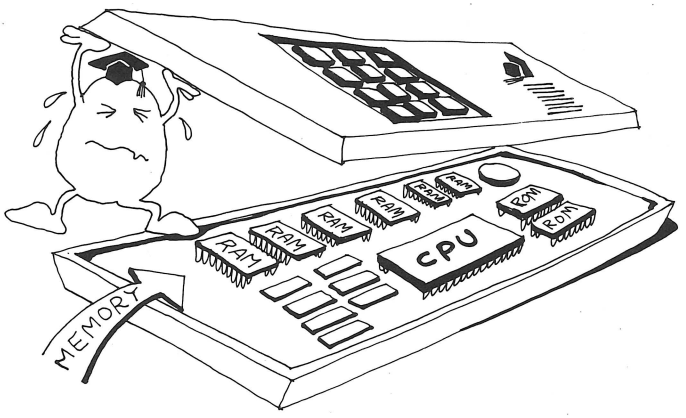
The cartridge provides the capacity for the user to enter special programs in BASIC or ASSEMBLY language to the MPF-II. Such as Games, Assembler,....etc.



## 3 • 8 Inside the MPF-II

The MPF-II itself houses the part of the computer that controls the rest of the system -- under your guidance, of course! Lurking behind the keyboard are the main MPF-II memory banks, the microprocessor, the connection points for all the accessory components, and much more. Figure 1-7 discloses the true identity of these undercover items.

The inside of your MPF-II may look a bit different from the one in Figure 1-7.




inside the MPF-II

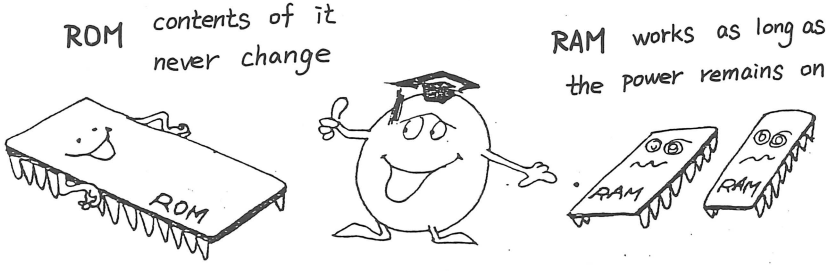
## 3 • 9 Memory

Computer memory size is typically measured in units called bytes. Each byte of memory can hold one character. Depending on the number of chips, your MPF-II computer has anywhere from 16,384 to 65,536 bytes of memory. This is usually stated 16K to 64K, where K represents 1,Ø24 bytes. The amount of memory available determines how much the MPF-II can do, as we will see later.

1K BYTES = 1024 BYTES

The MPF-II actually has two kinds of memory. One is called read-only memory (ROM); its contents never change, even when you turn the power off. ROM contains the programs which give the MPF-II its unique identity and enable it to understand and respond appropriately to the commands you type in at the keyboard. The other kind of memory is calld read/write memory (also called random-access memory or RAM); its contents do change. In fact, the program in read/write memory determines what task the MPF-II is currently executing.

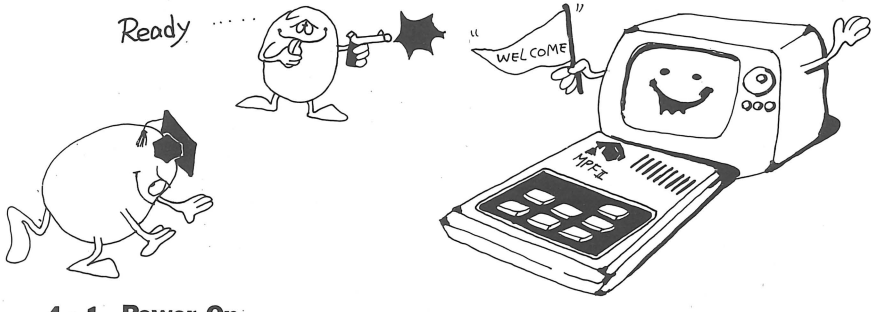
Read/write memory works only as long as the power remains on. As soon as you turn the MPF-II off, everything disappears from read/write memory.



# 4

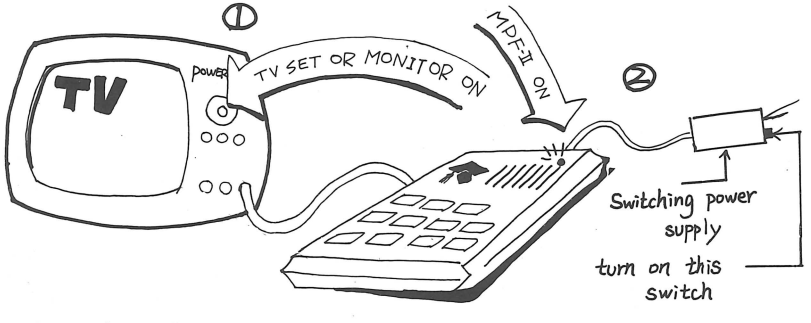
# HOW TO OPERATE THE MPF-II

Any computer system can be a bit intimidating when you first sit down in front of it, even if it's all hooked up, as your MPF-II system must be before you go any further. This chapter will make you feel more comfortable when using the MPF-II by explaining how to use it.

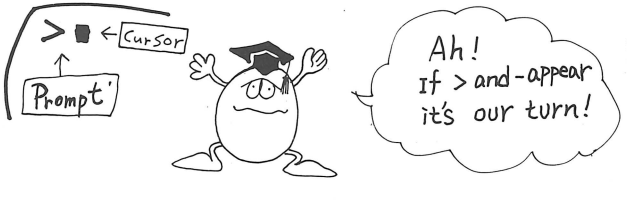



## 4 • 1 Power On

This first thing to do, now that all the connections have been made, is to turn on the MPF-II. First, find the switch on the switching power supply, which is a box about 5.51 inches long, 3.14 inches wide, and 1.49 inches tall (14 x 8 x 3.8 cm). Turn the switch on. You should hear a beep from inside the MPF-II. The beep tells you the MPF-II is ready. The power lamp on the upper right corner of MPF-II main board will be on now.

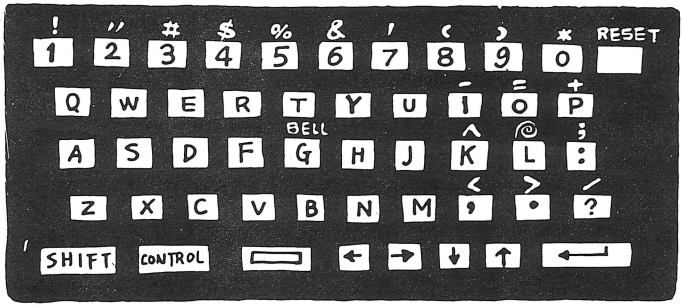



The title "MPF-II" should also appear on the top of the screen along with a ">" and a square " " called the "cursor" to the far left.



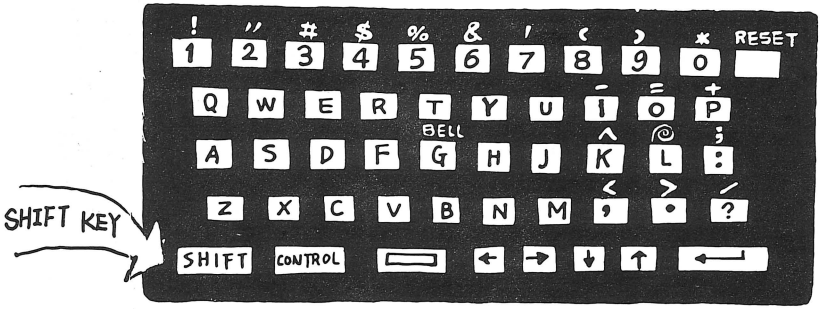
## 4 • 2 Study the Keyboard

If you are familiar with standard typewriters, you will find a few differences between the MPF-II keyboard and a typewriter keyboard. First, there are no lower case letters. You can get only capital letters on the MPF-II. This is all you need for programming in MPF-II BASIC.




**SHIFT-Key**

Using the diagram, locate the SHIFT keys on the keyboard. The reason the keyboard has the SHIFT keys is to allow for nearly twice as many characters with the same number of keys. A keyboard with a separate key for each character would be very large, making it hard to find any desired key.

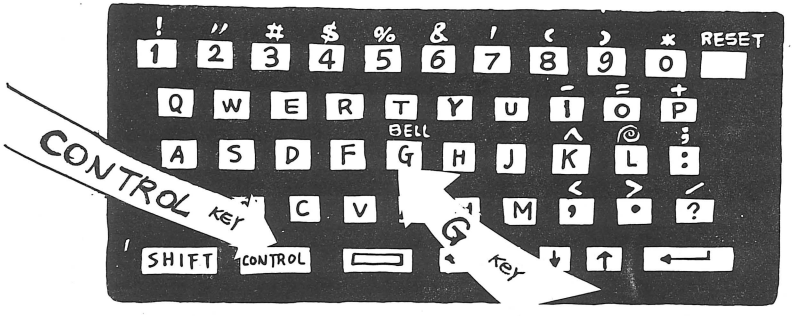



If you only press a key which has two symbols on it, the lower symbol will appear on the screen. If you press the same key while also holding down the SHIFT keys, the upper symbol will appear on the screen. You will find that the SHIFTed comma and the SHIFTed period are "<" and ">" respectively. You will also find other symbols on the MPF-II keyboard that are not on a standard typewriter. Feel free to try operating any of these keys.



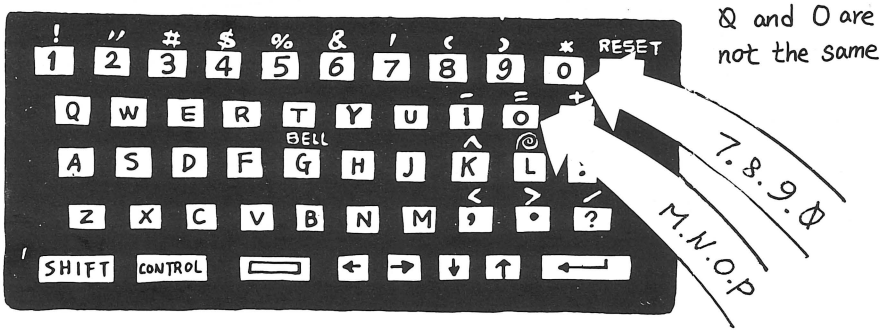

If there is no upper symbol on a key, then holding down the SHIFT while the key is pressed has no effect, except the G.

The G key has the word "BELL" above the "G". But SIFT G does not put a bell on the screen, it just puts a "G" there. The meaning of the word "BELL" on the G key will be explained later.



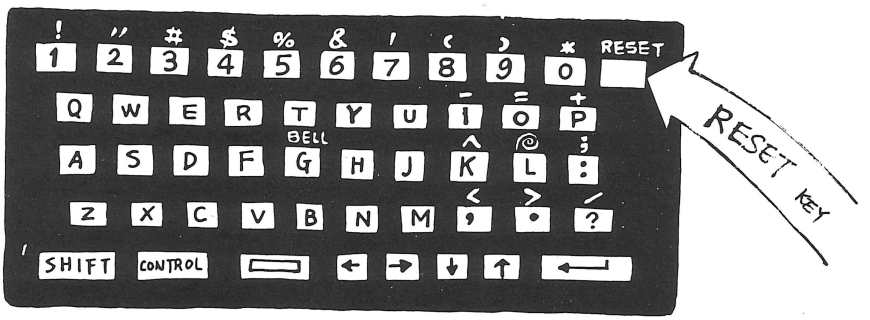

An important difference between using MPF-II keyboard and most typewriters is that you cannot employ a lower case "L" for the number "1". Of course, there is no lower case "L" on the MPF-II, but some typists will have to break the habit of reaching for the letter "L" when they are the number "1".

When the Hindu mathematicians invented the open circle for the numeral zero, they didn't use the Roman alphabet. So they chose a symbol that, while not conflicting with their alphabet, looks just like our letter "Ø". The computer (and any straight-thinging individual) will want to keep zeros and oh's distinct. The usual method for doing this, on the MPF-II and many other computers, is to put a slash through the zero. Now you can tell them apart. The keyboard and the TV display both make the distinction clear. Try them.



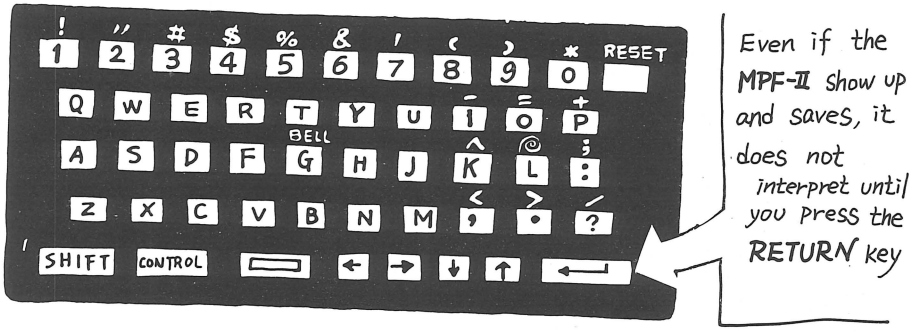

## The Reset Key

Reset is a very special key on the MPF-II keyboard. When Reset is pressed, everything stops. No matter what the MPF-II is doing when Reset is pressed, control of the MPF-II returns to the keyboard. Reset will cause MPF-II prompt to appear.



## The Return Key

As you type along, the characters you type show up on the display screen. In addition, the MPF-II saves everything you type in its memory but does not try to interpret what you type as an instruction until you press the Return key.



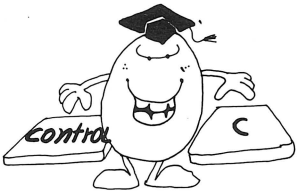

The Return key signals the MPF-II that you have finished the line you have been typing. When you press Return, the MPF-II erases any stray characters that might be to the right of the cursor. Then it examines everything on the line that you just typed in. If those characters make up an instruction that the MPF-II can understand, it will take the appropriate action. Otherwise you will hear a beep and see the message.




The MPF-II is letting you know that it did not understand what you meant by the characters you typed before you pressed Return. You must retype the line (without the error that trippd up the MPF-II).

## The Control Key

The control key is always used together with another key in the same manner as the SHIFT key. You hold the Control key down while you press and release another key.



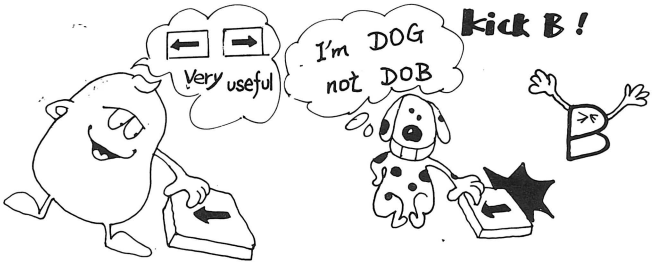

halt program or Listing




cancel line being typed Currently

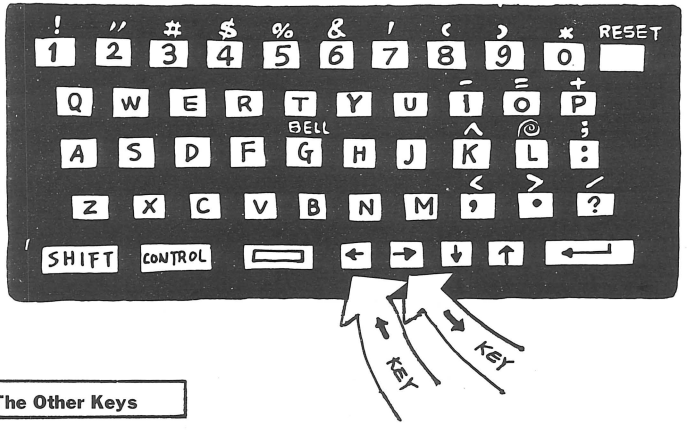
**The ← and → Keys**

The two arrow keys are called left-arrow and right-arrow. You will find the ← and → keys very useful because they allow you to correct any typing mistakes you might make, and allow you to change information you have already entered. The ← key works like the back-space key on a typewriter.




Each time you press it, the character under the cursor is erased from the MPF-II memory and the cursor backs up on space. Try it right now. Type in any word (try PRINT). Press the ← key several times and watch the cursor back up along the word you just typed in. Notice that the characters you back over do not disappear from the display screen. You can rest assured the MFP-II has put them out of its memory. Try backing the cursor all the way to the left edge of the screen. When you get to the edge and press the ← key again, the cursor jumps down one line and a new prompt character appears.

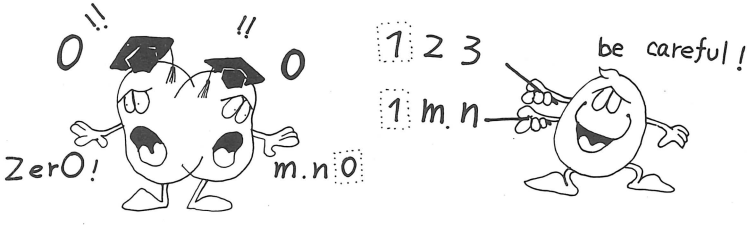
As you might suspect, the ⟶ key moves the cursor to the right along the display line. As the cusor moves forward along the line, every character it passes over gets copied into the MPF-II memory exactly as though you had pressed the key to generate that character. To see the →key in action, type in another word and back the cursor up a few spaces using the →key. Now press the key a few times. Each time you press this key, the character the cursor passes over gets put back into the MPF-II memory exactly the same as if you had retyped it.




## The Other Keys

The other keys on the MPF-II keyboard are no doubt familiar to you. These are the letters of the alphabet, the numbers zero through nine, and a standard set of symbols.

Many typists do not distinguish between the number zero and the letter Ø or the number 1 and the lower-case lette l. The MPF-II can't cope with this ambiguity. You must be very careful to type a numeral when you mean a numeral. To help you remember, the MPF-II keyboard sh the zero with a slash through it, and zeros are displa on the screen with that slash, too.



## 4 • 3 The Cassette Recorder

If your MPF-II system includes a cassette recorder, you can load programs from cassette tapes. Some program tapes come with the MPF-II, you can buy others, and you can make your own as well.

### Handling Cassettes

You should exercise care with the cassettes themselves. They are very easily damaged, and not easily replaced.

Be very careful not to touch the surface of the tape in the cassette. No matter how clean your skin is, natural oils will contaminate the tape. Make sure you put tapes back in their cases when they are not being used. Never store them in hot areas, direct sunlight, or near magnetic fields (like those found near electric motors).

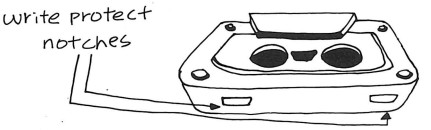
### Label Every Cassette

You should label every cassette with information about the programs it contains. This avoids the headache of searching through cassette after cassette for the program you need.

### Write-Protecting Cassettes

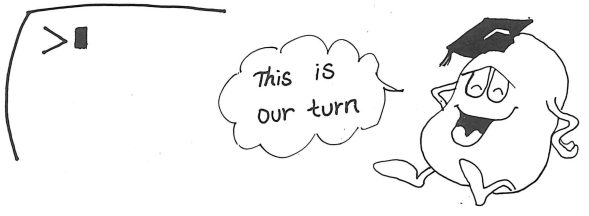
Each cassette has two notches in the rear edge. Most cassette recorders can sense the notches and will not record when they are present. Blank cassettes have tabs covering the notches so the tape may be recorded over. You can protect important programs by removing the correct tab and exposing the notch.

To determine which tab is correct, hold the cassette so that the exposed tape is away from you and the side you wish to protect is facing up. Remove the tab on the right side to prevent recording over the side facing up (see Figure 2-2). Covering a notch with adhesive tape will allow recording over a cassette that has been protected.



**Setting the Tape Recorder**

(If you are not using a cassette recorder, skip this section.) Now press the RETURN key. The prompt and the cursor that show on the screen's left edge to let you know that this is your turn. Now you are ready to set the volume control on the tape recorder.




When you play a tape recorder, it is usually with the intent of making sounds that you can hear. If it is too soft, you miss some of the words or music. If it is too loud, it is annoying.




When you play the tape recorder into the MPF-II, it is with the intent of putting the tape's informatio into the computer. If the volume setting is too soft, the MPF-II will miss some of the information, and it will complain by giving an error message. If the volume setting is too loud, the MPF-II will also complain.

To find the right volume setting, you will use a trial-and-error method. You will play a MPF-II tape softly to the computer and see if the information got in OK. If it doesn't work, you will try the tape again, a little louder this time. If that doesn't work, you will make it a little lounder still. Eventually the volume will be just right for the MPF-II, and it will say so with a beep.

## 4•4 Loading

The MPF-II can be loaded with two types of cassette tapes: 1) tapes of MPF-II format, and 2) some tapes compatible to Apple II format. (Note that not all tapes of Apple II format can be loaded into the MPF-II. For those of you possesses Apple II cassettes, you may use the try-and-error method to find out those tapes of Apple II format which can be loaded into the MPF-II.)

The MPF-II receives filenames of from zero to six alphanumeric characters. But when your cassette only contains one file, it is not necessarily to enter the filename.

Two commands--"LOADT" and "LOADA"--may be used to load your program data into the MPF-II. LOADT is used when loading cassettes of MPF-II format into your MPF-II, while LOADA is entered when loading cassettes of Apple II format into the Micro-Professor II.

One advantage of using the LOADT command when loading a program into the MPF-II is that the video display will show the number of file records and a down counter used to count whether a program has been read completely. If the program is read without error, the display will show OK. Otherwise, the display will show error messages.

Now try the following steps to load the self-diagnosis cassette "Micro-Nurse" into your MPF-II. For each position of the volume control, you are suggested to do the following:

1. Rewind the tape to the beginning.
2. Type: L O A D T "filename" <----
3. Press the Play button on your tape recorder.

After you have finished the three steps, the cursor will disappear. It may take up to 15 seconds before the video display shows

```
NURSE 1 14  40   WAIT
```

Filename (Ø to 6 alphanumeric characters)
The number of file records
The down counter that counts down from 40 through ØØ.
A reminder

If the video display shows

```
NURSE1  14  14  WAIT  OK
>
```

Then your program has been read by the MPF-II without error. If the display shows

```
?  SYNTAX ERROR
```

Then, it is telling you that you have typed the LOADT command incorrectly. If the display shows

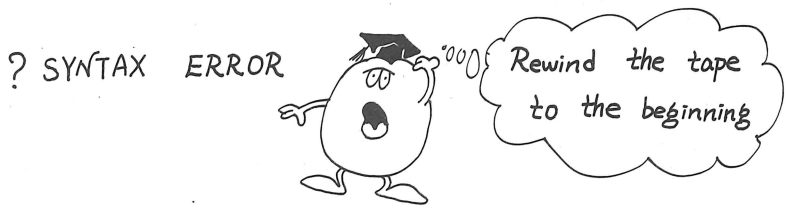
```
ERROR
```

Then, your MPF-II has been loaded with incorrect data.

In summary, the following possibilities may happen:

a. The message? SYNTAX ERROR appears.
b. Nothing at all happens.
c. The message ERR or ERROR appears (with or without a beep).
d. The computer goes "beep" and nothing appears.

In case a, do not reset the volume control, but go back to step 1 where you rewind the tape.




In cases b and c, make sure you waited for 15 seconds before giving up. If there is no prompt characte or cursor, and the MPF-II does not respond to its keyboard, press RESET, set the volume control a bit higher and go back to step 1. Once in a great while the LOADT command may not work properly, and the cursor will appear on the screen immediately without waiting for the tape to be LOADed. If this happens just turn your MPF-II off and then on again with the power switch on the switching power supply, and then try LOADing the tape again.

The use of the LOADA command is very similar to that of the LOADT command. After you have entered the LOADA command, filename, and pressed the PLAY button on your tape recorder, the three possibilities mentioned previously may occur. If any one of those phenomenon happens, you can solve the problem following the procedures mentioned above.

If the MPF-II goes "beep" and the display shows the prompt character ">" and square cursor, then you are on the right track. When you hear the beep, wait another fifteen seconds. Either you will get an error message (case c), or the prompt character (>) and the blinking cursor will reappear. If they do reappear, stop and rewind the tape. Mark the position of the recoder's volumn control, so that you can use this setting each time you LOAD a tape in the future. Then type



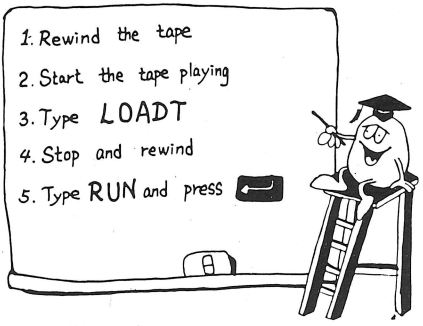

**The Usual Procedure for Loading Tapes**
(once the recorder's volume control has been set correctly)

1. Rewind the tape.
2. Start the tape playing.
3. Type LOADT.

After you press RETURN the cursor will disappear. Nothing happens from 5 to 20 seconds, and then the MPF-II beeps. This means that the tape's information has started to go into the computer. After some more time (depending on how much information was on the tape, but usually less than a few minutes) the MPF-II beeps again and the prompt character and the cursor reappear.

4. Stop the tape recorder and rewind the tape. The information has been transferred, and you are finished with the tape recorder for the time being.

5. Type RUN and press RETURN, and your program will begin to execute.



Computerniks use many different words to describe the process of taking information from a tape and putting the information into the computer. The computer is said to "read" (pronounced "read") the tape. The information on the tape is said to be "entered" or "read" (pronounced "red") into the computer. The act of reading a tape is also called "loading" a tape into the computer and the information on the tape is said to be "loaded into" the computer. All these expressions are ways of saying the same thing.

**A Helpful Hint**

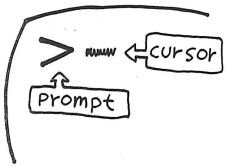
What is it that the computer finds so interesting about these tapes? Listen to one of them. It's not music to your ears. Yet you can recognize some of the sounds the computer listens for. The information starts with a steady tone. The tone is at 1000 cycles per second. This pitch is just below the C two octaves above middle C. After the tone comes a burst of sound rather reminiscent of a rainstorm.

When you are used to the sound of a good tape, you can quickly check a tape by ear to see if it is a computer tape or not. If you can tell what the tape contains by listening to it, you are a mutant, and will go far in the computer world.



## 4•5 A First Look at the Print Statement

When you power on the MPF-II, the prompt character (>), followed by the blinking cursor, will appear at the left edge of the screen each time you press RETURN.




Now that you have the prompt character (>) and the blinking cursor on the screen, you are ready to begin using the MPF-II BASIC language.

Type

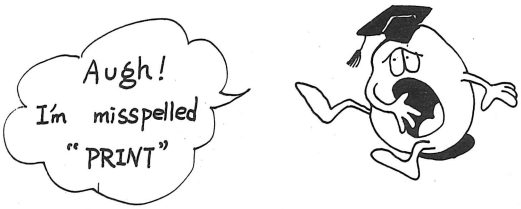
```
PRINT "HELLO"
```

and the computer will print the word

**HELLO**

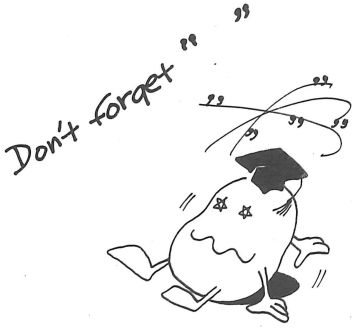
on the next line. If it didn't, ask yourself this question: "Did I forget the RETURN?" If you misspell the word "PRINT", you will get this error message:

```
?SYNTAX ERROR
```




If you forget either the first quote or both quotes, the computer will print a zero (you can tell it's a zero by the slash):

If the final quote is the last character before the RETURN, you don't have to type it: the word "HELLO" will be printed with or without it. It's a good idea to put the end quote in anyway, though. The habit of putting in the final quote will become important later. This manual will assume that you use the final quote.




The statement PRINT "HELLO" is an instruction to the computer telling it to display on the screen all the characters between the quotes, in this case a word of greeting. You can use the PRINT statement to tell the computer to display any message you wish. However, if you type much beyond 240 characters, the computer will start to beep, then it will give you a backward slash and let you start over again.

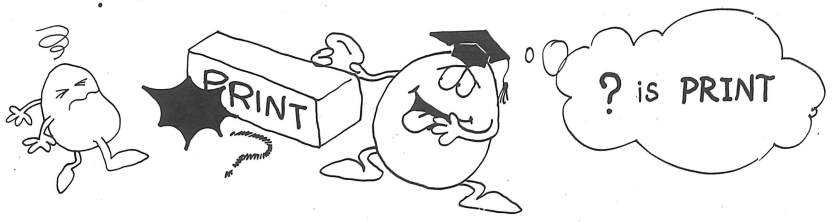
## 4 • 6 Abbreviated Print Statement

MPF-II BASIC allows you to abbreviate the PRINT statement with a question mark (?). Here are some examples you can try:

For example

? 3*2 and PRINT 3*2 is the same

? SQR(2) and PRINT SQR(2) is the same



# 5

# THE MPF-II AS A CALCULATOR

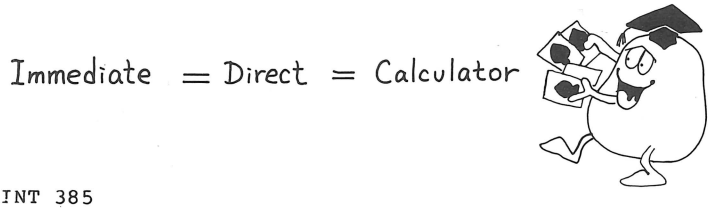
This chapter teaches you how to use your MPF-II as a calculator.

## 5 • 1 Immediate Mode

When you first put the MPF-II in BASIC, it is in immediate mode, also called direct or calculator mode. In this mode, the computer responds immediately to any instructions you issue it. Try typing in this example:

```
PRINT "385"
```

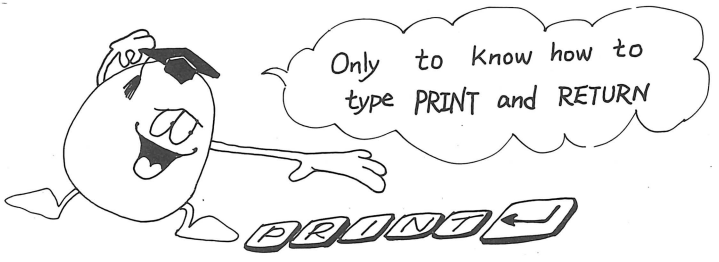
The computer obediently prints the number 385 on the next line, as expected. But type




```
PRINT 385
```

and the computer again prints the number, without error message about the missing quotation marks. MPF-II will let you PRINT any number at all without error in quotes.

Without further study, the MPF-II can be used as a desk calculator.




## 5 • 2 Addition and Subtraction

Try this on your MPF-II

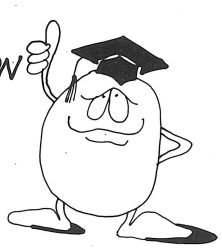
```
PRINT 3 + 5
```

The answer, 8, appears on the next line. The MPF-II can do six different elementary arithmetic operations:

1. ADDITION. Indicated by the usual plus sign (+).
2. SUBTRACTION. Uses the conventional minus sign (-).

Try again

```
PRINT 8 - 3          RETURN
PRINT 38 + 56        RETURN
PRINT 79 + 63 - 54   RETURN
```




## 5 • 3 Multiplication and Division

3. MULTIPLICATION. Many people use an "X" to represent multiplication. This could be confused with the letter "X". Some people use a dot (.), but this could be confused with a period or a decimal point. So the MPF-II uses an asterisk (*). To find 7 times 8 (in case you don't remember the answer), just type

```
PRINT 7 * 8
```

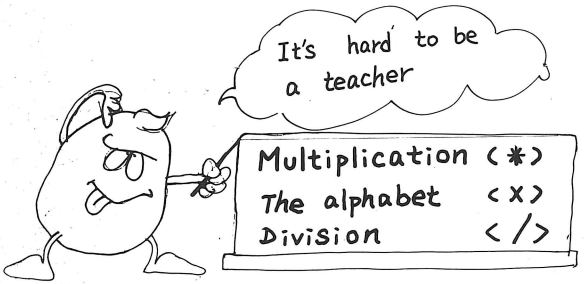
and have your memory jogged.

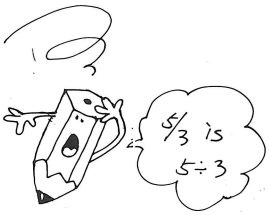
4. DIVISION. As is customary, use a slash (/). To divide 63 by 7, type

```
PRINT 63/7
```

and the correct answer will appear.

Try dividing 3 by 2. The answer is one and one half. The MPF-II gives the answer to you in the decimal form: 1.5.

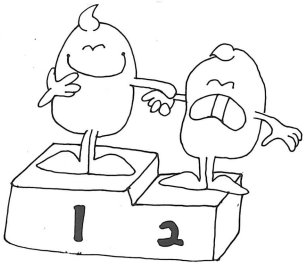


One thing we should point out here is that you can do more than one arithmetic operation in the same instruction. For example, it is legal to say

```
PRINT 3 * 5 * 9 * 4
```

The exact rules governing such usage will be given later, but you can experiment with it now if you wish.




First multiplication and division
and then addition and substraction

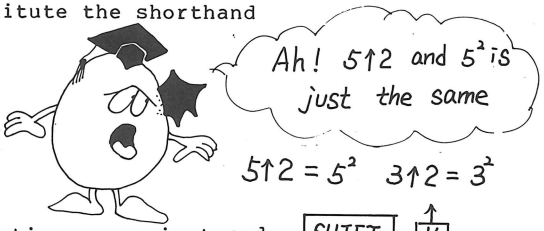
## 5 • 4 Exponentation

5. EXPONETIATION. It is often handy to multiply a number by itself a given number of times. Instead of bothering to write

```
PRINT 4 * 4 * 4 * 4 * 4
```

you can substitute the shorthand

```
PRINT 4↑5
```




The upward pointing arrow is typed: SHIFT ↑ K

There is nothing special about exponentiation. It is just an abbreviation for repeated multiplication! In non computer-notation, this would be written with a superscript five, like this: $4^5$

First ^
and them * and /
finally + and −



## 5 • 5 Math Function

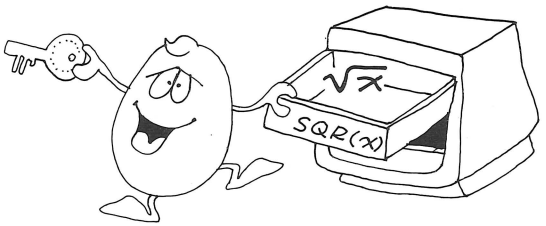
The MPF-II has many math functions. Here is just a brief description of the "SQR" function.

Try this on your MPF-II:

```
PRINT    SQR  (2)
```

The answer, 1.41424, appears on the next line.

SQR computes a positive square root. This is a special implementation that executes more quickly than ↑.5 .




## 5 • 6 MPF-II's Format for Numbers

Type

```
PRINT 45.34Ø
```

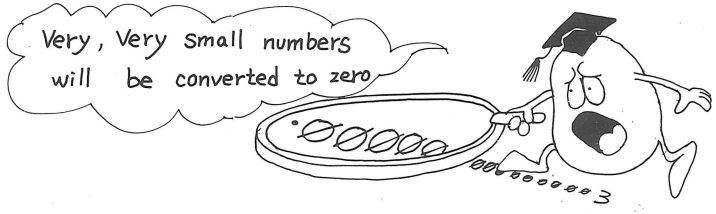
Your computer responded with

```
45.34
```

and didn't PRINT the trailing zero. The MPF-II does not PRINT leading or trailing zeros, that is, zeros that are at the beginning of a number and to the left of the decimal, or zeros that are at the end of a number and to the right of the decimal.



Very, very small numbers (between about .000000000 0000000000000000000000000003 and -.0000000000000000000000 000000000003) will be converted to zero by the MPF-II. (We hope that was the right number of zeros.) An easier way to write these numbers is 3 * 10↑- 39 and -3 * 10 ↑ - 39. Don't take our word for it. Try it yourself.




Now type

PRINT 985788.6898

Surprise! The last two digits are lost, and the number left behind is the closest approximation the computer can think of. This process is called "rounding". Try typing

PRINT 788.6898

Your computer did not round the number, but PRINTed it just the way you typed it. Madness you say? Ah, but there is a method to this seeming madness. Numbers are rounded only if they have more than nine digits. Any number that has fewer than ten digits will not be rounded. The computer does the best it can, but it only has nine digits to work with.

If you type a PRINT statement with a long number like

1234567890

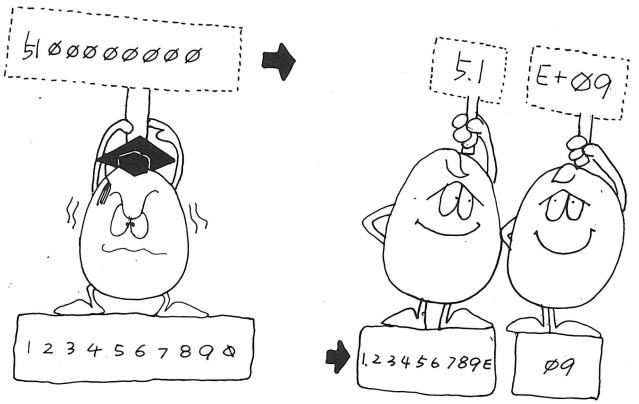
the MPF-II responds with

1.23456789E+09

Scientific notation
"E"

The numbers 1234567890 and 1.23456789+09 have the same value. Really, the number PRINTed by your computer is in "scientific notation". If you need numbers like this you probably know how to read them.

Try some more numbers. How many digits can a number without a decimal point have before the MPF-II changes it to scientific notation? If scientific notation seems complicated, don't worry. You probably won't be wanting to use number that require it for some time yet. Remember that any number will be PRINTed just the way you type it if the number is surrounded by quotes. However, the MPF-II can't use numbers in quotes for some time yet.

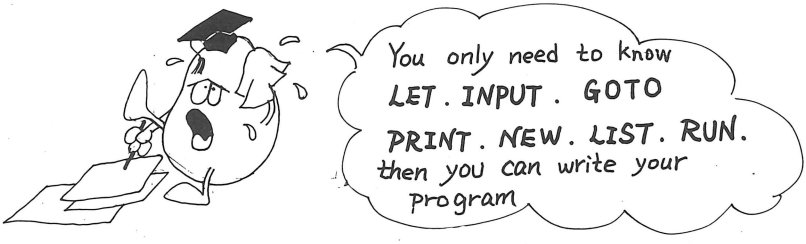


# 6

# ELEMENTARY PROGRAMMING

This chapter teaches you how to start writing your own BASIC programs on the MPF-II.

BASIC is a programming language. Like any programming language, it consists of a set of statements, which you combine to create programs. A program defines the task you want the computer to perform.




## 6 • 1 Statements

A program consists of one or more statements which provide the MPF-II with an exact and complete definition of the task which it is to perform. If the task is short and simple, the program can be short and simple as well. The immediate mode instructions we have experimented with so far are each small, simple programs. Each one has just one statement - one instruction to the MPF-II. These are trivial cases. Most programs have at least 10 statements, and some have move than 1000, or even more statements. Consider the following statements:

```
PRINT  "MICRO"
MICRO
PRINT  "MICRO PROCESSOR"
MICRO  PROCESSOR
PRINT  "MPF-II"
MPF-II
PRINT  "HELLO MPF-II"
HELLO  MPF-II
```

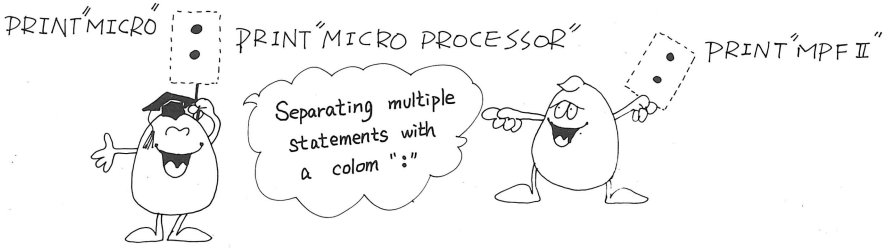



Each of these immediate mode programs prints a line of text on the display screen. Each program has exactly one statement and exactly one line.

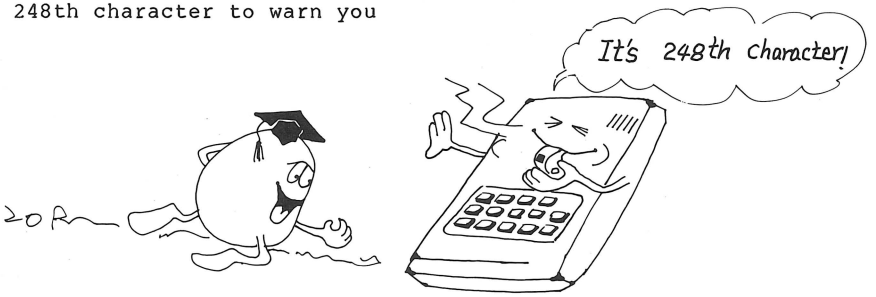
MPF-II allows you to put more than one statement on a line. You can separate multiple statements on the same line with a colon (:). Compare this immediate mode program with the example above:

```
PRINT "MICRO": PRINT "MICRO PROCESSOR"
             : PRINT "MPF-II"
MICRO
MICRO PROCESSOR
MPF-II
```

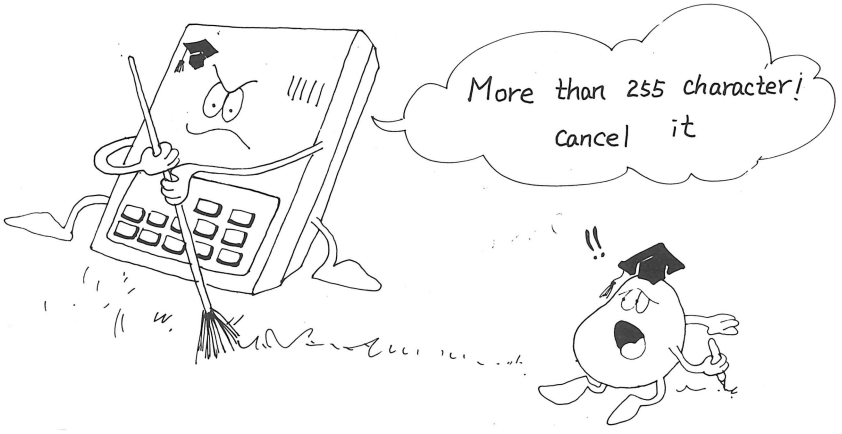
This three-statement, one-line program prints the same three lines of text as the previous three one-statement programs.




There is no specific limit to the number of statements on one MPF-II BASIC program line. Remember that a line cannot be longer than 255 characters, though. If you are typing a long line, the computer will start beeping when you type the 248th character to warn you




are approaching the limit. If you exceed the limit, it automatically delete the line, just as if you had typed CTRL-X, and you must start over. It does not perform any of the instructions you typed on the too-long line. So there is a limit as to how much you can do with a one-line immediate mode program.




## 6 • 2 Programmed Mode

The programming we have done so far is educational and, hopefully, interesting, but there is only so much you can do in immediate mode.

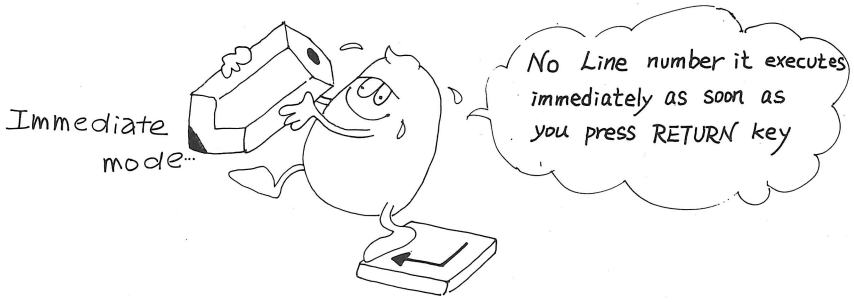
Another problem with immediate mode programs is that you have to retype the program each time you want to use it. There are some advanced editing techniques which we will discuss shortly that will allow you to reuse the program as long as it still appears on the display screen, but this is still a limitation.

What you need is a way to enter several program lines and to hold off using those lines. That way you can write programs to do tasks that are too complex for one-line programs.

There is a way to get around the problems of immediate mode, and that is to write programs in programmed mode, also called deferred or indirect mode. In programmed mode, the computer accepts and stores the program in its memory, but does not perform any of the operations specified by the program until you tell it to do so. You can enter as many program lines as you want. Then, when you enter the appropriate command, the computer performs the operations specified by the programmed mode program.

We say the computer executes or runs a program when it performs the operations which the program specifies.

In immediate mode a program is executed as soon as you press the RETURN key.




In programmed mode you must issue the RUN command to execute a program. Each time you do so, the program runs all over again.

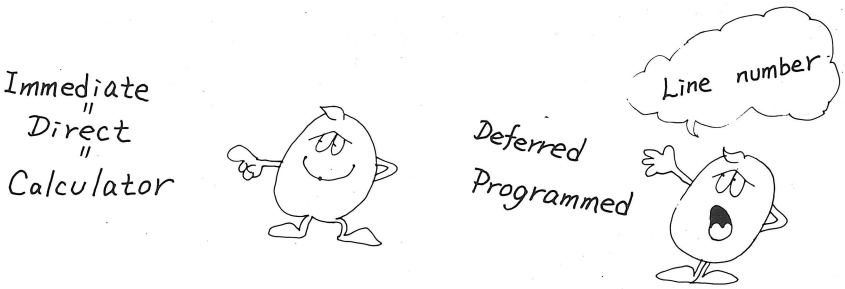


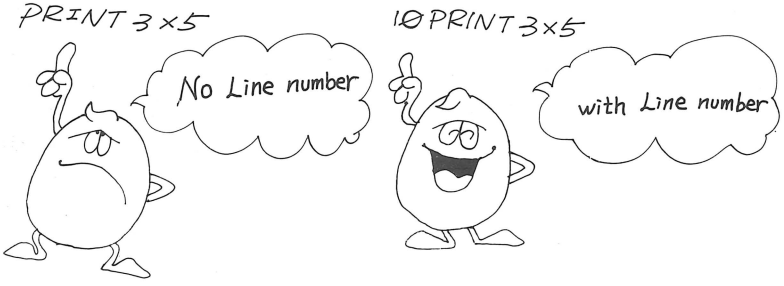

## 6 • 3 Line Numbers

To tell the computer to store a statement, just type a number before typing the statement. For example, if you type

```
100 PRINT 3 + 4
```

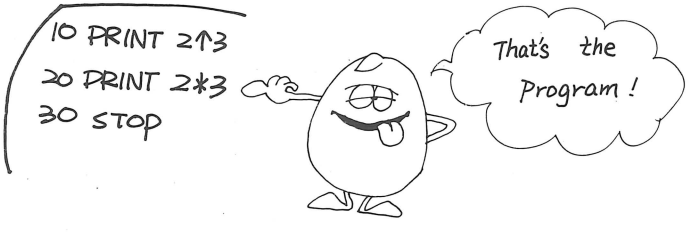
nothing seems to happen, when you press RETURN. The MPF-II has stored the statement.



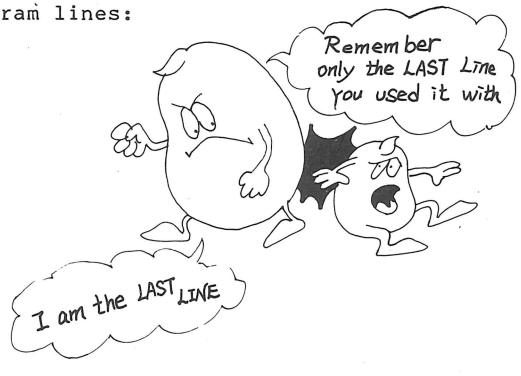



Line numbers make programmed mode possible. A line number is simply a one, two, three, four, or five-digit number entered at the beginning of a program line. The line number is the only difference between a programmed mode program line and an immediate mode program line. There are some instructions that can be used only in immediate mode and others that can be used only in programmed mode; we will discuss them later.

Try this sample programmed mode program:

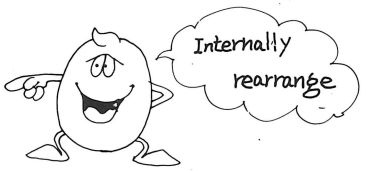


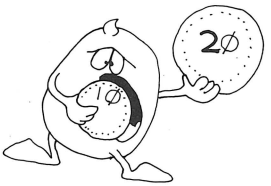

Each line number must be unique. No two program lines can have the same number. If you use the same line number more than once, the computer only remembers the last program line you used it with. To see how this works, type in these program lines:



Line numbers determine the sequence of program lines in a BASIC program. The first line must have the smallest line number, while the last line must have the largest line number. Even if you type in the lines out of order, the MPF-II will internally rearrange them in the proper sequence by line number. Consider this program, with line numbers out of order:

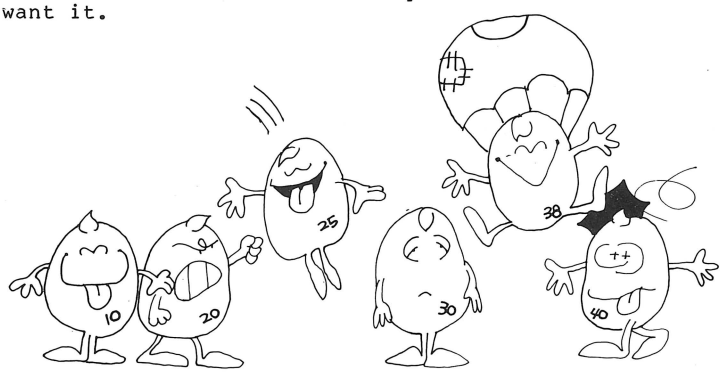
```
>30 PRINT "CUT"
>10 PRINT "FISH"
>20 PRINT " OR"
>40 PRINT "BAIT"
>50 END
> RUN
  FISH
  OR
  CUT
  BAIT
```




To prove that the MPF-II does not forget programmed mode in programs, run the program again.

It is a simple matter to add program lines to a program that is currently in the computer's memory. You can add a line to the beginning, the end, or anywhere in the middle of a program by typing the line with a line number that will position it where you want it.



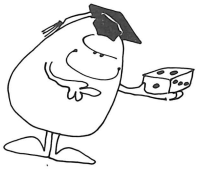
Suppose you wanted to add a line to the beginning of the last example program. As long as you have not typed the command NEW, the program will still be in the MPF-II's memory. Since the lowest line number currently in that program is 1Ø, any program line you type in now with a line number less than 1Ø will be placed at the beginning of the program. Try this:

```
>5 PRINT "EITHER"
>RUN
EITHER
FISH
OR
CUT
BAIT
```

It's a good thing the original program started with line 1Ø rather than line Ø1. It's always a good practice when assigning line numbers to start your program with a fairly high line number and leave plenty of room between line numbers so you can add program lines later on.

You can put more than one statement on a single program line. The first statement follows the line number. The second statement follows the first, with a colon (:) in between. Colons separate the statements on a multiple-statement line.

Multiple-statement program lines are allowed in MPF-II BASIC in both programmed and immediate modes. In both cases, the line length limit is 255 characters, as we described earlier.



## 6 • 4 New, List, End and Home

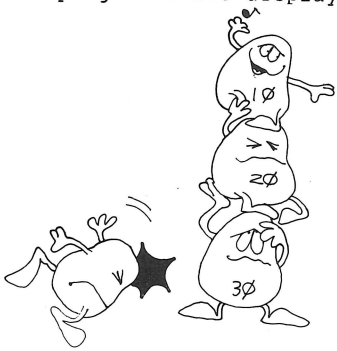
### Clearing Out Old Programs

Because the MPF-II stores programmed mode programs in its memory, you must specifically instruct it to erase an old program before you type in a new program. Do this by typing the command NEW. If you forget to type NEW, your new program will be mixed in with your old program.




### Listing Program Lines

You can see what program lines the computer has stored in its memory at any time by typing the command LIST. Try it right now. If you have not typed NEW, since you tried the last example, you should see the last program lines displayed on the screen:







### Ending Programs Properly

The end of an immediate mode program is obvious. Not so with programmed mode, as we shall see. The END statement tells BASIC to stop executing your program.



Example

To make sure that the computer's memory is cleared of any previous programs, type

```
NEW
```

Like almost everything else you have seen, NEW has to be followed by a RETURN. To tell the computer to store a statement, just type a number before typing the statement. For example, if you type

```
10Ø PRINT 3 + 4
```

nothing seems to happen, even if you press RETURN. The MPF-II has stored the statement. To see that it has stored the statement, you type the instruction

```
LIST
```

Try it. Unless you mistyped something (and probably got a

```
?SYNTAX ERROR
```

for your effort),

```
10Ø PRINT 3 + 4
```

appears on the screen. Now type the statement

```
RUN
```

and the answer

```
7
```

appears on the screen.

Typing RUN caused your stored statement to be executed, but the computer has not forgotten the statement. You can RUN the same statement as many times as you like. Try it.

What's more, the computer does not forget the stored statement when you clear the screen. Here's a new way to clear the screen:

clear the screen and put cursor at top



HOME

The HOME command clears the screen and puts cursor at top.

The HOME command can be used in deferred execution as well as immediate execution. To try this out, type

```
10Ø HOME
```

Now when you type

```
RUN
```

the computer faithfully executes the stored statement and clears the screen. Type

```
NEW
```

and then

```
LIST
```

and see what happens. Typing NEW has caused the stored statement to be lost permanently. Type

```
RUN
```

and nothing appears on your screen. That is because your old statement has been erased by the NEW command.

It is possible to store many statements by giving each of them a different number. Try typing this:

```
1 PRINT "HELLO"
2 PRINT 4↑5
3 PRINT 67/12
```

Nothing much has happened so far. But now type

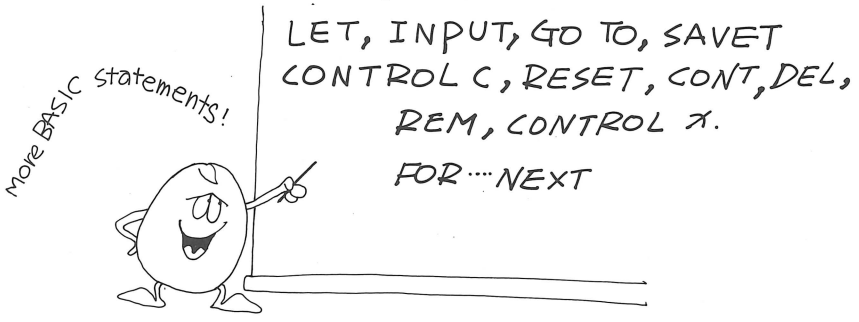
```
RUN
```

and watch the answers appear.

## 6 • 5 More Basic Statements

This section introduces more of the frequently used statements when programming your MPF-II. From here on, we can begin to write interesting programs for home, school, and personal use.

Now you will learn what the following statements and commands mean, how they work, and how to use them.




We will emphasize having fun while learning things that can be used for both fun and serious use of the computer.

Now, we are ready to enter a program in the memory of the MPF-II, our program causes the MPF-II to PUT John's name on every line of the screen. Here is the program.




The above program consists of three statements.

1. The statement

10 HOME

tells the computer to clear the screen.

2. The statement

```
20 PRINT "JOHN"
```

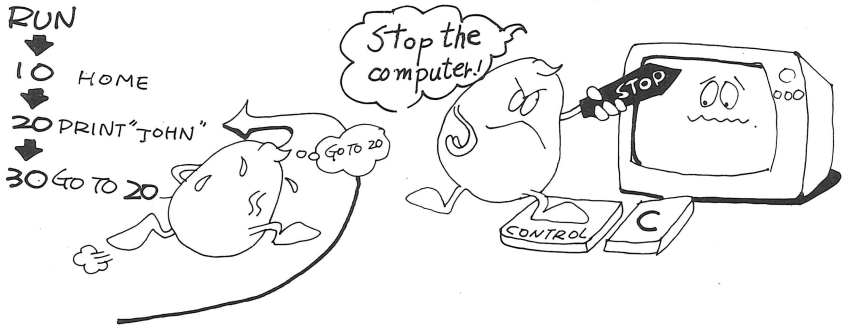
tells the computer to print the word JOHN on the screen.

3. The statement:

```
30 GOTO 20
```

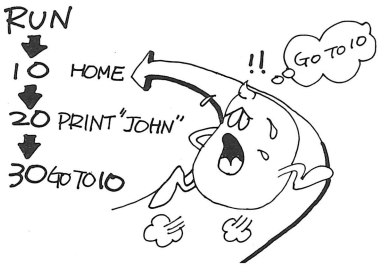
tells the computer to go to line 20 and continue.

When we type RUN and press RETURN, the computer begins the program, beginning with the smallest line number. The computer executes line 10, then 20, then line 30, then line 20, then line 30, and so on -- until you press the CONTROL C, which stops the computer.




Here is a slightly different program. Draw arrows to show the order in which things are done.

```
RUN
 10 HOME
 20 PRINT "JOHN"
 30 GO TO 10
```



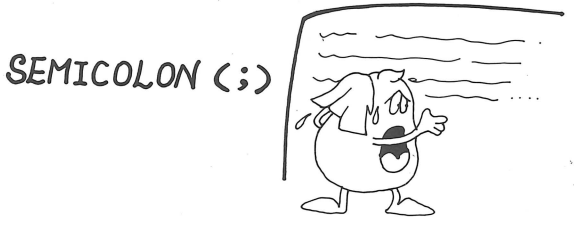

This program causes JOHN's name to blink rapidly in the upper left corner of the TV screen. Why? Because HOME clears the screen and also moves the cursor to its "HOME" position in the upper left corner of the screen. Difficult to explain, but easy to see when you RUN this program on your MPF-II!

Now, we want to change line 20, but only line 20, we do not type NEW. Instead, we do this:

```
We type:     20 PRINT "JOHN"; <-- (Semicolon at the end)
We type:        LIST
It prints:   10 HOME                <-- OLD Line 10
             20 PRINT "JOHN";       <-- NEW Line 20
             30 GO TO 20            <-- OLD Line 30
```

Our new line 20, with a semicolon on the right end, replaces the old line 20 which did not have a semicolon.

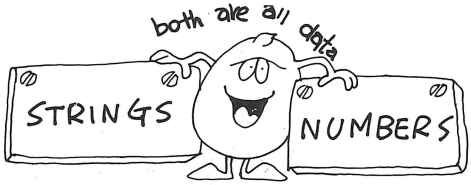
What does the semicolon do? As usual with computers, the best way to find out is to experiment -- try it and see what happens.




So, type RUN and press the RETURN ke..

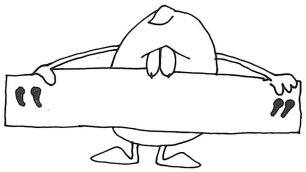
## 6 • 6 Data

The main business of computer programs is to input, manipulate, and output data. So the way a programming language handles data, whether it be numbers or text, is very important indeed. We will now describe the types of data you may encounter in an MPF-II BASIC program.




### Strings

A string is any character or sequence of characters enclosed in quotation marks. We have already used strings with the PRINT statement as messages to be displayed on the screen. Here are some more examples of strings:



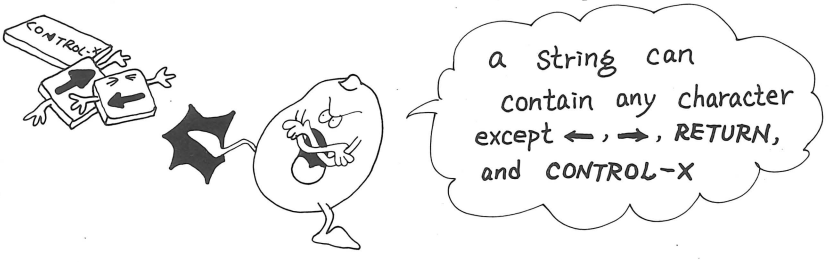
"BYTE , BIT"
"MPF-II BASIC"
"R6502 CPU"
"MICRO-PROCESSOR II"
"BUILT IN BASIC LANGUAGE"

String is enclosed in quotation marks

With just a few exceptions, a string can contain any character you can produce at the keyboard using the normal alphabetic and numeric keys, with or without the CONTROL or SHIFT keys. The exceptions are <—, —>, RETURN and CONTROL-X. These exceptions either move the cursor around or end the line you're working on, or both.

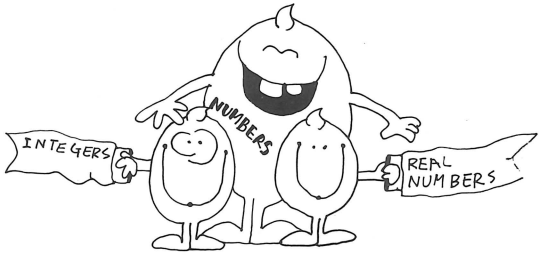
Strings can be any length from Ø to 255 characters. A string with no characters in it is called the null string.

We will describe "string" further in chapter 7.

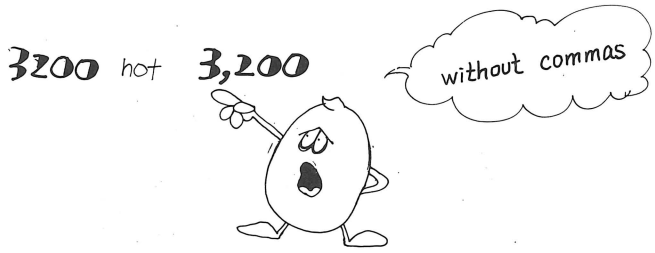



**Numbers**

There are two kinds of numbers that can be stored in the MPF-II: integers, which are numbers without any fractional part, and real numbers (also called floating point numbers), which can have fractional parts.




You must express all numbers without commas. For example, you must use 32000, not 32,000.




**1. Integers**

An integer is a number that has no fractional portion or decimal point. The number can be negative (-) or positive (+). An unsigned number is assumed to be positive. Integer numbers must have values in the range -32767 to 32767. The following are examples of integers:

0
1
44
32699
-15

**2. Real Numbers**

A real number can be a whole number, a whole number with a decimal fraction, or just a decimal fraction. The number can be negative (-) or positive (+). If the number has no sign it is assumed to be positive:

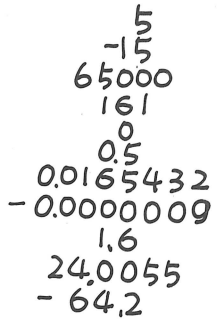
The smallest (most negative) real numbers is:

-1.00000000000000000000000000000000000

and the largest is:

1.00000000000000000000000000000000000

Here are some examples of real numbers:




When the value of any fractional number gets closer to zero than about 0000000000000000000000000000000003, it will be converted to zero.

### 3. Scientific Notation

Very large and very small real numbers are represented in MPF-II using scientific notation. Any number that has more than nine digits in front of the decimal point will be expressed in scientific notation. Any fractional number closer to zero than ±.01 will be expressed in scientific notation.

A number in scientific notation has the form:

number E ±ee

where: number is an integer, fraction, or combination, as illustrated above. The number portion contains the number's significant digits; it is called the coefficient. If no decimal point appears, it is assumed to be to the right of the coefficient.

E is always the letter E. It stands for the word exponent.

± is an optional plus sign or minus sign.

ee is a one-digit or two-digit exponent. The exponent specifies the magnitude of the number, that is, the number of places to the right (positive exponent) or to the left (negative exponent) that the decimal point must be moved to give the true decimal point location.

Here are some examples of scientific notation:

| Standard Notation | Scientific Notation |
|---|---|
| 1000000000 | 1E + 09 |
| .000000001 | 1E - 09 |
| 200 | 2E + 02 |
| -1.23456789 | -1.23456789+09 |
| -0.00000123456789 | -1.23456789- 06 |

As you can see, scientific notation is a convenient way of expressing very large and very small numbers. The maximum and minimum values for real numbers, which we just expressed with lots of zeros, can also be expressed as 1E + 38, respectively (much more compact). Similarly, the closest a number can get to zero is 3E-38.

maximum ⟶ 1E+38
minimum ⟶ -1E+38
Closest to zero ⟶ 3E-38

**4. Roundoff**

We mentioned earlier in this chapter that real numbers can have as many as nine digits of precision. For a number greater than 1 or less than -1, this means only the leftmost nine digits can be nonzero. The MPF-II rounds off any digits in excess of nine. Here are some examples (note that large numbers print in scientific notation):

```
> PRINT  123456789/
 1.23456789E +09
> ? -123456789/123456789
  -1.23456789E +17
> ? -150000475.75
  -150000476
> ? 90000000.7558
  90000000.8
```

Fractional numbers (those between 1 and -1) are subject to the same limitation. In this case, though, the nine digits of precision start with the first nonzero digit to the right of the decimal point. Here are some examples:

```
> PRINT      .123456789
 .123456789
> ? -.123456789 123456789
 -1.23456789E+17
```

## 6.7 Let and Variables

### Variables

Thus far in our discussions of data we have only considered constant values. It is often handier to refer to data items by name rather than value. That is what variables are all about.

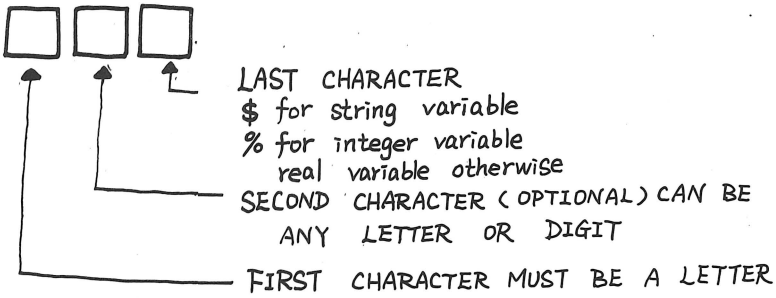
If you have studied elementary algebra, you will have no trouble understanding the concept of variables and variable names. If you have never studied algebra, then think of a variable name as a name which is assigned to a letter box. Anything which is placed in the letter box becomes the value associated with the letter box name, until something new is placed in the letter box. In computer jargon we say a value is stored in a variable.



A variable does not always have to refer to the same value. That is its real power -- it can represent any legal value. You can change its value during the course of a program. BASIC has a number of statements to do this; we will describe them later.

**Variables Names in MPF-II BASIC**

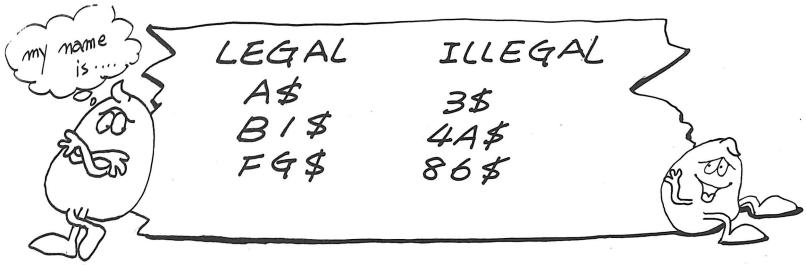
A variable name can have one, two, or three characters in MPF-II BASIC. The following rules apply:




Thus the last character of the variable name tells MPF-II BASIC which type of data the variable represents.

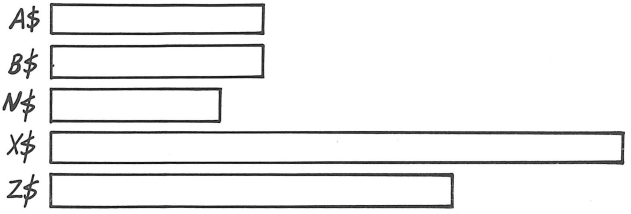
**String Variables**

A string variable in MPF-II can store a string value of any length from 0 to 255 characters. Here are some examples of string variable name, both legal and illegal:




We can imagine that, a string variable is a box placed in the computer's memory in which we can store information.

We can give a box a name consisting of a letter followed by a dollar sign ($). For example, below are boxes called A$, B$, N$, X$, and Z$.




Into any box, we can put a string. A string is simply any bunch of keyboard characters, typed one after the other. For example:

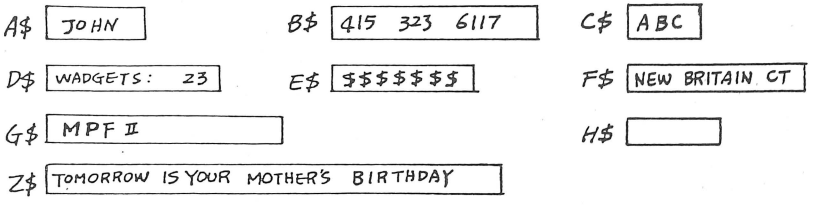
| | |
|---|---|
| A string can be a name: | JOHN |
| A string can be a number: | 123456789 |
| A string can be gibberish: | AB#$%JFDZ?* |

Almost any keyboard character can be part of a string. One improtant exception is the double quotation mark ("). It can't be part of a string. Instead, quotation marks are sometimes used to enclose a string, as follows.




The string JOHN is enclosed in quotation marks.

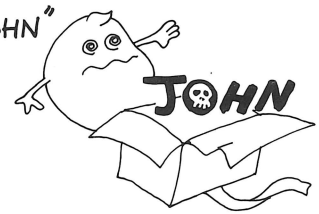
Below are some boxes. We have already stored strings in some of these boxes. For example, JOHN is in box A$ and 415 323 6117 is in box B$.



A box will be as short or as long as necessary to hold the string we want to put into it. Actually, there is a limit on string length. You may use strings with up 255 characters. Here is how we stuff a string into a box, deep down inside the computer. To tell the computer to put JOHN into box A$, we use a LET statement, without a line number, like this:

We type: LET A$="JOHN"

It prints: >_




The MPF-II has put the string JOHN into box A$. Note how we told the computer to do this.

```
> LET A$ = "JOHN"
```

Let's find out what is in box A$. To do this, we will tell the MPF-II to print what is in A$.

We type: PRINT A$

It prints: JOHN

>_

To tell the computer to print the string that is in A$, we type:

```
> PRINT A$
```

The names A$, B$, C$, and so on, which identify boxes, are called variables. Since these boxes can hold strings, these variables are called string variables.

A$ is a string variable,
B$ is a string variable,
C$ is a string variable,

and so on.

Later, you will learn about numeric variables, which can store only numbers. We will use numeric variables when we want to do arithmetic with numbers.

A$, B$, C$,..., Z$ are string variables. The string in box A$ is called the value of A$; the string in box B$ is the value of B$; the string in box C$ is the value of C$; and so on.

We use the LET statement to tell the computer to "put a string into a box" or, more technically, "assign a string to a string variable."

We type: LET C$ = "GREEN" ← Put GREEN into box C$

We type: PRINT C$ ← display the contents of box C$

It prints: GREEN

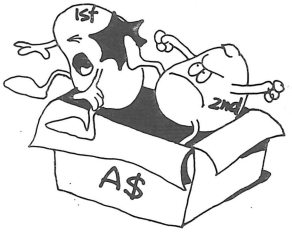
IMPORTANT NOTICE! The word LET is optional; it can be omitted.

Instead of: LET C$ = "GREEN"

We can type: C$ = "GREEN"




A box can hold one string at a time. When the computer puts a string in a box, it first erases the previous contents of the box.

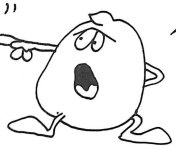



Last string erases the previous contents

Remember, a variable can have only one value at a time. When we assign a value to a variable, the computer automatically erases any precious value of that variable.

Now try this program. First, type NEW to erase any old program in the MPF-II's memory. Then, enter the following program.

```
10 HOME
20 A$="JOHN"
30 PRINT A$;
40 GO TO 30
```




RUN the program. The screen should fill with JOHN's name. Let's compare the following two programs.

## OLD PROGRAM

```
10 HOME
20 PRINT "JOHN";
30 GO TO 20
```

Semicolon

## NEW PROGRAM

```
10 HOME
20 A$ = "JOHN"
30 PRINT A$ ;
40 GO TO 30
```

Semicolon here!

In the old program, we include JOHN's name as a string in the PRINT statement. In the new program, we put the string in the box A$, then tell the MPF-II to print what is in A$.

Draw arrows to show the order in which the computer does the statements in the new program.

RUN

10 HOME ← THIS IS DONE ONCE

20 A$="JOHN" ← THIS IS DONE ONCE

30 PRINT A$

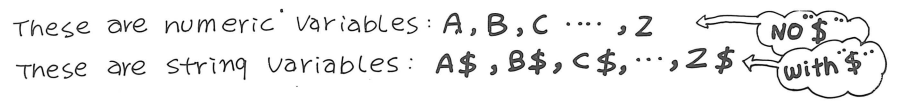
40 GO TO 30



← AROUND AND AROUND AND AROUND UNTIL PRESSES CONTROL C

**Numeric Variables**

You already know about string variables. Now let's talk about numeric variables. Numeric variables are names of boxes that can hold only numbers. Any letter of the alphabet can be a numeric variable.




A string variable always ends with a dollar sign ($). A numeric variable never ends with a dollar sign. String variables may have any string as a value. Numeric variables may have only numbers as values. And values of numeric variables must be typed without quotation marks, as you will see later.

We can use the LET statement to instruct the computer to "put a number in a box." To say it more technically, we are assigning a numerical value to a variable.

We type: `LET A=7` ← PUT 7 into box A
We type: `PRINT A` ← print the contents of box A
It prints: 7

In this program the variable is A and the value assigned to it by the LET statement is 7.

Complete the following:

(a) We type:
```
LET X = 23
PRINT X
```
It prints:_____

(b) We type:
```
LET Z = -1
PRINT Z
```
It prints:_____

(c) We type:
```
LET A = 1
LET A = 2
PRINT A
```
It prints:_____

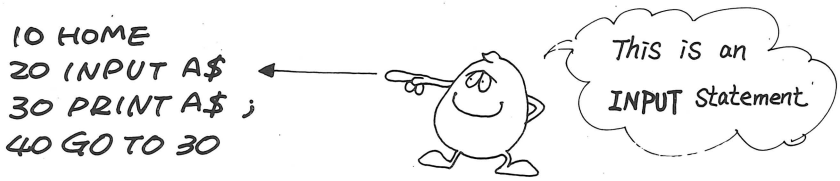
(d) We type:
```
LET D = 7
LET W = D
PRINT W
```
It prints:_____

As you read through this book, you will see less and less of LET. If you read other books and magazines, however, you may occasionally encounter it. In your programs, we encourage you not to use LET.

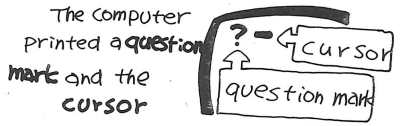
## 6 • 8 Input

### Input Statements

Now we will tell you how to put strings in boxes -- the easy way -- by using a new statement called the INPUT statement.




We stored the above program in the MPF-II, then typed RUN. Here is what happened.




The MPF-II is executing the INPUT statement. The statement

```
2Ø INPUT A$
```

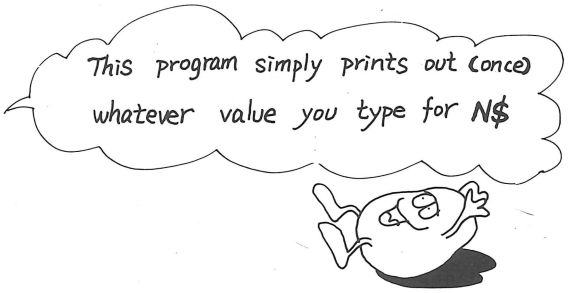
tells the MPF-II to:

(a) type a question mark;
(b) turn on the cursor;
(c) wait for someone to type a string. When you type a string and press the RETURN key, the MPF-II puts the string into box A$ and goes on to the next statement in line number sequence.

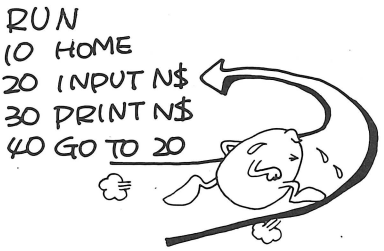
Let's cooperate with our ever-patient computer and type in a string. To keep things familiar, we will type JOHN's name and then press the RETURN key.

The following program will help you learn more about the INPUT statement. You can use it to experiment!

```
NEW
10 HOME
20 INPUT N$
30 PRINT N$
40 GO TO 20
```

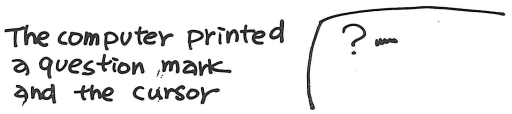



Draw arrows to show the order in which statements are done by the computer.




In following this program, the MPF-II will always return to the INPUT statement, print a question mark, turn on the cursor, and wait for the next value of N$.

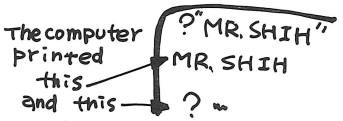
We stored the program in the MPF, then typed RUN and pressed the RETURN key. Here is what happened.




We then typed the string "MR. SHIH" (including the quotation marks).




Then we pressed the RETURN key.




The computer printed the string that we typed, enclosed in quotation marks, following the question mark. We continued entering strings in response to question marks. When we finished experimenting, press CONTROL C to stop the computer.

**Enhanced INPUT**

The INPUT statement, as we have known it, tells the computer to put a question mark on the screen, then wait for the user to answer. Wouldn't it be nice if, instead of printing just a question mark, the computer would put some informative words on the screen, so that the user knows what the computer wants? The following program has an "enhanced" INPUT statement that is more informative.

```
10 HOME
20 INPUT "WHAT IS YOUR NAME"; N$
30 PRINT N$
40 GOTO 30
```




Let's examine the "enhanced" INPUT statement. The statement:

```
20 INPUT "WHAT IS YOUR NAME"; N$
```

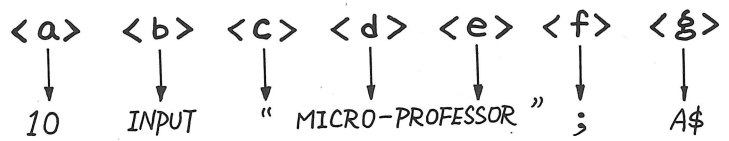
tells the MPF-II to:

(a) print WHAT IS YOUR NAME on the screen;
(b) print a question mark;
(c) turn on the cursor; and
(d) wait for someone to type in a string and press the RETURN key.

Look at the statement carefully. It consists of:

(a) a Line number,
(b) the word INPUT,
(c) quotation marks,
(d) a string,
(e) quotation marks,
(f) a semicolon,
(g) a string variable

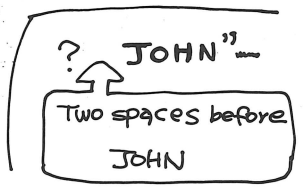
One more time:




The question mark is always an automatic part of an INPUT statement. When an INPUT statement includes a string, the question mark is printed at the right end of the string. We type RUN and press RETURN.




If you type your name and press the RETURN key, the computer will print it "everywhere" on the screen. If you wish to include spaces in front of your name, remember to use quotation marks. For example:

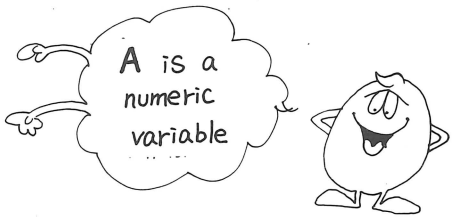



Now, when JOHN presses ENTER, his name will appear everywhere on the screen, but always preceded by two spaces.

**Input Numbered Variables**

We can also use the INPUT statement to assign a value to a numeric variable.

```
NEW
10 HOME
20 INPUT A
30 PRINT A
40 PRINT
50 GOTO 20
```




We entered the above program and ran it. Here is what happened on the screen.

```
? 258          <-- We typed 258 with no quotation marks.
 258               It printed 258, the value of A.
? -6           <-- We typed -6 with no quotation marks.
 -6                It printed -6, the value of A.
? 12.95        <-- We typed 12.95 with no quotation marks.
 12.95             It printed 12.95, the value of A.
? "73"         <-- We typed 73 with quotation marks.
?REDO              It rejected our "73"
?                  and is waiting for our next try.
```

When we enter a number that is to be stored as the value of a numeric variable, we should not enclose it in quotation marks.

However, if we should enter a number as the value of a string variable, it is OK to enclose it in quotation marks. Remember though, you can't do arithmetic with strings.

Below is a program to compute the distance traveled in one turn of a wheel of diameter D.

```
NEW
10 HOME
20 INPUT "WHEEL DIAMETER";D
30 PRINT 3.14*D
40 PRINT
50 GOTO 20
```

```
RUN
WHEEL DIAMETER? 20 We typed the wheel diameter.
62.80              It printed the distance traveled.

WHEEL DIAMETER? 26 We typed the wheel diameter.
81.64              It printed the distance traveled.

WHEEL DIAMETER? 27 We typed the wheel diameter.
84.78              It printed the distance traveled.

WHEEL DIAMETER?    It is waiting for more.
```

What happens when the computer executes line 40 in the above program? The computer prints an empty line on the screen. This separates each example from the previous example.

What the computer wants us to type is clearly identified by the string WHEEL DIAMETER in the INPUT statement. We can use the PRINT statement to identify the answer that the computer prints. Change line 39 to the following.

```
30 PRINT "DISTANCE IN ONE TURN IS" 3.14*D
```

Did you make the change? Now list the program.

```
LIST
10 HOME
20 INPUT "WHEEL DIAMETER";D
30 PRINT "DISTANCE IN ONE TURN IS"; 3.14*D
40 PRINT
50 GO TO 20
```



Let's RUN it.

```
RUN
WHEEL DIAMETER? 20
DISTANCE IN ONE TURN IS 62.80

WHEEL DIAMETER? 26
DISTANCE IN ONE TURN IS 81.64

WHEEL DIAMETER? 27
DISTANCE IN ONE TURN IS 84.78

WHEEL DIAMETER ?
```

Apparently, the statement 30 PRINT "DISTANCE IN ONE TURN IS" 3.14*D tells the MPF-II to:

(a) print the string: DISTANCE IN ONE TURN IS
(b) compute and print the value of 3.14*D

## 6.9 For·····Next Loops

In this section, you will explore the power of the FOR-NEXT loop.

The FOR-NEXT loop takes over with automatic counting for all your repeatable tasks. You decide where to start and where to stop -- it does the rest.

After completing this section, you will be able to:

1. use the FOR-NEXT loop as a time delay;
2. perform mathematical feats with the FOR-NEXT loop.
3. use the FOR-NEXT loop for repeatable parts of a program (other than time delays and mathematical feats).

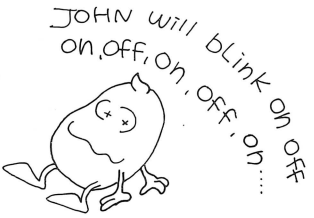
We will describe more features of the FOR-NEXT loop in chapter 6.

The following program will cause John's name to blink on, off, on, off, on...., and so on, until someone presses the CONTROL C key. Try it.

Type NEW, enter this program and RUN it.

```
100 HOME
110 PRINT "JOHN"
120 FOR I=1 TO 500        Clear the screen
130 NEXT I                Print the word JOHN
140 HOME
150 FOR I=1 TO 500
160 NEXT I
170 GOTO 100
```



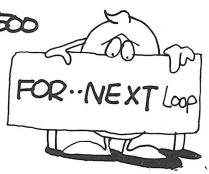




--Press CONTROL C to stop the computer.
--Change line 110 to: 110 PRINT"_____"
--RUN it again. Your name will blink on, off, on, off, on, off,.....

FOR ??? NEXT ??? What do they mean? In response to popular demand, we will explain what is happening in lines 120 and 130, and again in lines 150 and 160.

120 FOR I=1 TO 500
130 NEXT I

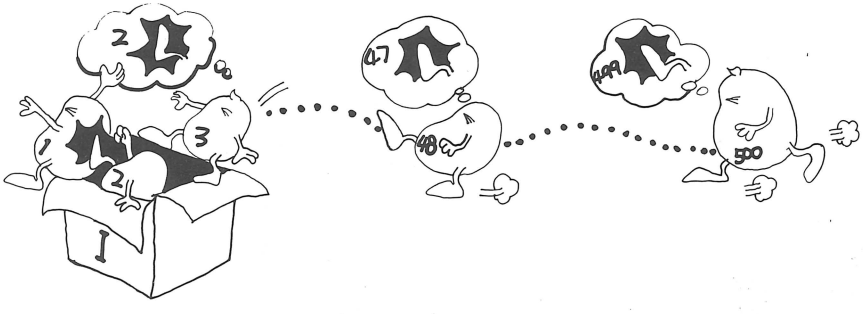



This is called a.....

The above FOR-NEXT loop simply tells the computer to count from 1 to 500.




As the computer counts from 1 to 500, each counting number is put, momentarily, into box I.




--First the computer puts 1 into box I.
--Then the computer erases 1 and puts 2 into box I.
--Then the computer erases 2 and puts 3 into box I.
--Then the computer erases 3 and puts 4 into box I.
--Then the computer erases 4 and puts 5 into box I.

.
.
.
.

--And so on. Eventually, the computer puts 500 into box I (Of course, it first erased 499.).

This all happens very quickly, in less than one second!

Lines 150 and 160 do the same thing. Both FOR-NEXT loops are simply time delays. The computer is counting silently to itself while we see JOHN on the screen or we see a completely blank screen. We see JOHN for a little while, then we see a blank screen for a little while, then we see JOHN for a little while, then we see a blank screen for a little while, then... and so it goes.

(a) How can we change lines 120 and 150 so that JOHN's name blinks more rapidly?

Use a number less than 500. For example, change line 120 to 120 FOR I = 1 TO 200 and line 150 to 150 FOR I = I TO 200. Now the MPF-II will count from 1 to 200, which takes less time than counting 1 to 500.

(b) How can we change lines 120 and 150 so that JOHN's name blinks less rapidly?

Use a number larger than 500. For example, change line 120 to 120 for I = 1 TO 1000 and line 150 to 150 FOR I = 1 TO 1000. Now the MPF-II will count from 1 to 1000, which takes more time than counting from 1 to 500.

```
120  FOR  I=1  TO 500
 .
 .
 .
150  FOR I= 1  TO 500
```




Let's make it easier to change the name.

```
100 HOME
110 INPUT "WHAT IS YOUR NAME";N$

120 HOME                          NAME IS ON
130 PRINT N$                      while MPF-II
140 FOR I = 1 TO 500              Counts to 500
150 NEXT I

160 HOME                          NAME IS OFF (screen is
170 FOR I = 1 TO 500              while MPF-II  clear)
180 NEXT I                        Counts to 500

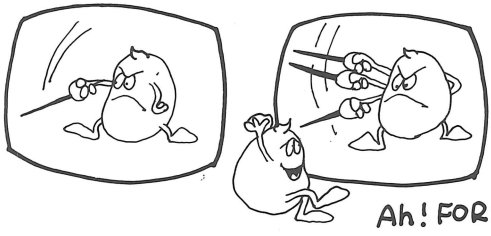
190 GOTO 120
```

(a) Suppose we change line 140 to: 140 FOR I = 1 TO 1000. Will the name be ON for a shorter time or a longer time?_____

Longer. It takes the computer about twice as long to count to 1,000 as it took to count to 500.

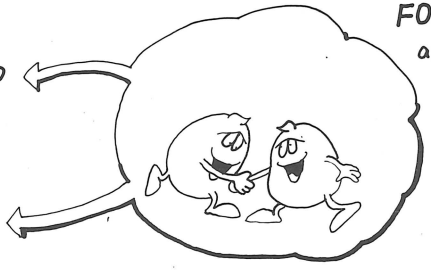
(b) Suppose we change line 170 to: 170 FOR I = 1 TO 250. Will the name be OFF for a shorter time or a longer time?_____

Shorter. It takes the computer about half as long to count to 250 as it took to count to 500.



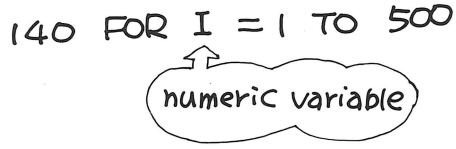

Follow the arrows to see how the program works.

```
RUN
 ↓
100  HOME
 ↓
110  INPUT" WHAT iS YOUR NAME"; N$
 ↓
120  HOME
 ↓
130  PRINT N$
 ↓
140  FOR I = 1 TO 500
150  NEXT I
 ↓
160  HOME
 ↓
170  FOR I = 1 TO 500
180  NEXT I
 ↓
190  GO TO 120
```

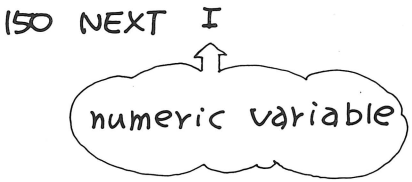


A FOR-NEXT Loop begins with a FOR statement and ends with a NEXT statement.

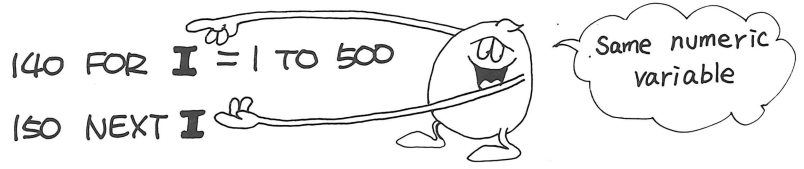
--A numeric variable must follow the word FOR:




--The same numeric variable follows the word NEXT:



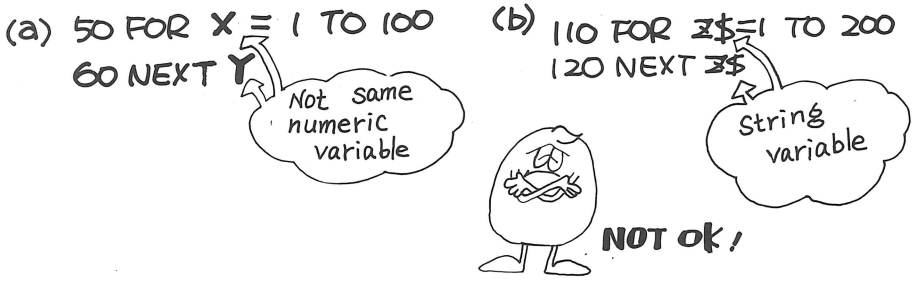
Here is another way to say it:




Let's not overwork the variable I. Any numeric variable may be used. --These are OK FOR-NEXT loops:

```
20 FOR K = 1 TO 3          73 FOR A = 1 TO 10
30 NEXT K                  89 NEXT A
```

--But these FOR-NEXT loops are not OK:




What is wrong with (a)?_____

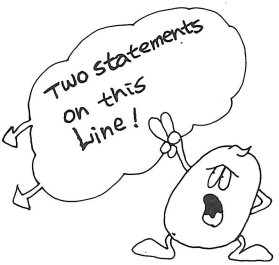
The variable following NEXT is different from the variable following FOR. These must be the same variable.

What is wrong with (b)?_____

Z$ is a string variable. We can use only numeric variables in FOR and NEXT statements.

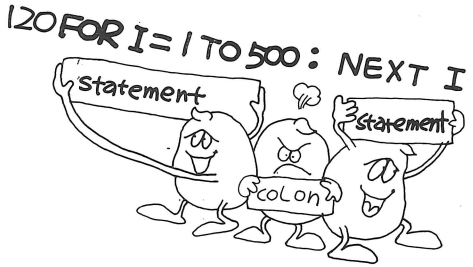
The following program features multiple statement per line.

```
100 HOME
110 PRINT"JOHN"
120 FOR I = 1 TO 500 : NEXT I
130 CLS
140 FOR I = 1 TO 500 : NEXT I
150 GOTO 110
```




In the above program, line 120 contains two statements. Line 140 also contains two statements. In a line that contains two statements, what symbol, or character, is used between the statements?_____

A colon (:)




We usually put a space on each side of the colon. However, this is not necessary. We could have typed line 120, as follows:

```
120 FOR I = 1 TO 500 : NEXT I
```

In fact, we can leave out all spaces and do it like this:

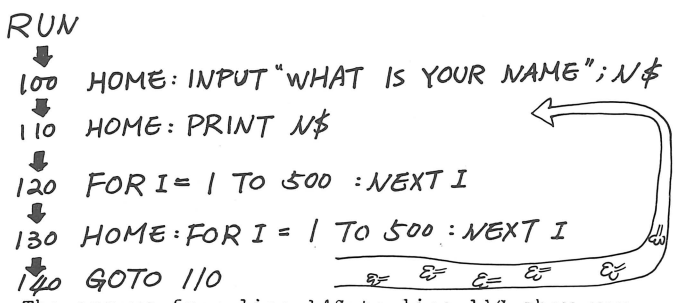
```
120 FOR I = 1 TO 500:NEXT I
```



The MPF-II will understand perfectly well, but it is hard for people to read! So, use spaces to make programs easier for humans to read. Here is another example, using multiple statements per line.

```
100 HOME: INPUT "WHAT IS YOUR NAME"; N$
110 HOME: PRINT N$
120 FOR I = 1 TO 500 : NEXT I
130 HOME: FOR I = 1 TO 500 :NEXT I
140 GOTO 110
```

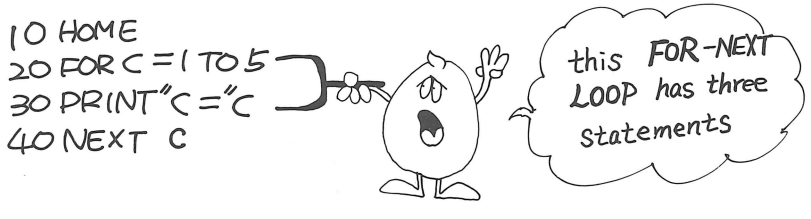
Follow the arrows.




The arrows from line 140 to line 110 show you that multiple statements in a line are done from left to right. In line 110, the HOME statement is done first, then the PRINT statement. In line 130, the HOME statement is done first, then the FOR-NEXT loop.

There is much about FOR-NEXT loops that we haven't told you. So, here we go.

A FOR-NEXT loop begins with a FOR statement and ends with a NEXT statement. A FOR-NEXT loop may also have one or more statements between the FOR statement and the NEXT statement.



In the above FOR-NEXT loop, what statement is between the FOR statement and the NEXT statement?_____

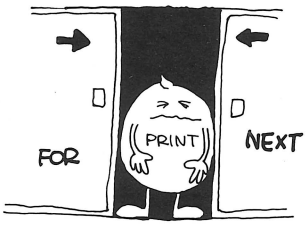
ANS : 30 PRINT "C="C

We could have written the above program like this:

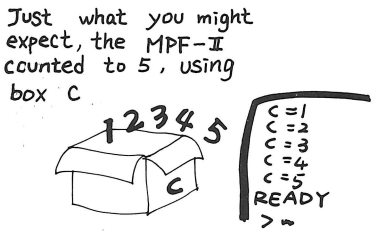
```
10 HOME
20 FOR C=1 TO 5 : PRINT "C="C:NEXT C
```

In the above FOR-NEXT loop (line 20), what statment is between the FOR statement and the NEXT statement?____

ANS : PRINT"C="C




What do you think will happen if we enter the program and RUN it? Here is what happened after we typed RUN.




Now, you are to be the computer. Show what will appear on the screen if we enter and RUN the following program.

The program(ours)

```
10 HOME
20 FOR N = 1 TO 3
30 PRINT N
40 NEXT N
50 PRINT "GO!"
```

The RUN (yours)

ANS:

```
1
2
3
GO!
READY
>
```

It is OK with us if you omitted these

A FOR-NEXT loop can consist of three things.
---A FOR statement
---A NEXT statement
---Statements between the FOR statement and the NEXT statement.
Of course, as you already know, the last item is optional. How does the FOR-NEXT loop work? Follow the arrows.

RUN

10 HOME

20 FOR N=1 TO 3

30 PRINT N

40 NEXT N

50 PRINT "GO!"







## Let's Look at the FOR statement

FOR N = 1 TO 3

This is the last value N will have (the limit of N).

This is the FOR-NEXT loop control variable.

This is the first value N will have.

In line 40, N is increased by one, and the computer compares the increased value of N to the upper limit for N indicated in the FOR statement (N < = 3 means "N less than or equal to 3").

When N goes past 3, the computer stops executing the loop and continues on with the rest of the program past the FOR-NEXT loop. We call this 3 the limit of N, or the limit of the FOR variable.

Each time the computer comes to a NEXT N statement, it increases the value of N by one, and checks the new value against the limit for N. In this case, the limit is 3, because the FOR statment reads: FOR N = 1 TO 3. When the value of N is greater than 3, the computer continues the statement following the NEXT statement, if there is one. If not, the computer has finished executing the program and stopped.

It's math time: You may find the next few programs interesting.

The FOR statement defines a sequence of values for its variable. "Sequence" is just a fancy math word. It means that the values come one after another.

Here is a table showing a FOR statement, the variable it uses, and the sequence of values which the variable assumes, one after another. Read the sequence of values from left to right.

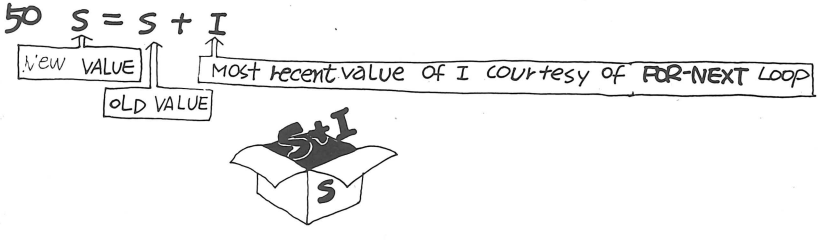
| FOR Statement | Variable | Sequence of values |
|---|---|---|
| FOR I = 1 TO 5 | I | 1, 2, 3, 4, 5 |
| FOR k = 3 TO 5 | k | 3, 4, 5 |
| FOR A = 0 TO 3 | A | 0, 1, 2, 3 |
| FOR N = 1 TO 1 | N | 1 |




The following program will compute the sum of the positive integers from 1 to N. Here is our program.

```
10  HOME
20  INPUT"N=";N
30  S=0
40  FOR I=1 TO N
50  S=S+I
60  NEXT I
70  PRINT"THE SUM IS";S
80  GO TO 20
```

The variable S (for sum) is used in lines 30, 50, and 70. Line 30 occurs before the FOR-NEXT loop. It sets the value of S to zero. Line 50 is inside the FOR-NEXT loop. It is executed for I=1, I=2, I=3, and so on, until I=N. Each time, a new value of S is computed.




Suppose the INPUT value of N is 3, then the sequence of value for I will be 1,2,3. So, S will become:

In Line 30 : S=0

In Line 50 : S=S+I=0+1=1

In Line 50 : S=S+I=1+2=3

In Line 50 : S=S+I=3+3=6 ← End of the FOR-NEXT LOOP this Value is printed by Line 70

Let's RUN the program. Here is a possible RUN.

```
N = ?3
THE SUM IS 6
N = ?4
THE SUM IS 10
N = ?7
THE SUM IS 28
N = ? AND SO ON
```

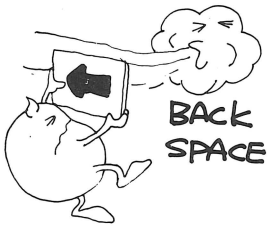


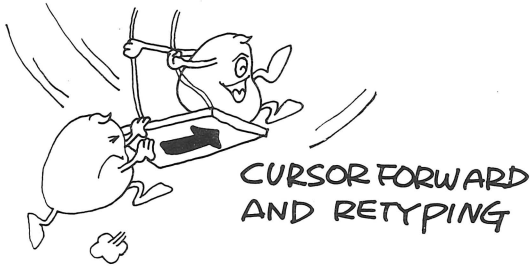

## 6 • 10 Advanced Editing Techniques

In chapter 2 we looked at ways you could correct errors in a line you are typing before you press the RETURN key. Let's quickly review those simple editing techniques.

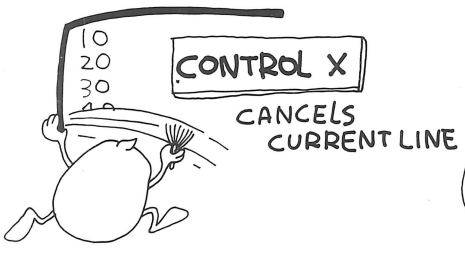
The <-- key backspaces the cursor and erases characters it passes over. Characters are erased from the program line even though they still appear on the display screen.

The --> key moves the cursor forward, copying over (retyping) characters it passes over.




The CONTROL X command cancels the line you're currently typing. The HOME command clears the display screen and leaves the cursor in the upper left corner.




We will now discover new ways to edit program lines. These new methods are particularly useful when you want to make changes to previous lines (i.e., those with line numbers).

**Deleting Program lines**

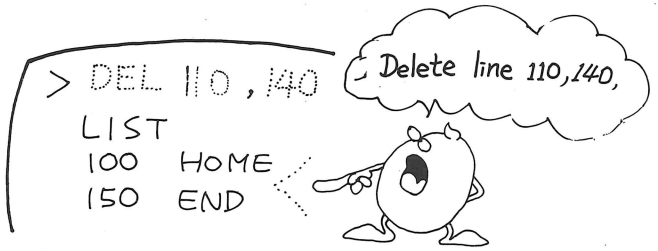
To delete an entire line, type its line number followed immediately by a RETURN. When you list the program, you will see that the line and line number are no longer part of the program. Here is an example:

```
NEW
100 HOME
110 FOR  N=1 TO 10
120 PRINT N, SQR(N)
130 N=N+1
140 NEXT N
150 END
```

```
100 HOME
110 FOR N=1  TO 10
120 PRINT  N, SQR(N)
140 NEXT   N
```




You can use the command DEL to delete a block of program lines. For example:




The command DEL 110, 140 deletes all program lines starting at line number 110 and ending with line number 140. Even though line 110 does not exist, all lines between 110 and 140 are deleted.

**Adding Program lines**

You can type in new program lines in any order, at any time. Their line numbers will determine their positions in the program. The MPF-II will automatically merge them into any other program lines currently in memory. Try adding lines 120 and 110 back into the example above.

**Moving the Cursor**

To move the cursor around on the screen, you can use the following four keys: --> <-- ↑ ↓. Figure -4 illustrates how the four possible key sequences affect cursor movement.

Now press -->, <--, ↑, ↓ to move the cursor around on the display screen.

**Changing Characters**

Replacing one character with the other is simplicity itself. Merely position the cursor on the offending character and type the replacement right over it. For example, with the cursor as shown:

```
100 PRINT "ESTIMATED TIME OF ARRIVAL"
```

You can type the word DEPARTURE and get this:

```
100 PRINT "ESTIMATED TIME OF DEPARTURE"
```

Press RETURN to effect the change.

**Deleting Characters**

You can effectively delete individual characters by typing over them with blank spaces. Remember that in BASIC extra blank spaces do not affect anything unless they are inside of quotation marks.

## 6 • 11 Remarks

Remark statements have line numbers, like any other statement. A remark statement's line number can be used like that of any other statement's.

If the first three characters of a BASIC statement are REM, then the computer ignores the statement entirely. So why include such a statement? The answer is that remarks make your program easier to read.

REM the computer ignores the statement entirely

If you write a short program with five or ten statements, you will probably have little trouble remembering what the program does --- unless you leave it around for six months and then try to use it again. If you write a longer program with 100 or 200 statements, then you are quite likely to forget something very important about the program the very next time you use it. After you have written dozens of programs, you will stand no chance of remembering each program in detail. The solution to this problem is to document your program by including remarks that describe what is going on.

Good programmers use plenty of remarks in all of their programs. In all of this chapter's program examples we will include remarks that describe what is going on, simply to get you into the habit of doing the same thing yourself.

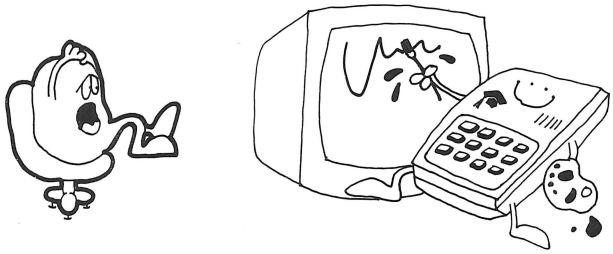
```
90 REM  INITIALIZE VARIABLE X
100 LET  X=8
```



## 6 • 12 Plotting and Developing in Color

MPF-II has the function of plotting and developing in color. But, of course, if you want it to be developed in color, the screen or TV set you use must be a color one.

REM ignores the statement entirely



These are some of the instructions that are used when writing plotting or color programs.

HOME, COLOR, PLOT

Let's explain them with some example programs:

### 1. Home

HOME is the instruction which calls prompt and cursor back home -- the starting point, and at the same time, clears the screen. After we press PRINT HOME and RETURN, the screen will be cleared. After you examine the following programs you will understand.

```
10 HOME
20 PRINT"MR SHIH"
30 GOTO 10
```

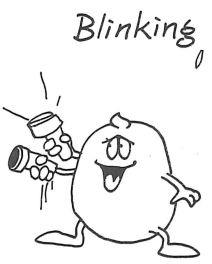
Because it returns from line 1Ø to line 3Ø, MR. SHIH, which appears on the screen, flashes. Let us remind you that if you want to stop performing this program, you have to press CONTROL C or RESET.

Try the following example: The new program on the left shows little difference from the previous one and yet the results of its performances are quite different.

```
10 HOME
20 PRINT "MR. SHIH"
30 GOTO 20
```

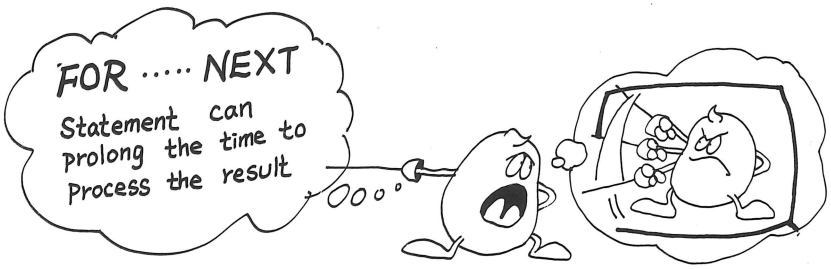
```
10 HOME
20 PRINT "MR. SHIH"
30 GOTO 10
```




The program on the right hand side is the original one, and line 3Ø is GOTO 1Ø; the program on the left hand side is the new one and line 3Ø is GOTO 2Ø. Because the line 3Ø of the original program is GOTO 1Ø, MR. SHIH which is just printed is cleared out. The result of the printing and clearing makes the screen flash. But the line 3Ø of the left hand side program is GOTO 2Ø which indicates the program to keep on printing. In order to give you a good understanding of the way to use HOME, let's combine HOME and FOR.... NEXT. After examining the following program, put it in the computer and let it run.

```
10 HOME
20 PRINT "MR.SHIH"
30 FOR I = 1 TO 500
40 NEXT I
50 HOME
55 FOR I= 1 TO 500
60 NEXT I
70 GOTO 10
```

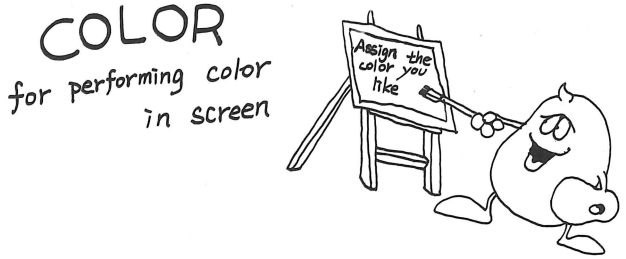



Now, you know that FOR.....NEXT can be used to prolong the duration. Take the above example for example. If you want to change the time duration, you only have to change the number of FOR.....NEXT.

```
130 FOR I=1 TO 500
155 FOR I =1 TO 500
```

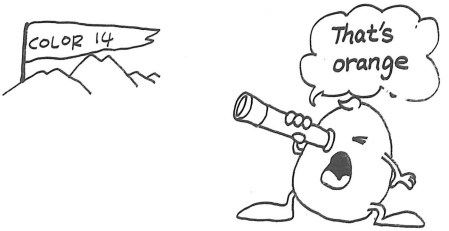
**2. Color**

In MPF-II, there is a color instruction. The instruction COLOR directs the computer to perform color functions. (If your TV set or screen is black-and-white, this instruction will be useless.)



| | | | |
|---|---|---|---|
| 0 Black | 4 Green | 8 Purple | 12 White |
| 1 Green | 5 Orange | 9 Green | 13 Orange |
| 2 Purple | 6 Blue | 10 Purple | 14 Blue |
| 3 White | 7 White | 11 White | 15 white |

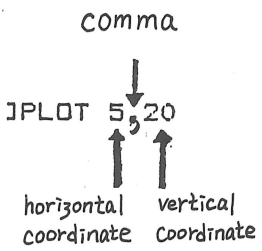
When using a microcomputer for color graphics, only in rare occasions a user needs to use more than 10 colors. Thus, the MPF-II is so designed that a maximum of six colors can be used for color graphics. The numbers given in the chart below is used to identify different colors.

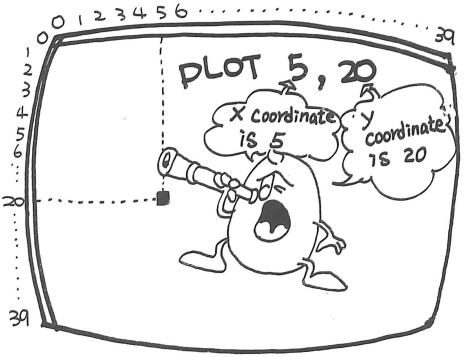



But the color instruction should be used with PLOT instructions. Let's proceed.

**3. Plot**

Using the MPF-II for plotting means the assignment of horizontal and vertical coordinates and then placing a dot at the coordinates. The MPF-II divides the screen up into 40 lines (vertical coordinate) and 40 columns (horizontal coordinate). Coordinates are numbered from 0 to 39. Plot must be followed by two arguments separated by a comma. For example:

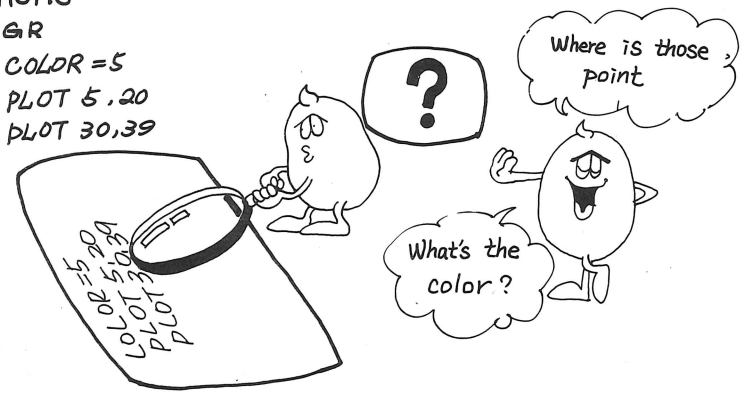


The left number shows its horizontal coordinate; the right number shows its vertical coordinate. When performing PLOT 5,20, as shown in the picture, there will be a coordinate dot which appears on the screen.

All right! Now let's put together the three instructions: HOME, COLOR and PLOT.

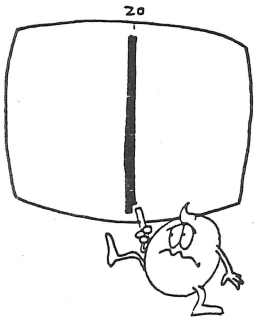
```
10 HOME
20 GR
30 COLOR=5
40 PLOT 5,20
50 PLOT 30,39
```

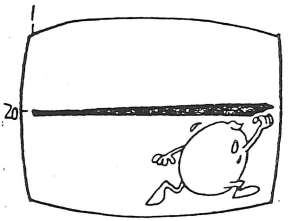



In the following two programs, we can draw a horizontal line to the left hand side program; a vertical line to the right hand side program.

```
10 HOME
20 GR
30 COLOR=3
40 FOR I=0 TO 39
50 PLOT 20, I
60 NEXT I
```

```
10 HOME
20 GR
30 COLOR=14
40 FOR I=0 TO 39
50 PLOT I,20
60 NEXT I
```



There are still two instructions which can be used when the MPF-II draws a line. They are: HLIN and VLIN. H is the abbreviation of horizontal; V vertical; and LIN line.

Now, let's have the following example.

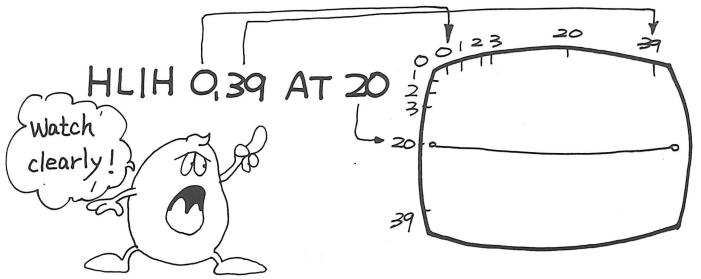
**4. HLIN and VLIN**

a. The results of the two programs are the same. Please compare them.

```
10 HOME
20 GR
30 COLOR=14
40 FOR I=0 TO 39
50 PLOT I,20
60 NEXT I
```

```
10 HOME
20 GR
30 COLOR=14
40 HLIN 0,39 AT 20
```

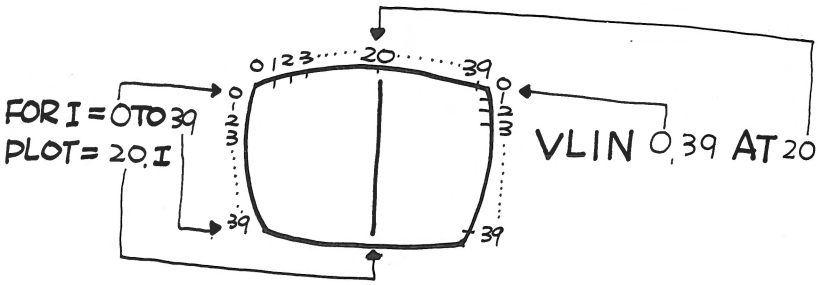
HLIN 0,39 AT 20 tells the MPF-II to draw a horizontal line from 0 to 39 at vertical coordinate 20.




b. The results of the following two programs are also the same. They are all used to draw vertical lines. Please compare them.

```
10  HOME
20  GR
30  COLOR = 14
40  FOR I = 0 TO 39
50  PLOT 20, I
60  NEXT I
```

```
10  HOME
20  GR
30  COLOR = 14
40  VLIN 0, 39 AT 20
```

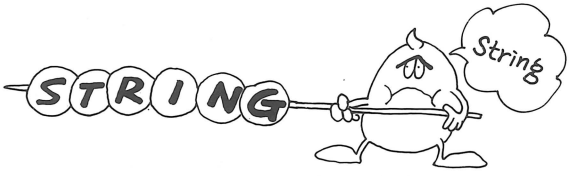


## 6 • 13 Handling String

### 1. String

The computer's major function is to handle data. The two classes of data are numeral and character. We have discussed numeral data. Now let's proceed to talk about alphabetical data handling.

We have to combine a series of letters (characters) together to make them meaningful. This combining process is called stringing, or more soundly we refer to combined characters as a string.




When using BASIC and many other computers, the string needs to be set off with quotes. For example:

```
]PRINT"MR. SHIH"
```

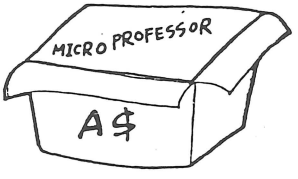
the strings need to be quoted

Mr. SHIH in the quotation marks is a string. As with a numerical variable, a string can be assigned with to a variable. But we must add an "$" after each string variable.




For example, if we key in the following program:

```
10 A$ = "MICRO PROFESSOR"
20  PRINT A$
```



When computer runs the program, it will print:

```
MICRO PROFESSOR
```
← This is the string in the quotation mark.

In this example, MICRO PROFESSOR is put in the A$ location in memory.

**2. LEN**

How many characters are there in a string? You will need a method of computing the length of a string. So in the BASIC language, there is an instruction LEN to solve this problem. LEN is an abbreviation for length.

```
]PRINT LEN("YES")
3
```
← The length of string is 3

```
]A$="MPF-2"

]PRINT LEN(A$)
5
```
← The length of string is 5

Try the following program:

```
10 A$ = "MPF-II"
20  PRINT  LEN (A$)
30  PRINT  LEN ("MPF-II")
```
← Computer will print out 6.

← Computer will print out 6.



LEN is length

In line 1Ø, MPF-II is put in A$.

In line 2Ø, the length of A$ is printed out (it is 6 in this example).

In line 3Ø, the length of MPF-II is also 6.

Note the differences between line 2Ø and line 3Ø.

### 3. LEFT$, RIGHT$, and MID$

Suppose we only want to use some parts in a string, we can use these instructions:

a. LEFT$

For example:

```
]A$="HOW ARE YOU"
```

Suppose we only want to print the front three characters in A$, we can put in:

```
]PRINT LEFT$(A$,3)
```

The computer will print:




The instruction LEFT$(A$, 3) tells the computer to print three characters on the left side in string A$. Don't forget that there is a "$" after LEFT.

All right! Let's run the following program:

```
5  HOME
10 A$ = "HOW ARE YOU"
20  FOR I = 1 TO  LEN (A$)
30  PRINT  LEFT$ (A$,I)
40  NEXT I
```

clear the screen

perform I from 1 to the length of A$

print the left I's alphabets

The following is the result of the execution. Study the results and the explanation below:

```
H
HO
HOW
HOW
HOW A
HOW AR
HOW ARE
HOW ARE
HOW ARE Y
HOW ARE YO
HOW ARE YOU
```

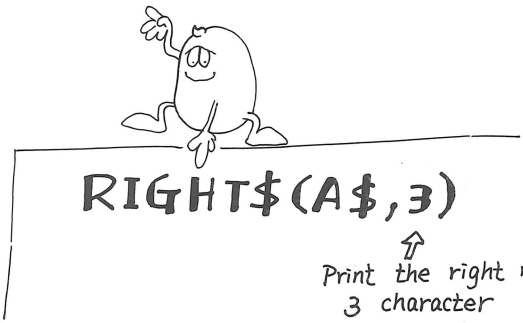
a. There are altogether eleven characters in the string "HOW ARE YOU", including the space.
b. First, print the first character "H" on the left side.
c. Second, print the second character "O" on the left side.
d. Third, print the third character "W" on the left side.

Print all the eleven characters.

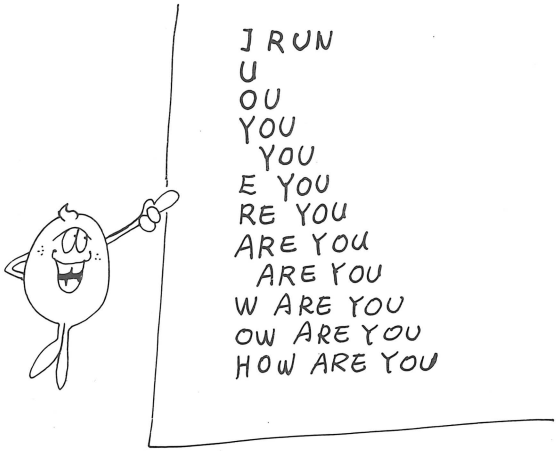
b. RIGHT$

RIGHT$ (A$, 3) tells the computer to print the three characters on the right in string A$. Don't forget that there is a "$" after RIGHT. The following program, only differs from the previous program line 30.

```
5  HOME
10 A$ = "HOW ARE YOU
20  FOR I = 1 TO  LEN (A$)
30  PRINT  RIGHT$ (A$, I)
40  NEXT I
```




The following is the result of the execution.



Explanation:

a. There are altogether eleven characters in the string "HOW ARE YOU", including the space.
b. First, print the first character "U" on the right side.
c. Secondly, print the second character "O" on the right side of "U".

:

Finally print all the eleven characters.

c. MID$

MID is the abbreviation of middle. MID$ means that we print out some characters in the middle part. For example: MID$ (A$3, 5) tells the computer to print out five characters in serial starting at the third character.

```
]PRINT MID$(" HOW",2,2)
```

Press [↵], the computer will print out two characters.

OW ← Printing

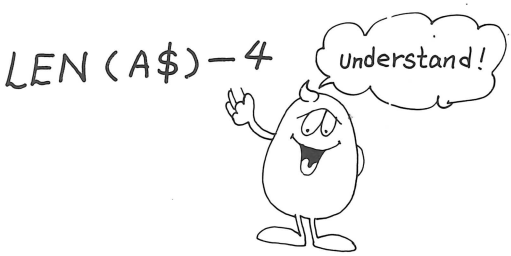
Let's try the following program and see its result:

```
5  HOME
10 A$ = " HOW ARE YOU "
20  FOR I = I TO  LEN (A$) - 4
30  PRINT  MID$ (A$,5,I)
40  NEXT I

]RUN
```

← This statement is the same with "FOR I = 1 TO 7"

```
A
AR
ARE
ARE
ARE Y
ARE YO
ARE YOU
```




All right! Let's try the following example and after this, it will be the end of this section.

```
        PROGRAM                                  RESULT

5  HOME
10 A$ = "MICRO PROFESSOR"                        ]RUN
20  PRINT A$          <------------------------> MICRO PROFESSOR
30  PRINT  LEN (A$)   <------------------------> 15
40  PRINT  LEFT$ (A$,3)  <--------------------->  MIC
50  PRINT  RIGHT$ (A$,3) <--------------------->  SOR
60  PRINT  MID$ (A$,3,6) <--------------------->  CRO PR
```

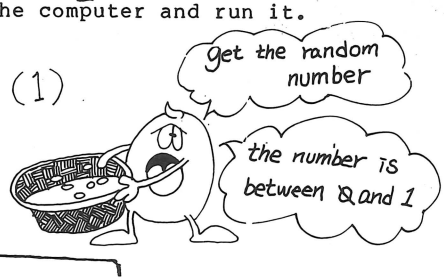
## 6 • 14 Random

In BASIC language, there is a RND function. RND is the abbreviation of random. In BASIC language, RND means applying RND.




### 1. RND

a. In MPF-II, RND(1) means that the computer chooses for you a random number where Ø ≤ number < 1. Put the following program in the computer and run it.




```
5  HOME
10  FOR I = 1 TO 10
20  PRINT  RND (1),
30  NEXT I
```

The following is the result of running the program. Please note that if you keep on running for several times, the results will be different. You can test whether these numbers are in the range of Ø ≤ number < 1.

```
]RUN
.131137465          .80924873
.846447204          .841536558
.591965711          .26800113
.419217095          .878831482
.368373372          .123235316
```

b. If you want a number where number is $0 \leq$ number $< 10$, you can have the number by multipling by 10. That is, change line 20.

```
120 PRINT RND(1),

120 PRINT 10*RND(1),
```




The following is the result of the change.

```
5  HOME
10  FOR I = 1 TO 10
20  PRINT 10 *  RND (1),
30  NEXT I
```

The following are the results of a sample execution. You've known that the result of each execution will not be the same. You can execute the program again to see whether the results are in the reange of $0 \leq$ number $< 10$.

```
]RUN
2.80111176      8.36328912
1.15622079      8.15110127
5.73253785      8.89939655
4.27020051      .226942065
1.23590262      9.8134051
```

Will the result of 6 * RND(1) be in the range of $0 \leq$ number $< 10$? Try the following program:

```
5  HOME
10  FOR I = 1 TO 10
20  PRINT 6 *  RND (1)
30  NEXT I
```

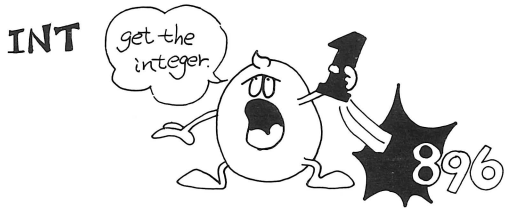
get the random number

enlarge six times

## 2. INT

As a matter of fact, we often want to generate random integers. When casting dice, we want an integer from 1 to 6 and sometimes we want to get an integer in the range from 1 to 10.

a. INT is the abbreviation of integer.




For example:

```
PRINT INT (1.896)   ---> 1
PRINT INT (3.254)   ---> 3
PRINT INT (Ø.999)   ---> Ø
```

All right! Let's try the following program.

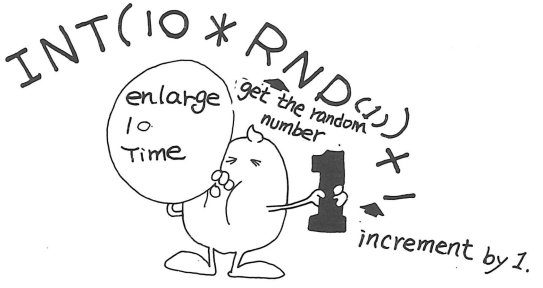
```
5  HOME
10  FOR I = 1 TO 10
20  PRINT  INT (10 *  RND (1)),
30  NEXT I
```

The following is one of the results. Of course, the result of each execution will not to be the same, but the answer is always between Ø and 9.

```
]RUN
9                7                9
4                6                9
3                8                3
5
```

b. If you want to get a number from 0 to 10, please try the following program:

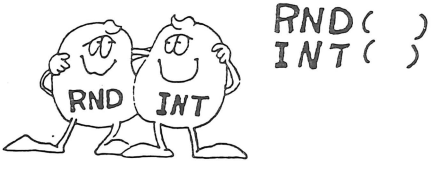
```
5  HOME
10  FOR I = 1 TO 10
20  PRINT  INT (10 *  RND (1)) +
     1,
30  NEXT I
```




The results of the following executions are always between 1 and 10.

```
]RUN
2                7                4
9                1                6
8                5                9
7
```

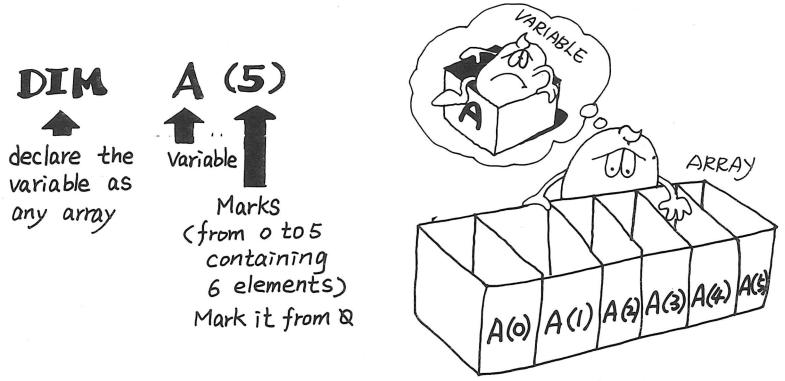
The joining together of RND and INT is very useful, you should make good use of it.



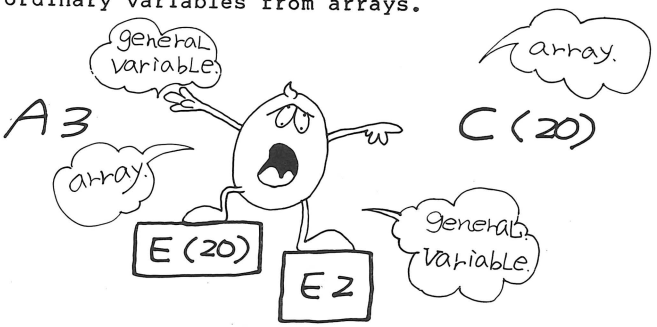
## 6 • 15 Array DIM

We can regard variables as the memory for storing data, but a variable can only store one special data. An array is the variable that can store many data, but this variable should be additionally marked. When using the array, you have to use DIM instruction first.

DIM is the abbreviation of dimension. The size of an error is indicated by a subscript. A subscript is the number associated with a particular character. For example, in A(2). "2" is the subscript of A. Let's look at the following picture and then you will be able to understand array, DIM and subscript.




The instruction DIM A(5) indicates that the variable A is not an ordinary variable but an array. The array consists of 6 memory boxes, A(Ø), A(1), A(2), A(3), A(4), A(5). In general, a microcomputer starts numbering from Ø. So, DIM B(15) tells B is an array, and there are 16 memory boxes(location), the subscript is from Ø to 15. There are altogether 16 subscripts ones. Distinguish the following ordinary variables from arrays.



All right! Let's try a simple program to see if you can understand it.

```
5  HOME
10  DIM A(12)
20  FOR I = 0 TO 10 STEP 2
30 A(I) = 3 * I
40  PRINT "A("I")="A(I),
50  NEXT I
```

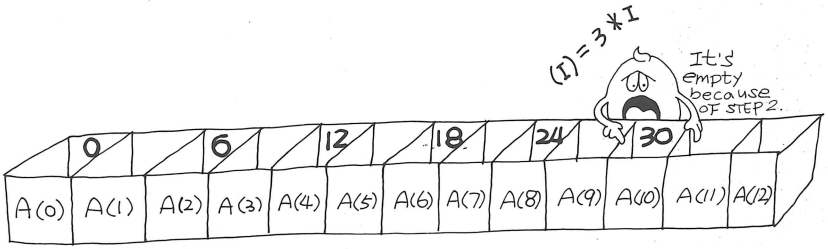
Declare A as an array in the range 0 TO 12 (containing 13 marks)

Put the result of 3*I into A(I)

Print the contain of A(I)

The following are the results of the execution. Do you understand the results?

```
]RUN
A(0)=0          A(2)=6          A(4)=12
A(6)=18         A(8)=24         A(10)=30
```

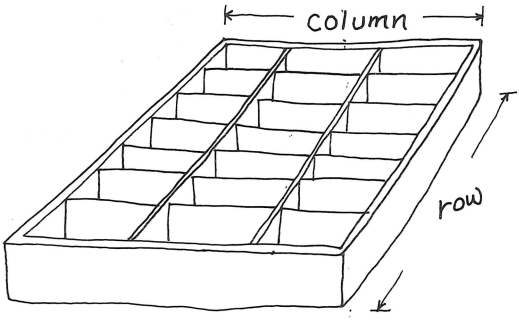



The following is another program and its result. Are the results meaningful?

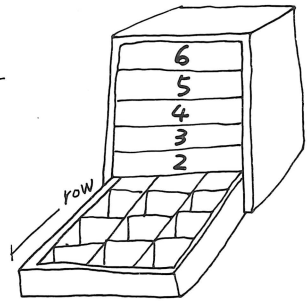
```
5  HOME
10  DIM A(12)
20  FOR I = 0 TO 12
30 A(I) = I * 2
40  NEXT I
50  FOR I = 0 TO 12
60  PRINT A(I),
70  NEXT I
```

```
JRUN
0           2           4
6           8           10
12          14          16
18          20          22
24
```

In BASIC language, there are two-dimensional arrays and three-dimensional arrays. They are shown below.

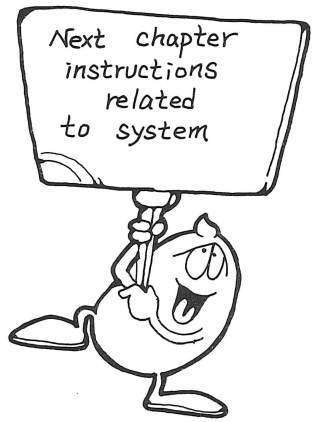



Two dimentional array.




Three dimentional array.

We will discuss DIM later. This is the end of Chapter 6. Starting from the next chapter, there will be a detailed and systematic introduction to DIMs.



# 7

# COMMANDS RELATING TO THE FLOW OF CONTROL

## 7 • 1 Instruction Analysis Related to System

Every computer needs some instructions to tell the computer what to do. If we analyze instructions systematically, we may generally divide them into three groups:

1. Instructions related to execution: For example, to tell computer to RUN, to STOP, or to erase a program (NEW) etc.



RUN NEW STOP····etc.

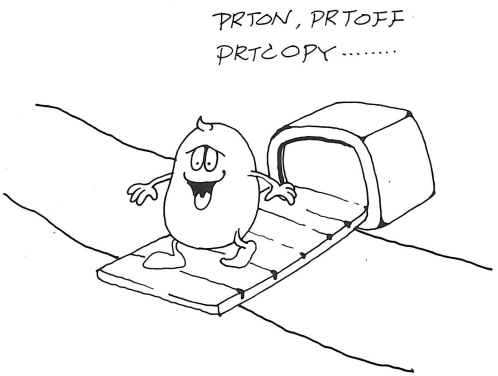
2. Instructions related to editing and formatting: Editing refers to the process by which we put the program into the computer through keyboard. In editing, perhaps we want to omit some unnecessary statements, then we may use instruction DEL. DEL means to delete, or eliminate, this is one of the editing instructions. Sometimes, we have to clear the screen before we write another program, then we may use instruction HOME. Of course, there are many other instructions, such as LIST etc., which will be discussed later in this chapter.

DEL HOME LIST



3. Instructions related to the communications with devices: For example, to tell computer to print data or output through the printer etc. PRTON and PRTOFF are such kind of instructions.



## 7 • 2 Instructions Related to Execution

Seven instructional groups are introduced to you in this section:

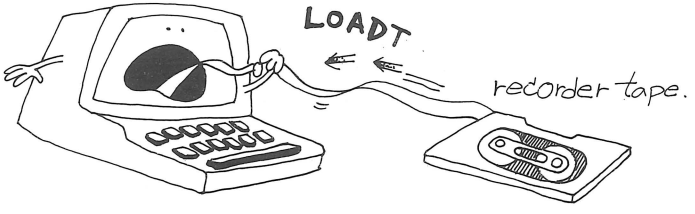
### 1. LOAD and SAVE

These two instructions are useful only when the computer is equipped with a tape recorder.

Cassette commands into cassette tapes. For commands are used when storing data or program on cassette tape or when loading program from cassette recorder onto a microcomputer.

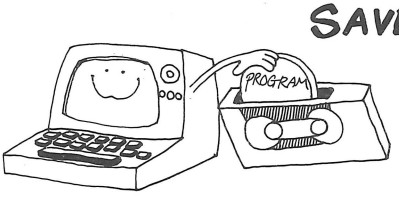
a. LOADT : Load a file whose filename is ABC from cassette to the MPF-II.

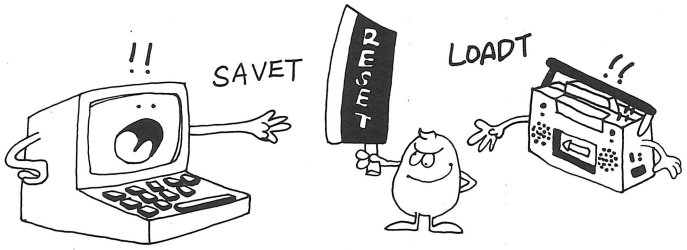
b. LOADA : Load programs from cassette tape of Apple II cassette format onto the MPF-II.




c. SAVET : Store the file whose filename is ABC from the MPF-II to cassette tape.

d. SAVEA : Store program or data on cassette tape of Apple II format.






**2. NEW**

Suppose you want to clear out completely all the BASIC programs and variables in the computer, you have to use NEW.

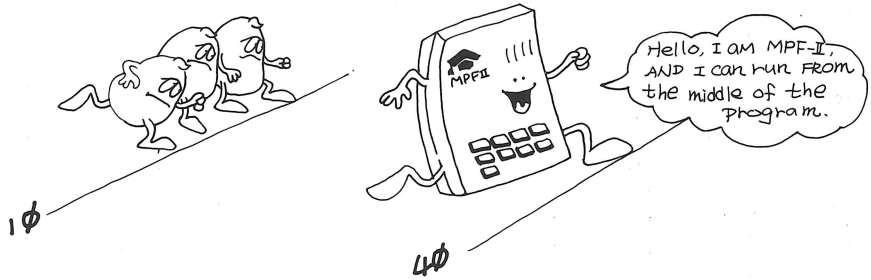
**3. RUN**

a. RUN means to run programs, but in MPF-II, you may start from any line number while you are running programs.

```
5  HOME
10  FOR I = 1 TO 10
20  PRINT I;
30  NEXT I
40  FOR I = 1 TO 10 STEP 2
50  PRINT I,
60  NEXT I

]RUN 40
1                3                5
7                9
```

FROM LINE NUMBER 40 TO RUN ONLY EXECUTE FROM 40 TO 60, PRINT OUT 1, 3, 5, 7, 9



b. If the line number you gave to start execution (RUN) does not exist in the program, the computer will tell you that you are wrong and that there is no such a line number (?UNDEF'D STATEMENT ERROR). The symbol ? means that the computer does not understand, UNDEF'D STATEMENT = undefined statement. In the above example, if you run 45, and there is no line number 45 in the above program, the computer will tell you that you are wrong.

```
]RUN 45

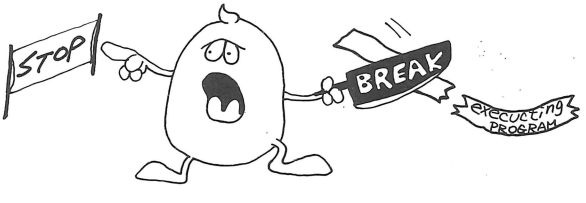
?UNDEF'D STATEMENT ERROR
```

**4. STOP, END and CONT**

a. We have known that both STOP and END can interrupt the program which is under execution. But, STOP is not necessarily placed at the end, there can be several STOPs in one program, if necessary. When the computer execute a STOP, it will display the signal BREAK IN XX. Let's take a look at the next example, in this program, there is a STOP in line number 40, so there is the signal BREAK IN 40.

```
10  FOR I = 1 TO 10
20  PRINT I;
30  NEXT I
40  STOP

]RUN
12345678910
BREAK IN 40
```



b. Suppose program is interrupted by a STOP command, we can go on executing with CONT. CONT is the abbreviation of continue. Let's take a look at the following program.

```
10  FOR I = 1 TO 2
20  PRINT I;
30  NEXT I
40  STOP          <-------- STOP HERE
50  FOR I = 1 TO 5
60  PRINT I;
70  NEXT I
```

While first executing the program, we run until statement 40. The computer stops and prints out BREAK IN 40, because there is a STOP.

```
]RUN
12
BREAK IN 40  <-------- STOP AT LINE 40
```

When we press CONT, the program will go on executing statement 50-70.

```
]CONT  <-------- USING "CONT", NOT "RUN"
12345
```




c. If a program uses END, the computer will not give any signal.

```
10  FOR I = 1 TO 10
20  PRINT I;
30  NEXT I
40  END

]RUN
12345678910
```

But, after END we can also use CONT to go on, please try the following example:

```
10  FOR I = 1 TO 10
20  PRINT I;
30  NEXT I
40  END
50  FOR I = 1 TO 10
60  PRINT I,
70  NEXT I

]RUN
12345678910
]CONT
1               2               3
4               5               6
7               8               9
10
```

**5. CONTROL C and RESET**

a. The action of CONTROL C and STOP is the same, but CONTROL C is entered by the operator to stop the program execution at any point. For instance, when we are performing the following program, the computer will go on printing MR. SHIH. If we want to stop it, we may press CONTROL C, but the computer will also print out BREAK.

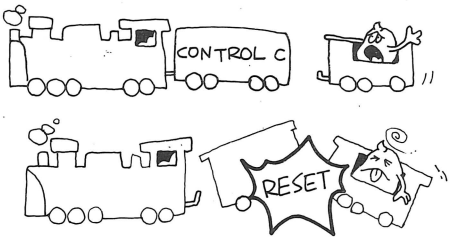
```
10 PRINT "MR. SHIH"
20 GOTO 10

JRUN
MR·SHIH
MR·SHIH
MR·SHIH
MR·SHIH
MR·SHIH
BREAK IN 10 <------BREAK
```

b. When we press RESET, we can also interrupt the execution of the program. But the program will not disappear, and there will not be the signal BREAK.

c. Suppose we use CONTROL C to interrupt the program, we may use CONT to go on. While we use RESET, we have to run from the beginning.

CONTROL C to interrupt the program, CONT to go on.




If you use control C to interrupt the program

RESET is use to interrupt, have to start again.

RESET to interrupt, RUN to execute again.

**6. TRACE and NOTRACE**

a. If you want to follow the process of execution of a program, you may use TRACE. Before you run, you first press TRACE and RETURN. Consequently, when the program is executing, the computer will show you the line number of each statement as it is executed. Here is an example:

```
10  FOR I = 1 TO 2
20  PRINT I;
30  NEXT I

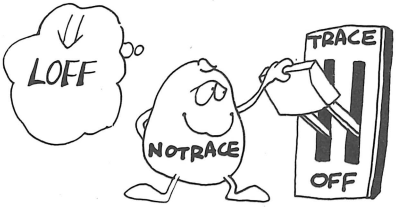
]TRACE  <-------- FIRST TYPE TRACE

]RUN    <-------- THEN TYPE RUN
#10 #20 1#30 #20 2#30
 ↑   ↑   ↑   ↑   ↑   ↑   ↑
LINE NO. LINE NO. PRINT LINE NO. LINE PRINT LINE NO.
 10      20       1     30       NO.20 2     30
```

b. The main function of TRACE is to help a programmer debug, TRACE functions as an aid which helps correct any errors in a computer program.




c. NOTRACE is just contrary to TRACE. When you don't want to use TRACE, you may press NOTRACE, the activity of TRACE will be ended then.
d. Once machine has been put in the TRACE mode, only NOTRACE can end the activity of TRACE, while NEW and RESET cannot.




NOTRACE TO CLOSE THE TRACE command.

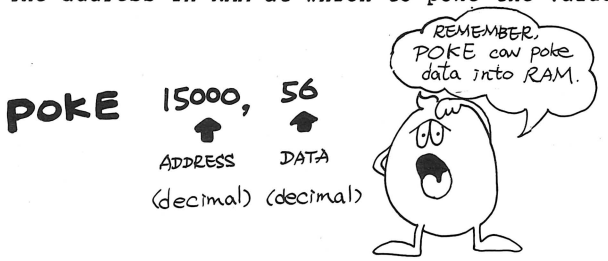
## 7. POKE and PEEK

a. POKE and PEEK are the two instructions which require more background to use than other BASIC statements. Suppose you want to use these two instructions efficiently, you have to understand the memory assignments of RAM and ROM in computer, therefore, we suggest that you comprehend the hardware structure of microcomputer first before you use POKE or PEEK.

b. POKE means to put data directly into the computer's memory; mind you, you can only poke data into RAM, but not ROM, because the data in ROM cannot be changed. When you poke, you have to indicate two things:

1. What to POKE.

2. The address in RAM at which to poke the value.




POKE 15000,56 means to put 56 at address 15000. Here, in order to make it easy for us to use the computer, both 56 and 15000 are decimal numbers. But the decimal numbers will be changed into binary numbers in computer in order to store the data.

c. The data of microcomputer is stored in a byte (8 bits), so the data you put into must be a number between 0 and 255. If you try to POKE a number outside the range 0 to 255, the computer will tell you:

ILLEGAL QUANTITY ERROR. Here is an example:

```
10  POKE 15000,259
]RUN
?ILLEGAL QUANTITY ERROR IN 10
```

data exceed 255

d. The address of POKE must be a RAM address, the maximum address of the MPF-II is 65535. If you go beyond this number, the computer will also tell you: ILLEGAL QUANTITY ERROR, but this time address is wrong.

address exceed 65535

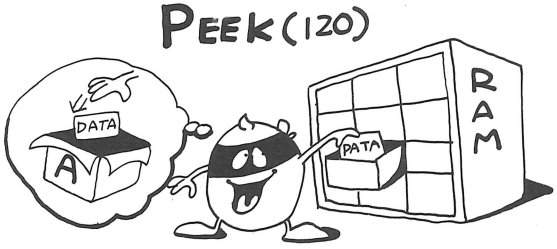
```
10  POKE 70000,235
]RUN
?ILLEGAL QUANTITY ERROR IN 10
```

e. Suppose you want to access the data at a particular address directly, you may use the instruction PEEK, and you only have to give the computer the address. For example,

```
10 A =  PEEK (120)
20  PRINT A
```

Put the data of address 120 into variable A

PRINT out A



```
10  FOR I = 0 TO 10
20 A =  PEEK (I) ←
30  PRINT A,
40  NEXT I

]RUN
76          212          212
76          50           219
255         255          255
255         76
```

Result

Data Stored in address 0

data stored in address 9

Data in address 10

data in address 5

Now, let's try the combined function of PEEK and POKE. Try this:

```
10  POKE 15000,56 ←
20 A =  PEEK (15000) ←
30  PRINT A ←

]RUN
56 ←
```

Poke 56 to the address 15000

Move the data in 15000 to variable A

Print Value of A

Result

```
10 K = 0 ←
20  FOR I = 15000 TO 15005
30  POKE I,K
40 K = K + 1
50  NEXT I ←
60 K = 0 ←
70  FOR I = 15000 TO 15005
80 A(K) =  PEEK (I)
90 K = K + 1
100  NEXT I ←
110  FOR I = 0 TO 5 ←
120  PRINT "A("I")=";A(I),
130  NEXT I ←
```

Poke the data 0~5 to the address 15000~15005

PEEK the data in address 15000~15005 to the array A(K)

Print out the result

## 7 • 3 Instructions Related to Editing and Format

Nine instructional groups are introduced to you in this section:

### 1. LIST

LIST as explained earlier means to list all parts of a program. But, MPF-II may also list part of a program. For instance, suppose we want to list part of a program. We may use:




```
5  DIM A(20)
10  FOR I = 1 TO 20
20 A(I) = I * 5 + 8
30  NEXT I
40  FOR I = 1 TO 20
50  PRINT A(I),
60  NEXT I

]LIST 20-40
```

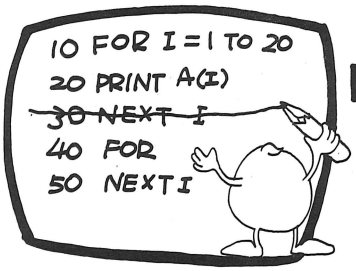
← Print out from line No. 20 to 40

```
20 A(I) = I * 5 + 8
30  NEXT I
40  FOR I = 1 TO 20
```

only 20~40

**2. DEL**

DEL is the abbreviation of delete. DEL means to remove or eliminate, e.g. to remove a record from a master file. Suppose you want to remove line number 30, you may use:




Suppose you want to remove line number, 50 to 80, you may use:




**3. REM**

REM is the abbreviation of remark. When the computer finds a statement REM, it will ingore it and not perform it. REM is needed to let others read easily the program or for documentation (reference) purposes. When a program is complicated, REM becomes rather important.

```
10  REM  PROGRAM 1
20  DIM A(20)
30  REM  ASSIGN A(I)
40  FOR I = 1 TO 20
50 A(I) = 5 * I + 2
60  NEXT I
70  REM  PRINT A(I)
80  FOR I = 1 TO 20
90  PRINT A(I),
100  NEXT I
```



**4. TAB, SPC and POC**

a. The TAB command is used to set the cursor horizontally. TAB can be used to start the printing of data at any required column. TAB is usually associated with PRINT.

IN NEXT EXAMPLE, PRINT OUT DATA 5 AT 10th POSITION

```
PRINT  TAB(10) ; 5
```

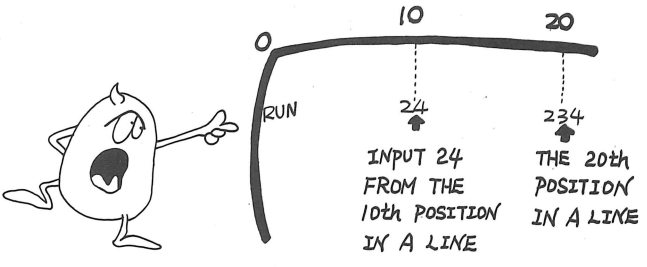
```
10  READ A,B
20  PRINT  TAB( 10);A, TAB( 20);B

30  DATA  24,234

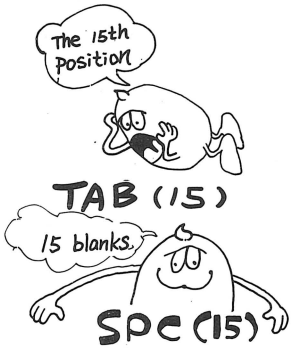
]RUN
          24          234
```

Print A from the 10th position

Print B from the 20th position




b. SPC is the abbreviation of space. SPC (20) means to move the cursor over 20 spaces horizontally. Let's study the following example and compare the differences between SPC and TAB.




```
10  PRINT "AAAA"; TAB( 15);"AAAAA
    A"
20  PRINT "BBBB"; SPC( 15);"BBBBB
    B"
```

15 Blanks

The 15 position

```
]RUN
AAAA          AAAAAA
BBBB                BBBBBB
```

15 Blanks

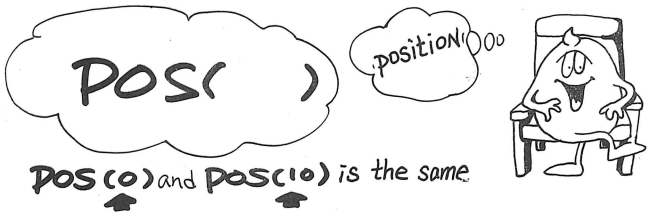
c. POS is the abbreviation of position. POS reports the position of the cursor. Usually, we have to add brackets behind POS, and the number in the brackets makes no difference, for example, POS (Ø) and POS (15) bear the same meaning. Let's try the following example, we may understand better the implication of POS (Ø).

```
10  PRINT "AAA"; POS (0)
20  PRINT "BBBB"; POS (12)
```

```
]RUN
AAA3
BBBB4
```

Position 3

Position 4



Now, let's apply the combined function of TAB, SPC and POS. Try the following two programs.

```
]10 PRINT TAB(23);POS(0)

]20 PRINT SPC(23);POS(0)

]RUN
 22
 23
```

from 23th position

from 23th position

```
10  PRINT "AAA"; TAB( 10);"AAAA";
      POS (0)
20  PRINT "BBB"; SPC( 10);"BBBB";
      POS (0)

]RUN
AAA       AAAA13
BBB            BBBB17
```

from 13th position

from the 17th position

Think about it!



**5. HOME, NEW and CLEAR**

a. HOME

Instruction HOME tells the computer to clear the screen and to remove the cursor to the beginning of the screen. Remember, although screen is cleared, the program within the memory is still there.

b. NEW

When you press NEW, the screen is not cleared, but the program within the memory disappears. The actions of HOME are just contrary to the actions of NEW.

c. CLEAR

CLEAR can clear out some variables or array in the computer.

**6. FRE(ø)**

This instruction tells you that how many memory addresses there are to be used, it uses byte as unit. It is to be used with PRINT, the usage:




**7. INVERSE and NORMAL**

When you press INVERSE, the characters on the screen and the background will reverse colors. For example, if a character is white with blue background, it will turn into blue character with white background. NORMAL is to be used to put the screen into original condition.




**8. SPEED**

We may assume the speed at which the screen displays the characters, when SPEED=ø, it is the lowest speed and when SPEED=255, the highest speed. An attempt to use a mumber larger than 255 will result in the error: ILLEGAL QUANTITY ERROR.

```
]SPEED=300

?ILLEGAL QUANTITY ERROR
```

← Something wrong, it's illegal value

Now, try the following proram and note the change of display rates.

```
10  SPEED= 1
20  FOR I = 1 TO 100
30  PRINT I;
40  NEXT I
50  SPEED= 255
60  FOR I = 1 TO 100
70  PRINT I;
80  NEXT I
```

Slow

fast

**9. CONTROL X**

CONTROL X tells MPF-II to omit the statement on the correct line. There will be the sign "\" at the end.

```
]10 FOR I=1 TO 10\
```

Symbol after control

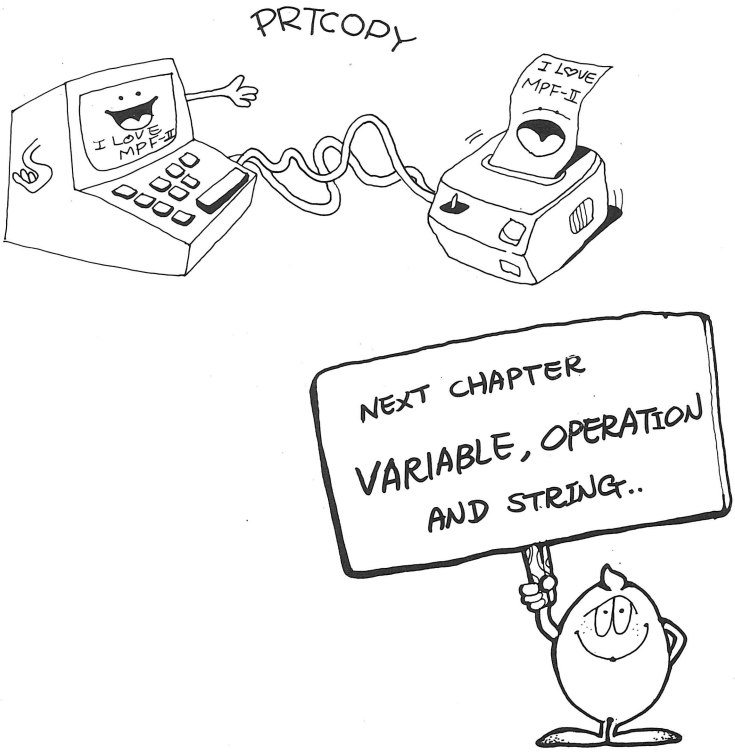
## 7 • 4 Control Instructions of Printer

In MPF-II, there are three instructions to control the printer: PRTON, PRTOFF and HC, Control - P.

1. PRTON is the abbreviation of PRINT ON. It can deliver the data shown in the screen to the printer to be printed out. But it can not print out data with pictures.

2. PRTOFF is the abbreviation of PRINT OFF. It can clear out the activity of PRTON.

3. To print out a hard copy of what is on the video display, press the H and C keys at the same time or press the Control and P keys simultaneously.



# 8

# VARIABLE, OPERATION AND STRING

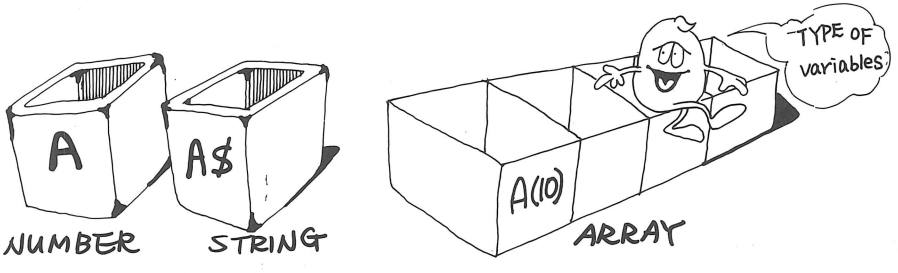
In computer programs, we can use a variable to represent the address of a memory. The function of each variable is the same with that of the memory in a computer.

In computer, variables can be divided into several categories:

1. Value variables, such as A, B.

2. String variables, such as A$, B$.

3. Array variables, such as A(1Ø), A(3,5), A(4,5,8).




Computer operators can be divided into three categories:

1. Numeric operators, such as +, -, *, /.

2. Comparative operators, such as >, <.

3. Logic operators, such as NOT, AND, OR.

## 8 • 1 Value Variable and its Operation

In MPF-II, variables are divided into two groups:

Integer: We have to add "%" after integer variables.

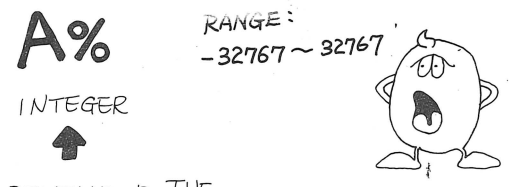
Real Number: MPF-II assumes a real number if the variable is not specially assigned.

**1. Integer:**

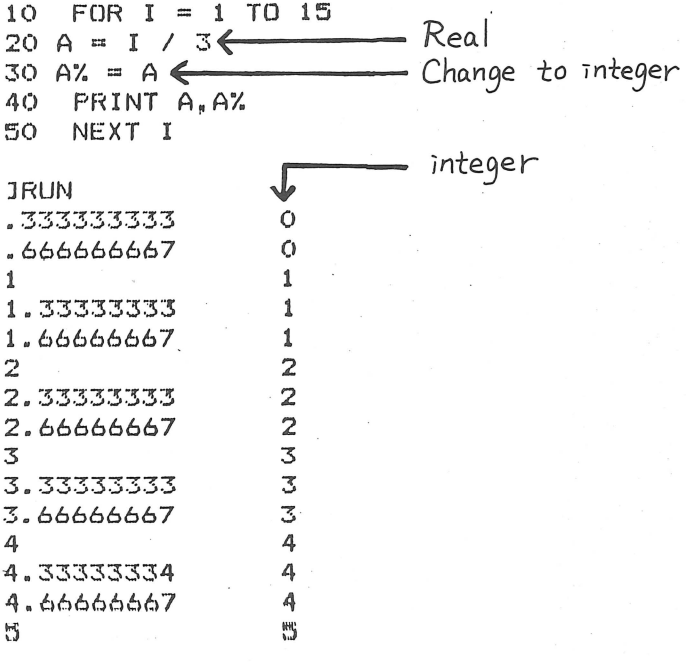
In MPF-II, each integer in computer memory occupies two bytes; that is, 16 bits. Integers are limited values between -32767 and 32767. If we want to change a real number into an integer, we can specify an integer on the left side of an equal sign. Try the following program, and then you will be able to understand how to convert a real number into an integer.

```
10  FOR I = 1 TO 6
20 A =  RND (1) * 6 + 1
30 A% = A
40  PRINT A,A%
50  NEXT I

]RUN
6.83882197        6
1.61870576        1
1.106289          1
5.67606013        5
4.31100663        4
4.70451467        4
```



The following program points out clearly that A and A% are two different variables -- the former is a real variable, the later is an integer variable, and both can coexist.




When applying integers, please don't use ",". You can not write the number 32000 as 32,000.

**2. Real number:**

In MPF-II, the range of real numbers is between +9.99999999E+37, and the effective number of digits is nine. Therefore, at most we can have nine digits.




In printing a number, if you want to predict its format, you can follow the following rules:

a. Suppose it is a negative number, the negative sign will always be printed out. For example:

```
]PRINT 132*-12
-1584
```

Negative number

Print negative sign

b. Suppose the absolute value falls between 0 and 99999999, it will be printed out as an integer. For example:

```
]PRINT 111111111*9
999999999
```

Print out the integer

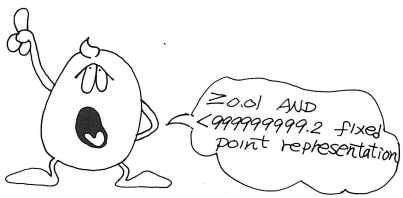
c. Suppose the absolute value of the number is equal to or is more than 0.01 and besides less than 99999999.2, it will be printed out in fixed point representation and without exponents. For example:

```
]PRINT 0.01
.01
```

> 0.01

```
]PRINT 999999999.199
999999999
```

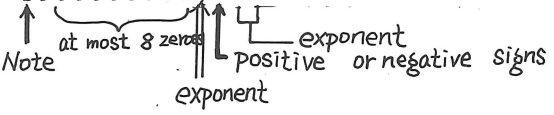
< 999999999.2



d. Suppose the limitation of the number is not included in items b and c, it will be printed in scientific representation.

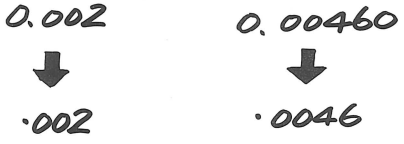
FOR EXAMPLE:

```
]PRINT 111111112*9
1.00000001E+09
```




```
]PRINT 999999999.21
1E+09
```

If the real number is printed out in scientific representation, before the decimal point, there must be an non-zero number, and after the decimal point there should be 8 places at most. E stands for exponent which is followed by either a negative or a positive sign and two places for exponents. Any leading zeroes are not printed. Trailing zeroes are also not printed. If all digits after a decimal point are all zeros, then the decimal point together with those zeros are all omitted.



When an exponent is present, the negative and positive signs will be printed. A "+" is printed when the exponent is a positive integer, and a "-" is printed while the exponent is negative; print the exponents of two digits (the first digits is always Ø, because the effective number of digits is nine.) We can now make use of the chart below, to study scientific representations:

| General notation | | Scientific notation |
|---|---|---|
| 1000000000 | ⟷ | 1E+09 |
| .000000001 | ⟷ | 1E-09 |
| -123456789 | ⟷ | -123456789E+08 |
| -.00123456789 | ⟷ | -1.23456789E-03 |

REMEMBER TWO KINDS OF REPRESENTATION METHODS



**3. The operation of MPF-II**

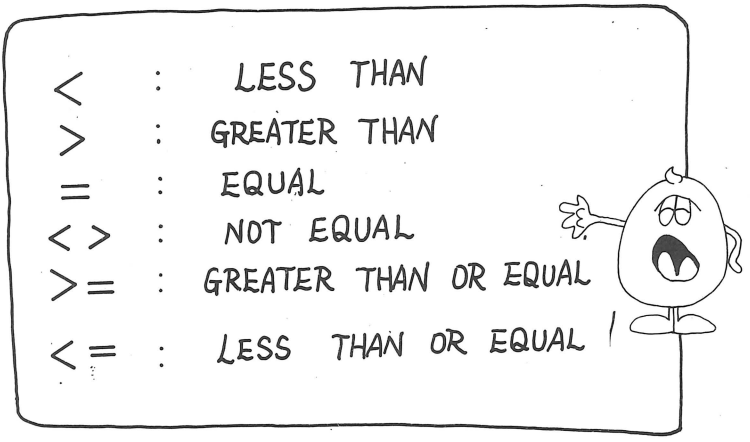
MPF-II operators can be divided into three groups:

a. Numerical Operators: shown below.

+ , − , ✳ , / , = , − ,

b. Relative Operators: shown below.

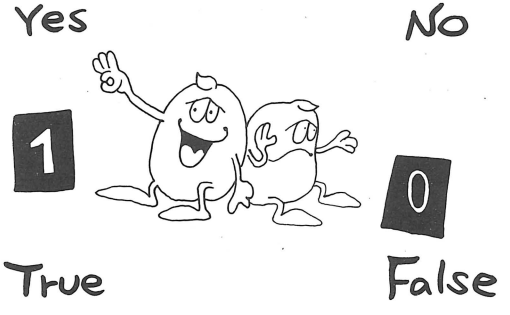
< , > , = , < > , > = , < =




These operation symbols are used to compare the relationship of two numbers. In MPF-II, if the comparison is true, the computer will retúrn a 1; if not, it will return a Ø.

Pay close attention to the following program and its results.

```
10 PRINT 5>4                    → 1
20 PRINT 4=5-1                  → 1
30 PRINT 12=3*4                 → 1
40 PRINT 3=12/5                 → 0
50 PRINT 9-8=1                  → 1
60 PRINT 8-5>4                  → 0
70 PRINT 12>=35-4               → 0
80 PRINT 34<>42=8               → 0
90 PRINT 64<=34+2               → 0
100 PRINT 34 >1234              → 0
```



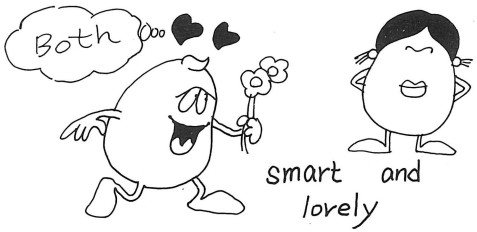
c. Logic Operators

In MPF-II, there are three logic operators :

AND, OR, NOT

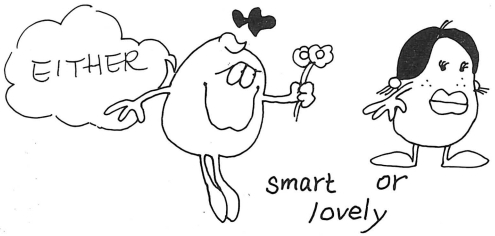
If you want to get a 1 from an AND operation, the number on each side of the AND must be a 1. This is shown below.

```
1 AND 1 = 1
1 AND 0 = 0
0 AND 1 = 0
0 AND 0 = 0
```



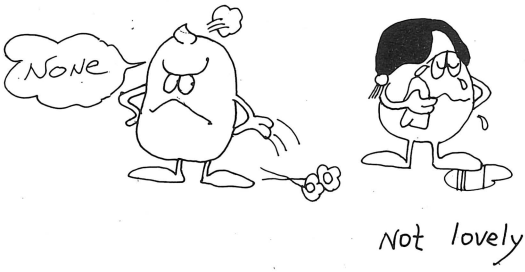

If you want to get a 1 from an OR operation, either one or both of the numbers on each side of the OR must be a 1. This is shown below.

```
1 OR 1 = 1
1 OR 0 = 1
0 OR 1 = 1
0 OR 0 = 0
```




In a NOT operation, you get a 1 from a Ø, and get a Ø from a 1. This is shown below.

```
NOT 1 = 0
NOT 0 = 1
```




Here are some examples:

```
]PRINT (5>4) AND (3>2)
1
```

Because both 5 > 4 and 3 > 2 are true, the result will be 1.

```
]PRINT (3<2) OR (1<0)
0
```

Because 3<2 and 1<Ø, are false, the result will be Ø.

```
]PRINT NOT(5>4)
0
```

Because it is true that 5 > 4, but operated on by NOT, the result is Ø.

The following program demonstrates logic operations.

```
                                                    ]RUN
 10  PRINT  NOT ((3 + 4) >  = 4)                    0
 20  PRINT (9 = 8) OR (4 * 6 > 3 *
      4)                                            1
 30  PRINT (9 > 8) AND (5 > 3)                      1
 40  PRINT (9 * 2 > 3 * 5) AND (2 *
      3 > 54 * 2)                                   0
 50  PRINT  NOT (3 = 2)                             1
 60  PRINT (2 = 3) OR (21 = 7 * 3)                  1

 70  PRINT 12 >  = 35 - 4                           0
 80  PRINT 34 <  > 42 = 8                           0
 90  PRINT 54 <  = 34 + 2                           0
 100  PRINT 34 * 12 > 1234                          0
```

## 8 • 2 String

A "$" must be added to string variables. The string instructions that are used by MPF-II are shown below.

| | | |
|---|---|---|
| LEN(A$) | STR$(X) | VAL(A$) |
| CHR$(X) | ASC(A$) | LEFT$(A$,X) |
| RIGHT$(A$,X) | MID$(A$,X,Y) | + |

We have learned the following string operators in chapter 5. If you don't remember, please review these operations.

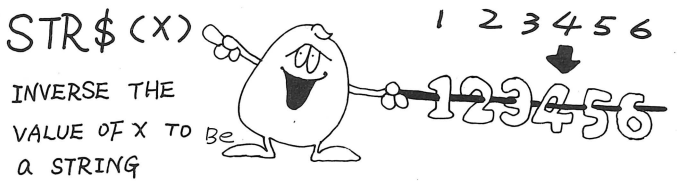
| | |
|---|---|
| LEN(A$) | LEFT$(A$,X) |
| RIGHT$(A$,X) | MID$(A$,X,Y) |

There are five remaining string operators that we have not learned yet. They will now be introduced one by one.

STR$(X), VAL(A$) CHR$(X) ASC(A$)

### 1. STR $(X) and "+"

STR is the abbreviation of string. STR$ indicates that we should convert the value of X in the brackets, which must be a number, into a string.



In the following programs, you can separate the seven-digit telephone number into two parts by using a "-" for the separation. For example:

7014733 changes to 701-4733

3219742 changes to 321-9742

7627535 changes to 762-7535

```
10  INPUT "TEL NUMBER    ";A
20  B$ =  STR$ (A)
30  C$ =  LEFT$ (B$,3)
40  D$ =  RIGHT$ (B$,4)
50  PRINT C$ + "-" + D$
60  GOTO 10
```

Change the number to a string

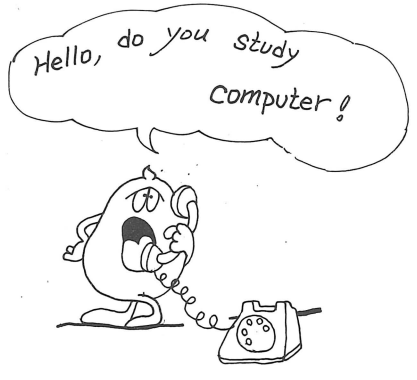
Get the left most three characters from string B$

Get the right most four characters from string B$

Combine two with "-" strings

```
]RUN
TEL NUMBER    3219742
321-9742
TEL NUMBER    7014733
701-4733
TEL NUMBER    7627535
762-7535
TEL NUMBER    7082273
708-2273
TEL NUMBER    3810653
381-0653
TEL NUMBER

BREAK IN 10
```



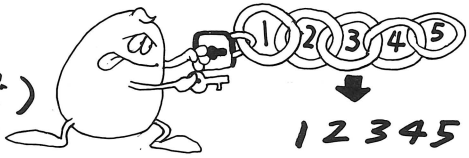
**2. VAL (A$)**

VAL is the abbreviation of value. The function of VAL(A$) is the reverse of STR$(A). Try the following program and then you will see how VAL$ works.

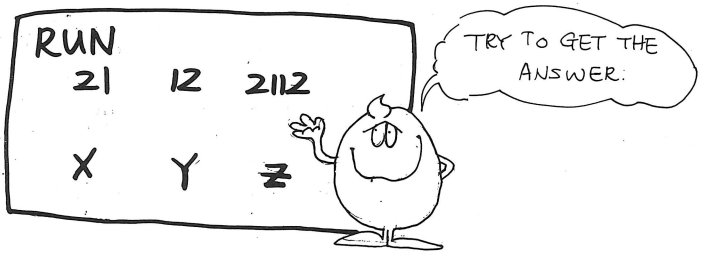
```
10 A$ = "53"
20 B$ = "12"
30  PRINT  VAL (A$ + B$)
```

← Convert the combined string into the number

```
]RUN
5312
```




```
10 A$ = "21"
20 B$ = "12"
30 X =  VAL (A$)
40 Y =  VAL (B$)
50 Z =  VAL (A$ + B$)
60  PRINT X,Y,Z
100  END
```




```
10 A$ = "12"
20 B$ = "13"
30 X =  VAL (A$) +  VAL (B$)
40 Y =  VAL (A$) *  VAL (B$)
50 Z =  VAL (A$) -  VAL (B$)
60  PRINT X,Y,Z
```

Add the value converted from the strings
Multiple the value converted from the strings
Substract the value converted from the strings

```
]RUN
25
 ⇧
(12+13)
```

```
156
 ⇧
(12x13)
```

```
-1
 ⇧
(12-13)
```

**3. What is ASCII?** (It stands for American Standard Code for Information Interchange)

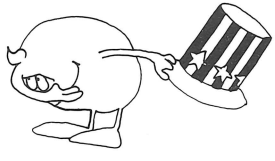
a. We have known that the computer deals only with binary digits. We call this binary digit a binary code.

b. In other words, the characters (such as A, B, C,....), digits (0,1,2,....), signs (+, -, *, /) used by computers are all represented by binary codes.

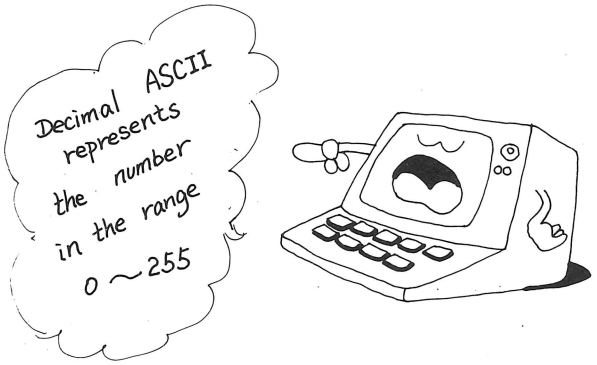
c. If we don't have a standard binary code, it would be very inconvenient. But now we have already had it standarized, and the most popular one is the American Standard Code for Information Interchange (ASCII).

American Standard Code for Information Interchange

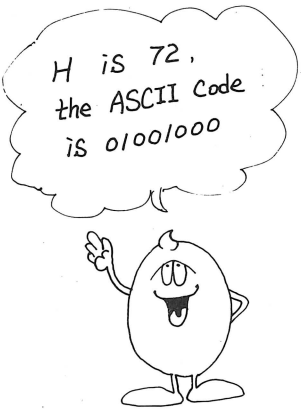
ASCII



d. The A in the ASCII is represented by 01000001 binary which 65 in decimal system. The L in the ASCII is is 01001100 binary which is 78 in decimal system. Each character, digit, and sign can all be represented by a specific ASCII code and in order to be easily recognized each has an equivalent decimal number. The following table lists the alphabetic decimal ASCII codes. Because a byte is equal to 8 bits and is equal to 256, the digits represented by decimal ASCII codes are between 0 and 255.




| CHRACTER | DECIMAL ASCII |
|---|---|
| A | 65 |
| B | 66 |
| C | 67 |
| D | 68 |
| E | 69 |
| F | 70 |
| G | 71 |
| H | 72 |
| I | 73 |
| J | 74 |
| K | 75 |
| L | 76 |
| M | 77 |
| N | 78 |
| O | 79 |
| P | 80 |
| Q | 81 |
| R | 82 |
| S | 83 |
| T | 84 |
| U | 85 |
| V | 86 |
| W | 87 |
| X | 88 |
| Y | 89 |
| Z | 90 |
| [ | 91 |



**4. ASC (A$)**

a. In BASIC language, there are two instructions to handle ASCII and string conversions. One of them is the instruction to change a string into an ASCII code. The instruction is ASC which is the abbreviation of ASCII. For example:

```
PRINT  ASC ("A")
65
```

The computer prints 65. From the above example, we realize that what is printed by the computer is not the real ASCII code represented by 1's and Ø's, but the ASCII code represented by the decimal system.

```
PRINT  ASC ("A B")
65 <-------------- only print the first
                   character
```

In the brackets, there are two characters, A ar B, but the computer only prints 65. From the above exa_ le, we realize that the ASC instruction only changes the first character in a string into decimal ASCII codes.

b. The brackets after ASC can also be used with other string operations. See the following example:

The results of the two programs are the same. The two statements of the first program lines 2Ø, 3Ø are equal to those of the second program line 2Ø.

```
10 A$ = "MICRO"
20 B$ = MID$ (A$,3,1)
30 PRINT B$"=" ASC (B$)
```

```
10 A$ = "MICRO"
20 PRINT MID$ (A$,3,1)"=" ASC
   (MID$ (A$,3,1))
30 END
```

In the examples above, because A$ assigns the string MICRO, we get the third character from MID$ (A$, 3, 1), the result of the printing is 67 which is the decimal ASCII code of C.

c. Here is another example for your understanding:

```
10 A$ = "MICRO"
20 B = LEN (A$)
30 FOR I = 1 TO B
40 PRINT MID$ (A$,I,1), ASC ( MID$
   (A$,I,1))
50 NEXT I
```

You will certainly realize the answers that are printed.

```
]RUN
CHA          ASCII
M            77
I            73
C            67
R            82
O            79
```

**5. CHR $(X)**

a. CHR is the abbreviation for character. CHR$ which is used to change decimal ASCII code into character or sign is ASC's. For example:

PRINT CHR$(65)

A ←

CHR$(X)

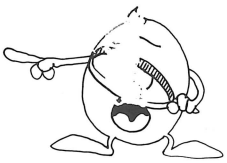
CHANGE ASCII CODE TO A CHARACTER OR ITS SYMBOL




b. To build a table of decimal values and their ASCII equivalents, you can run the following program. This program starts with the decimal value 32, before 32 there are some display control codes which might influence the computer operation. Line 32 is empty, because decimal 32 is a space in ASCII codes.

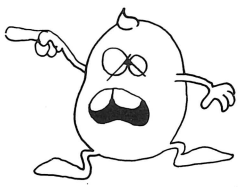
```
]LIST

10  HOME
20  FOR I = 32 TO 95
30  PRINT I;"    "; CHR$ (I),
40  FOR K = 1 TO 500: NEXT K
50  NEXT I
```



The following diagram is a part of the result shown on the screen.

```
]RUN
32                33  !             34  "
35  #             36  $             37  %
38  &             39  '             40  (
41  )             42  *             43  +
44  ,             45  -             46  .
47  /             48  0             49  1
50  2             51  3             52  4
53  5             54  6             55  7
56  8             57  9             58  :
59  ;             60  <             61  =
62  >             63  ?             64  @
65  A             66  B             67  C
68  D             69  E             70  F
71  G             72  H             73  I
74  J             75  K             76  L
77  M             78  N             79  O
80  P             81  Q             82  R
83  S             84  T             85  U
86  V             87  W             88  X
89  Y             90  Z             91  [
92  \             93  ]             94  ^
95
```



## 8 • 3 Array

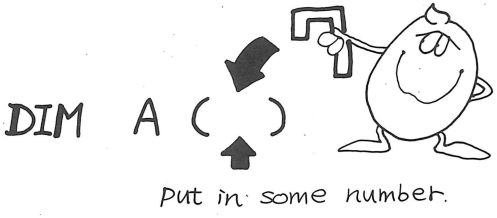
### 1. Array variable

The variables used in section 1 are called simple variables, because each variable can only represents a digit or a string. But in mathematics, we often need a variable to represent a series of digits. Similarly, in a class, we have so many names, so we often need a variable to represent a series of strings.

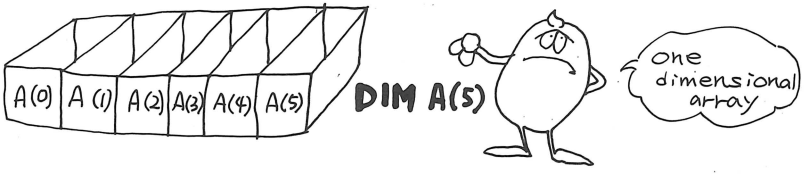
Array variables can be divided into three groups:
a. Integer array variables, such as A%(8,5), B%(2,4).
b. Real number array variables, such as A(9,5), D(5,9).
c. String array variables, such as A$(10,5), C$(6,8).

### 2. DIM

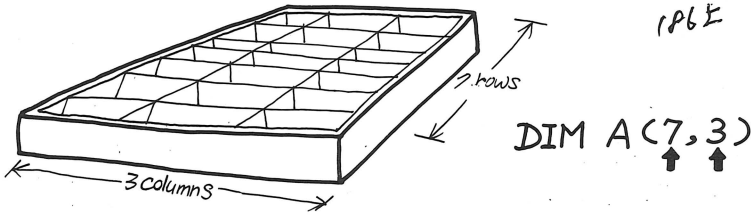
When giving the array a name, we need to include the size of an array. DIM is the abbreviation for dimension. There are one-dimensional, two dimensional and multi-dimensional arrays.



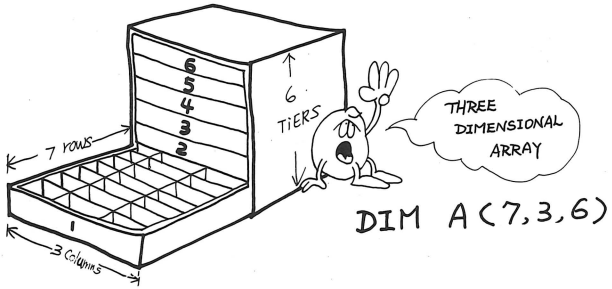

a. The following diagram is a one-dimensional array.



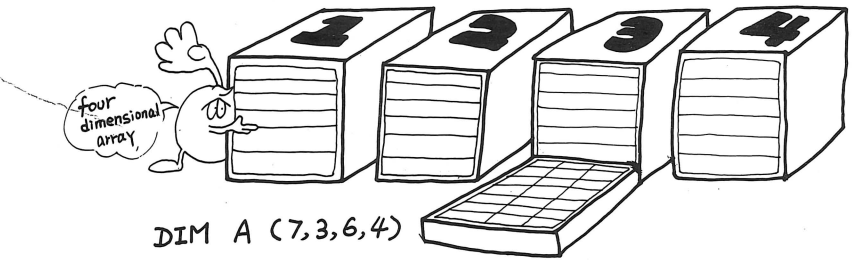
b. The following diagram is a two-dimensional array. In DIM A(7,3) 7 indicates the number of rows; 3 indicates the number of columns.




c. The following diagram is a three-dimensional array. In DIM A(7,3,6), 7 indicates row; 3 column; 6 tier.




d. The following diagram is a four-dimensional array.



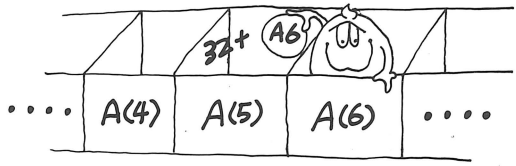
### 3. One-dimensional Arrays

a. DIM A(20) is a one-dimensional array

DIM A(20)

It shows that the name of the array is A and there are altogether 21 variables, from A(0) to A(20). The method of accessing these 21 variables is the same as accessing the common variables. Any dimension variable can be accessed. For example:

A(5) = 32 + A(6)




This statement means after we add 32 to the number in variable A(6), we put the result into A(5). 5 and 6 are called subscripts. Subscripts are used to access a specific variable (element) in an array.

b. In the following program, we can return all the digits in the array to zero.

```
10  DIM A(12)
20  FOR I = 1 TO 12
30 A(I) = 0
40  NEXT I
50  FOR I = 1 TO 12
60  PRINT A(I),
70  NEXT I
```

Return the value of "I" to be 0

Print A(I)

```
]RUN
 0            0            0
 0            0            0
 0            0            0
 0            0            0
```

c. In the following program, we can take some numbers and sequentially put them into the array. Look at the result:

```
]RUN
A%(1)=85          A%(2)=85
A%(3)=60          A%(4)=27
A%(5)=42          A%(6)=88
A%(7)=37          A%(8)=13
A%(9)=29          A%(10)=84
A%(11)=12         A%(12)=82
```

d. The following program, reads 8 numbers in an array and then prints them out.

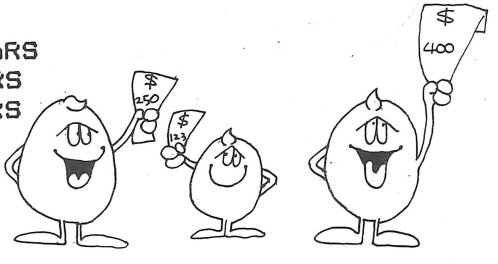
```
10 DIM A(8)
20  FOR I = 1 TO 8
30  READ A(I)
40  NEXT I
45  FOR I = 1 TO 8
50  PRINT "A("I")="A(I),
60  NEXT I
70  DATA  12,13,43,21,56,78,9,102
```

```
]RUN
A(1)=12           A(2)=13           A(3)=43
A(4)=21           A(5)=56           A(6)=78
A(7)=9            A(8)=102
```

e. The following program, demonstrates how to use one-dimensional digit variables and string variables.

```
 5   DIM A$(3) , N(3)
 10 Y$ = "DOLLARS"
 20  FOR I = 1 TO 3
 30  READ A$(I),N(I)
 40  NEXT I
 50  HOME
 55  FOR I = 1 TO 3
 60  PRINT A$(I)" HAS "N(I)" "Y$
 70  NEXT I
 00  DATA  SHIH,250,LIN,123,HUE,40
       0
```

```
]RUN
SHIH HAS 250 DOLLARS
LIN HAS 123 DOLLARS
HUE HAS 400 DOLLARS
```




**4. Two-dimensional Arrays**

The following example demonstrates how to use a two-dimensional array:

```
 10  DIM A(3,3)
 20  FOR I = 1 TO 3
 30  FOR J = 1 TO 3
 40 A(I,J) = I * J
 50  NEXT J
 60  NEXT I
 70  FOR I = 1 TO 3
 75  FOR J = 1 TO 3
 80  PRINT "A("I","J")="A(I,J);"
       ";
 90  NEXT J
100  PRINT
110  NEXT I
```

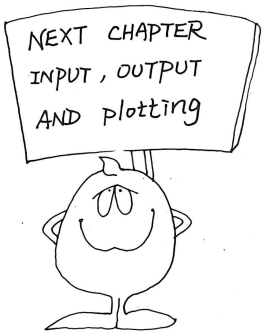
```
]RUN
A(1,1)=1      A(1,2)=2      A(1,3)=3
A(2,1)=2      A(2,2)=4      A(2,3)=6
A(3,1)=3      A(3,2)=6      A(3,3)=9
```

**5. Three-dimensional Arrays**

The following program demonstrates how to use a three-dimensional arrays.

```
10  DIM A(3,3,3)
20  FOR I = 1 TO 3
30  FOR J = 1 TO 3
40  FOR K = 1 TO 3
50 A(I,J,K) = I * J * K
60  NEXT K: NEXT J: NEXT I
70  FOR I = 1 TO 3
80  FOR J = 1 TO 3
90  FOR K = 1 TO 3
100  PRINT "A("I","J","K")="A(I,J
     ,K);"    ";
110  NEXT K
120  PRINT
130  NEXT J: NEXT I
```

Try it now, and find the result



# 9

# INPUT AND OUTPUT INSTRUCTION

## 9 • 1 INPUT and OUTPUT Instructions

The following instruction groups are described in this chapter.

1. INPUT A, INPUT A$  INPUT "XY" ; A
2. READ ····· DATA ····· RESTORE
3. GET
4. PRINT "X=" ; X
5. DEF FNA(X) = X * X + 2 * X + 3

Most of these instructions have been used in the previous chapters, the following instructions are those we have not described before.

GET A$ , RESTORE , DEF FNA (X)

**1. INPUT instruction**

In general, input instructions can be classified into four groups:

a. Accept as input a simple number variables for instance:

```
10  INPUT A
20  FOR I = 1 TO A
30 S = S + I
40  NEXT I
50  PRINT "SUM=";S
60  END

>RUN
?10000
SUM=50005000
```

What we get from this program is the sum of 1 up to the input variable A. In this example, we entered 10000, the sum is 50005000.

b. Accept as input a string variabl, for example:

```
10  INPUT A$
20  PRINT A$
30  PRINT  LEN (A$)
40  PRINT  LEFT$ (A$,2)
50  PRINT  RIGHT$ (A$,3)
60  PRINT  MID$ (A$,4,3)
```

Print the string

Count the length of the string

Print the left most 2 character of string

Print the right most 3 character of string

Print 3 characters from the 4th one

```
]RUN
?MICRO PROFESSOR
MICRO PROFESSOR
15
MI
SOR
RO
```

The string is input first and is then used in the program. In this example, we first entered "MICRO PROFESSOR" and then computed the length, the first two characters, the last three characters, and the fourth, fifth and sixth characters in MICRO PROFESSOR.

c. Request and accept a number variable, for example:

```
10  INPUT "ENTER TWO NUMBERS  ";A
    ,B
20  PRINT A"+"B"="A + B
30  PRINT A"-"B"="A - B
40  PRINT A"/"B"="A / B
50  PRINT A"*"B"="A * B
```

```
]RUN
ENTER TWO NUMBERS  123,343
123+343=466
123-343=-220
123/343=.358600583
123*343=42189
```

In this program, we first put A and B in and then printed out their sum, difference, product and quotient.

d. Request and accept number variables and string variables, for example:

```
10  INPUT "ENTER YOUR NAME AND AG
    E ";N$,A
20  PRINT "MR. "N$" YOU ARE "A" Y
    EARS OLD"
30  GOTO 10
```

input the string, then the number

```
]RUN
ENTER YOUR NAME AND AGE SHIH,43
MR. SHIH YOU ARE 43 YEARS OLD
ENTER YOUR NAME AND AGE LIN,41
MR. LIN YOU ARE 41 YEARS OLD
ENTER YOUR NAME AND AGE
```

## 2. READ·····DATA AND RESTORE

We have used READ and DATA before, now we will combine them with RESTORE.

a. When you perform READ.....DATA, there must be enough data, otherwise, the computer inform you of insufficient data.

```
10  HOME
20  READ A
30  PRINT A,
40  GOTO 20
50  DATA  33,44,5,66,7,88,99,80,9
      999

]RUN
33              44              5
66              7               88
99              80              9999

?OUT OF DATA ERROR IN 20
```

← data error

b. If not all the data is used, the computer will ignore the extra data. For example:

```
10  HOME
20  FOR I = 1 TO 3
25  READ A,B,C
30  PRINT A,B,C
40  NEXT I
50  DATA  12,13,24,35,46,12,2E+05
      ,0,123,987,999,567

]RUN
12              13              24
35              46              12
200000          0               123
```

redundant data



c. The following example is the reads string data the string variable name must use the symbol $.

```
10  HOME
20  FOR I = 1 TO 3
25  READ A$,B$,C$
30  PRINT A$,B$,C$
40  NEXT I
50  DATA  READ,DATA,FOR ,NEXT ,I,
      SHIH,LIN,PRINT,HOME
```

Read strings

```
]RUN
READ          DATA          FOR
NEXT          I             SHIH
LIN           PRINT.        HOME
```

**3. RESTORE**

RESTORE can return the data pointer to the first value in the data statement, thus the same data can be manipulated again. In the following example, when the computer first reads A (line number 30), it references line 70 to read data (the data pointer has advanced to 22). But at line number 50, the pointer returns back to 11. Therefore the four reads all access the same value 11, and we have not used 22, 33 and 44.

```
10  HOME
20  FOR I = 1 TO 4
30  READ A
40  PRINT A;"    ";
50  RESTORE
60  NEXT I
70  DATA  11,22,33,44
```

RESTORE

```
]RUN
11    11    11    11
```

all access the same value 11




In the following program, we may use RESTORE twice on the same data, one is weighted computation, another unweighted computation. Study the explanation in the diagram.

```
10  REM  USE THE RESTORE STATEMEN
     T TO READ DATA
20  READ G1,G2,G3,G4
40  DATA  65,78,98,87
50  DATA  0.1,0.2,0.3,0.4
60  READ W1,W2,W3,W4
70 A = W1 * G1 + W2 * G2 + W3 * G
     3 + W4 * G4
80  PRINT "WEIGHTED AVERAGE=";A
90  RESTORE
100   REM  RE-READ
110   READ S1,S2,S3,S4
120 A = (S1 + S2 + S3 + S4) / 4
130   PRINT "UNWEIGHTED AVERAGE=";
      A
140   END

]RUN
WEIGHTED AVERAGE=86.3
UNWEIGHTED AVERAGE=82
```

Comments of the program

Weighted operation

unweighted operation

result of weighted operation

result of unweighted operation



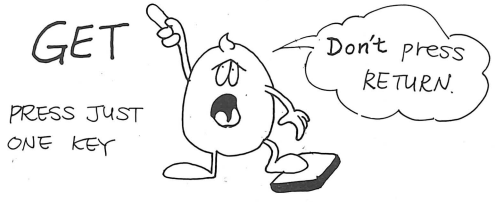
**4. GET instruction**

a. A GET instruction inputs the result of a key press (only one key) on the keyboard. When you press a key, the screen does not display the pressed key and you don't have to press RETURN.

b. The program below accepts one letter and instructs the computer to print the letter.

```
10  HOME
20  PRINT "YOU PRESSED  "
30  GET A$
40  PRINT A$
50  GOTO 20

]RUN
YOU PRESSED  M
YOU PRESSED  I
YOU PRESSED  C
YOU PRESSED  R
YOU PRESSED  O
YOU PRESSED
```



c. Here is another example of using GET.

```
10  HOME
15  PRINT "YOU PRESSED    ";
20  GET B$
30  PRINT B$
40  REM  IF ENTRY IS AN E, END PROGRAM
50  IF B$ = "E" THEN 80
60  GOTO 15
80  PRINT "END"
```

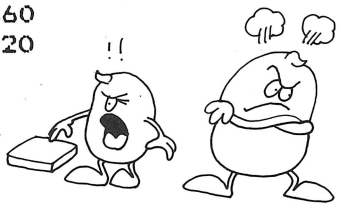
GET "E" then stop

d. GET can also be used to get value only and then to compute. Here is an example:

```
5  HOME
10  PRINT "GET A NUMBER DIGIT    "

20  GET N
25  IF N = 0 THEN 60
30  PRINT N"*5*8="N * 5 * 8
40  GOTO 10
60  PRINT "END"

]RUN
GET A NUMBER DIGIT    3*5*8=120
GET A NUMBER DIGIT    9*5*8=360
GET A NUMBER DIGIT    8*5*8=320
GET A NUMBER DIGIT    END
```



## 5. PRINT

We have used PRINT before in various places. Now we would like to give you a general review by way of a practical example:

```
10  INPUT "TWO NUMBERS  ";A,B
20  PRINT A;B
40  PRINT A,B
50  PRINT  TAB( 10);A, TAB( 20);B

60  PRINT  SPC( 10);A, SPC( 20);B

70  PRINT "A=";A,"B=";B

]RUN
TWO NUMBERS  123,4567
1234567
123                4567
          123        4567
           123                4567
A=123              B=4567
```

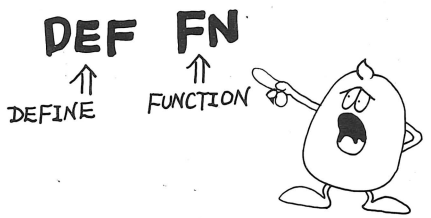


## 6. DEF FN

a. DEF FN is the acronyms of define and function. The instruction DEF FN enables you to define a function and then to use the function. Try the following program:

```
10  PRINT "X","FNA(X)"
20  DEF  FN A(X) = X * X + 3 * X +
     1
30  FOR X = 0 TO 5
40  PRINT X, FN A(X)
50  NEXT X

]RUN
X                  FNA(X)
0                  1
1                  5
2                  11
3                  19
4                  29
5                  41
```



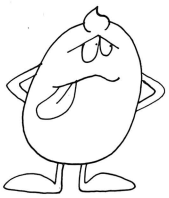

b. In the above program: i. Assign the function to be $X^2 + 3X + 1$.
ii. Enter a value for X.
iii. use the value in the function
iv. Print out the results.

c. The following program is the extension of the above program. The argument in line number 60 FNA (I) is I, it has the same effect as the X in FNA (X).

```
5  PRINT "X","FNA(X)"
10  DEF  FN A(X) = X * X + 3 * X +
    1
20  FOR X = 1 TO 5
30  PRINT X, FN A(X)
40  NEXT X
50  PRINT
60  PRINT "I","FNA(I)"
70  FOR I = 1 TO 3
80  PRINT I, FN A(I) + 8
90  NEXT I

]RUN
X                FNA(X)
1                5
2                11
3                19
4                29
5                41

I                FNA(I)
1                13
2                19
3                27
```



d. The following program uses several functions, namely DEF FNA and RND.

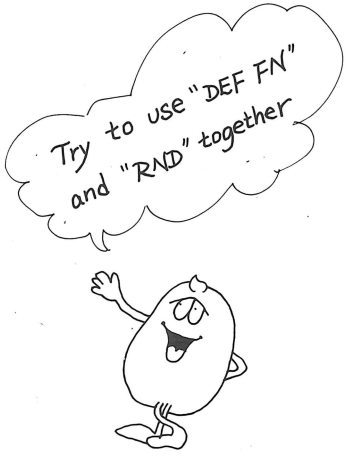
```
10  PRINT "X","INT(FNA(X))"
20  DEF  FN A(X) =  RND (1) * X
30  FOR I = 0 TO 12 STEP 3
40 A% =  FN A(I)
50  PRINT I,A%
60  NEXT I

]RUN
X                 INT(FNA(X))
0                 0
3                 0
6                 1
9                 6
12                6
```

Print out the title

Define function"x"

Get integer

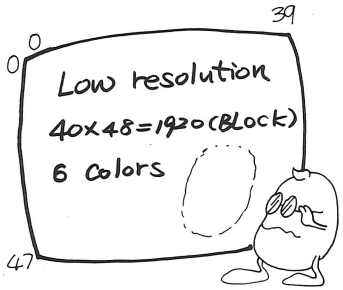


## 9 • 2 Plotting

MPF-II possesses the capability of plotting. Its plotting instructions can be divided into low and high resolution categories.

### 1. Low-resolution Plotting Instructions

Low-resolution refers to pictures plotted with less details. The low-resolution capability of MPF-II consists of 1920 blocks; that is, the screen is divided into 1920 blocks. Horizontal coordinates are 0 to 39, and vertical coordinates are 0 to 47.

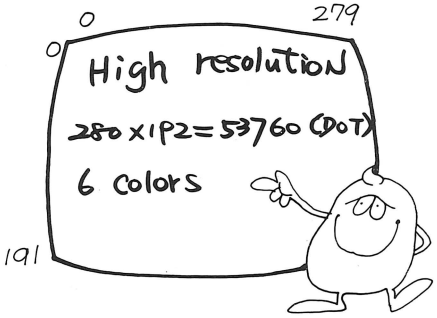



In low-resolution mode, we may assign 6 different colors, refer to chapter 5 and the next section.

### 2. High-resolution Plotting Instructions

High-resolution refers to pictures plotted with greater details. The high-resolution capability of MPF-II consists of 53760 dots; that is, the screen is composed of 53760 dots; that is, the screen is composed of 53760 dots.

In low-resolution, we use a block mode, while in high-resolution, we use a dot mode. In high-resolution, we can only assign six colors.



## 9 • 3 Low-resolution Instructions

The low-resolution instructions of MPF-II are given below. Among these instructions, only SCRN (X,Y) has not been mentioned before, so this section will focus on SCRN (X,Y) and combine the other instructions.

```
COLOR=X      PLOT X,Y
HLIN X1,X2 AT Y
VLIN Y1,Y2 AT X
SCRN(X,Y)
```

1. Review of low-resolution instructions

Try the following programs and see if you can understand the results; if not, please review chapter 5.

```
10  GR : COLOR= 13
20  FOR X = 0 TO 39
30  FOR Y = 0 TO 39 - X
40  IF Y > 19 THEN 60
50  PLOT X,X
55  GOTO 70
60  PLOT X,Y
70  NEXT Y
80  NEXT X
```

Assign the color

Draw the graphic

Draw the graphic



```
10  GR : COLOR= 13
20  FOR X = 0 TO 39
30  FOR Y = 0 TO 39 - X
35  IF X > 19 THEN 60
40  IF Y > 19 THEN 60
50  PLOT Y,X
55  GOTO 70
60  PLOT X,Y
70  NEXT Y
80  NEXT X
```

Assign the color

Draw the graphics

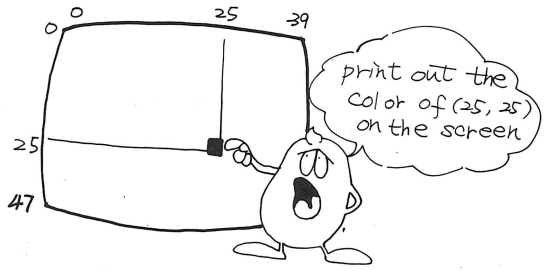
2. SCRN is the abbreviation of screen. SCRN (12,24) means that the computer prints out an assigned color at location (12,24) on the screen.

```
10  GR : COLOR= 10
20  PLOT 25,25
30  PRINT  SCRN( 25,25)
```

Assign the color

Print out the color one (25,25)

SCRN (25, 25)



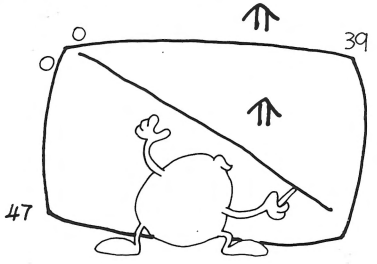
```
5  HOME
10  GR
20  COLOR=  INT ( RND (1) * 16 +
     1)
30  PLOT 12,34
40  PRINT  SCRN( 12,34)
50  END

]RUN
9
```

```
10  REM  DRAW A DIAGONAL LINE
20  GR
30  COLOR=  INT ( RND (1) * 16 +
     1)
40  FOR Y = 0 TO 39
50  PLOT Y,Y
60  NEXT Y
70  PRINT  SCRN( Y,Y)

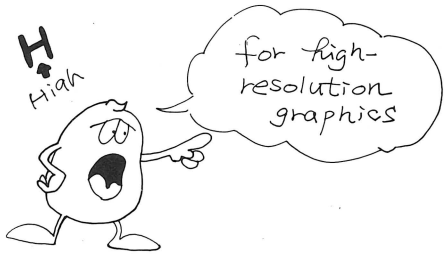
]RUN
15
```

DRAW A DIAGONAL LINE



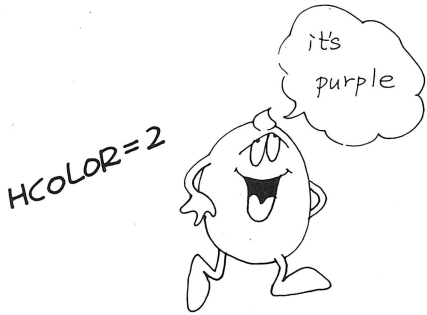
## 9 • 4 High-resolution Instruction

The high-resolution instructions of MPF-II are the following:




### 1. HCOLOR

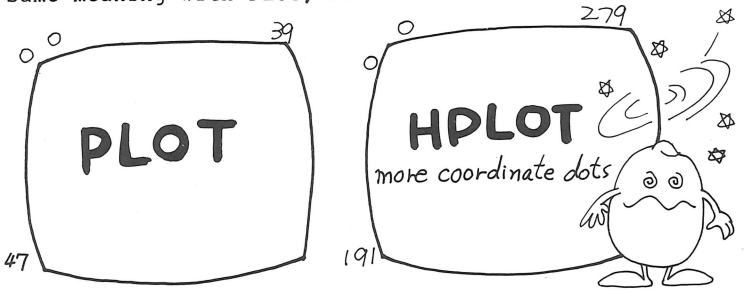
HCOLOR assigns colors in the high resolution mode; that is, we may use HCOLOR to assign the color while plotting in the high-resolution mode. HCOLOR can use a value of from Ø to 7 in selecting a color. Such as:




For example, when HCOLOR= 1, we assign the green color.

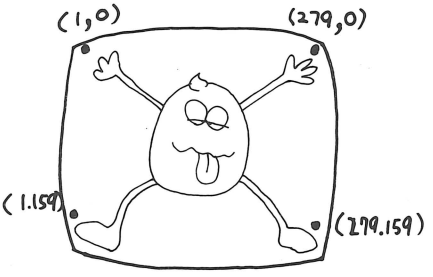
**2. HPLOT**

HPLOT is the plotting of high-resolution, it bears the same meaning with PLOT, but has more coordinate dots.

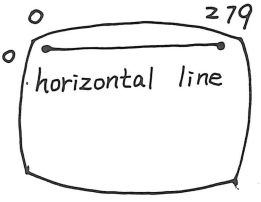



a. For example: This program can plot four dots around the edge as shown in the picture.

```
10  HGR
20  HOME
30  HCOLOR= 5
40  HPLOT 279,0
50  HPLOT 1,0
60  HPLOT 1,159
70  HPLOT 279,159
```



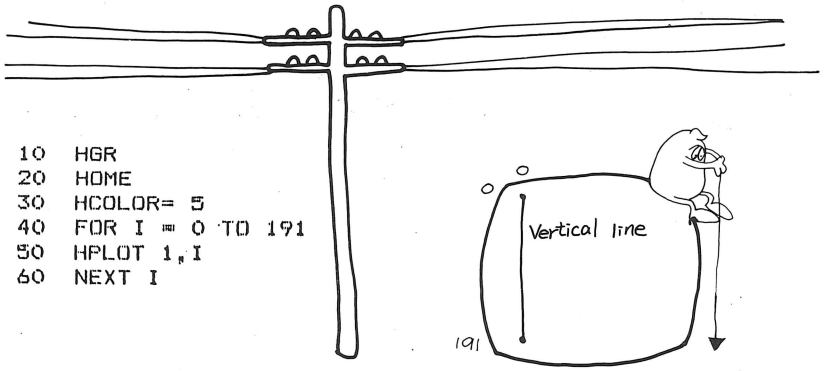

b. In the following program, we can form a horizontal line by means of dots (vertical axis is Ø).




```
10  HGR
20  HOME
30  HCOLOR= 5
40  FOR I = 0 TO 279
50  HPLOT I,0
60  NEXT I
```



c. In the next program we may form a vertical line by means of dots (horizontal axis is Ø).



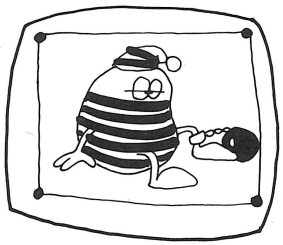
```
10  HGR
20  HOME
30  HCOLOR= 5
40  FOR I = 0 TO 191
50  HPLOT 1,I
60  NEXT I
```



d. In the next program, we may plot a rectangle around the screen.

```
10   HGR : COLOR= 3
20   FOR I = 0 TO 279
30   HPLOT I,0
40   NEXT I
50   FOR I = 0 TO 191
60   HPLOT 279,I
70   NEXT I
80   FOR I = 279 TO 0 STEP - 1
90   HPLOT I,191
100  NEXT I
110  FOR I = 191 TO 0 STEP - 1
120  HPLOT 0,I
130  NEXT I
```





e. If a value goes beyond the assigned value, the computer will print out a warning of errors.

```
10  GR : COLOR= 3
20  HOME
30  FOR I = 0 TO 300
40  HPLOT I,0
50  NEXT I

]RUN

?ILLEGAL QUANTITY ERROR IN 40
]
```

excess 279

illegal value

f. Try the following two programs by yourself.

```
10  HGR : HCOLOR= 5
20  HOME
30  FOR I = 0 TO 279
40  FOR J = 0 TO 191
50  HPLOT I,J
60  NEXT J: NEXT I
```

```
10  HGR
20  HOME
30  HCOLOR= 5
40  FOR I = 0 TO 191
45  FOR J = 0 TO 279
50  HPLOT J,I
55  NEXT J
60  NEXT I
```

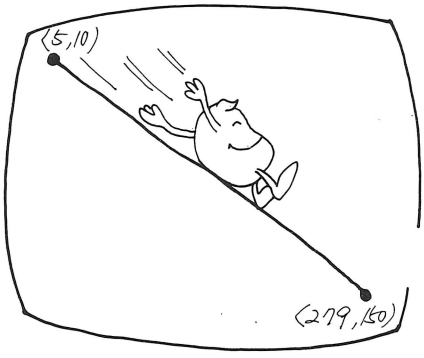


### 3. HPLOT $X_1, Y_2$ To $X_2, Y_2$

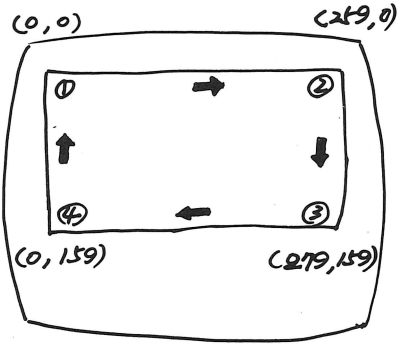
In the previous examples, we have already known how to use HPLOT. Now let's try some examples.

Example 1:

```
10  HGR
20  HCOLOR
30  HPLOT 5,10 TO 279,
    150
```

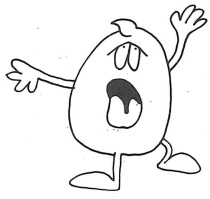



```
10  HGR
20  HCOLOR=3
30  HPLOT  0,0 TO 259,
    0 TO 259,159 TO  0,
    159 TO 0, 0
```

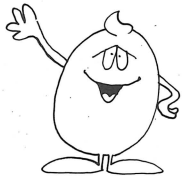


In this program, we may plot a frame.

```
10  HGR
20  HCOLOR= 3
30  FOR I = 0 TO 60 STEP 10
40  HPLOT I,I TO 259 - I,I
50  NEXT I
```



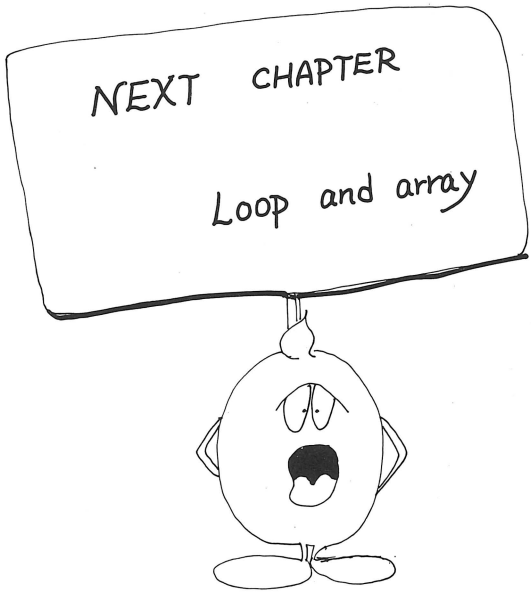
```
10  HGR
20  HCOLOR= 3
30  FOR I = 0 TO 60 STEP 10
40  HPLOT I,I TO 259 - I,I TO 259
      - I,159 - I
50  NEXT I
```



```
10  HGR
20  HCOLOR= 3
30  FOR I = 0 TO 60 STEP 10
40  HPLOT I,I TO 259 - I,I TO 259
      - I,159 - I TO I,159 - I TO
     I,I
50  NEXT I
```



```
10  HGR
20  HCOLOR= 3
30  FOR I = 0 TO 150 STEP 5
40  HPLOT I,I TO 259 - I,I TO 259
      - I,159 - I TO I,159 - I TO
     I,I
50  NEXT I
```



# 10

# LOOP AND ARRAY

In this chapter, we will learn the instruction about loop control.

Those instructions we have known are as follows:

GOTO, IF ..... THEN, FOR ..... NEXT,
GOSUB ..... RETURN, POP, ON ..... GOTO,
ON ..... GOSUB, ONERR ..... GOTO,
RESUME



There are still some new instruction to learn:

ON ..... GOTO, GOSUB ..... RETURN,
ON ..... GOSUB, ONERR ..... GOTO,
RESUME

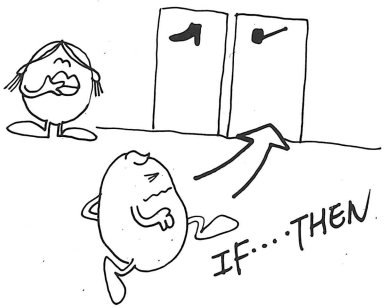
## 10.1 ON·····GOTO

1. Comparison between ON.....GOTO, IF.....THEN and IF.....GOTO

   a. The following program computes the number of times the numbers 1 to 6 appear when a die is rolled 600 times.

```
10  FOR I = 1 TO 600
20 B% =  RND (1) * 6 + 1
30  IF B% = 1 THEN A(1) = A(1) +
     1
40  IF B% = 2 THEN A(2) = A(2) +
     1
50  IF B% = 3 THEN A(3) = A(3) +
     1
60  IF B% = 4 THEN A(4) = A(4) +
     1
70  IF B% = 5 THEN A(5) = A(5) +
     1
80  IF B% = 6 THEN A(6) = A(6) +
     1
90  NEXT I
100  FOR I = 1 TO 6
110  PRINT "A("I")="A(I)
120  NEXT I
```

```
]RUN
A(1)=87
A(2)=96
A(3)=102
A(4)=91
A(5)=127
A(6)=97
```



b. The following program performs the same actions as the above one, but it uses IF...GOTO.

```
10  FOR I = 1 TO 600
20 B% =  RND (1) * 6 + 1
30  IF B% = 1 GOTO 100
40  IF B% = 2 GOTO 110
50  IF B% = 3 GOTO 120
60  IF B% = 4 GOTO 130
70  IF B% = 5 GOTO 140
80 A(6) = A(6) + 1
90  GOTO 200
100 A(1) = A(1) + 1: GOTO 200
110 A(2) = A(2) + 1: GOTO 200
120 A(3) = A(3) + 1: GOTO 200
130 A(4) = A(4) + 1: GOTO 200
140 A(5) = A(5) + 1: GOTO 200
200  NEXT I
210  FOR I = 1 TO 6
220  PRINT "A("I")="A(I)
230  NEXT I
```



decide

Store

Print the result

```
]RUN
A(1)=87
A(2)=96
A(3)=102
A(4)=91
A(5)=127
A(6)=97
```

c. As a matter of fact, we can also compute the same quanity ON.....GOTO. Try the following program:

```
10  FOR I = 1 TO 600
20 B% =  RND (1) * 6 + 1
30  ON B% GOTO 100,110,120,130,14
    0,150
100 A(1) = A(1) + 1: GOTO 200
110 A(2) = A(2) + 1: GOTO 200
120 A(3) = A(3) + 1: GOTO 200
130 A(4) = A(4) + 1: GOTO 200
140 A(5) = A(5) + 1: GOTO 200
150 A(6) = A(6) + 1
200  NEXT I
210  FOR I = 1 TO 6
220  PRINT "A("I")="A(I)
230  NEXT I
```

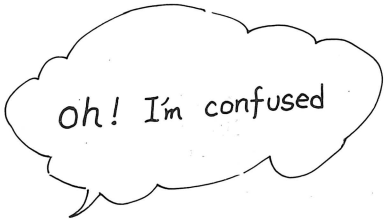
```
]RUN
A(1)=96
A(2)=108
A(3)=91
A(4)=102
A(5)=99
A(6)=104
```

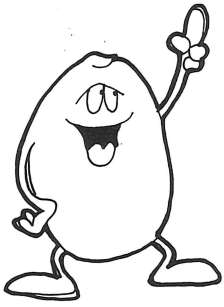


ON····
GO TO

IF ····· GO TO

ON····· GO TO



2. IF.....THEN, IF.....THEN.....GOTO and IF.....GOTO

In fact, in MPF-II, the usage of IF.....THEN, IF.....THEN....GOTO and IF.....GOTO are all the same. For example:

❶

```
10  FOR I = 1 TO 10
20 A% =  RND (1) * 20 + 1
30  IF A% >  = 10 THEN 50
35  PRINT A%"<10",
40  GOTO 60
50  PRINT A%">=10",
60  NEXT I

]RUN
19>=10          6<10            11>=10
1<10            18>=10          20>=10
19>=10          8<10            19>=10
2<10
```

THEN

❷

Same

```
10  FOR I = 1 TO 10
20 A% =  RND (1) * 20 + 1
30  IF A% >  = 10 THEN  GOTO 50
35  PRINT A%"<10",
40  GOTO 60
50  PRINT A%">=10",
60  NEXT I

]RUN
9<10            18>=10          4<10
17>=10          11>=10          11>=10
14>=10          13>=10          6<10
16>=10
```

THEN GO TO

❸

```
10  FOR I = 1 TO 10
20 A% =  RND (1) * 20 + 1
30  IF A% >  = 10 GOTO 50
35  PRINT A%"<10",
40  GOTO 60
50  PRINT A%">=10",
60  NEXT I

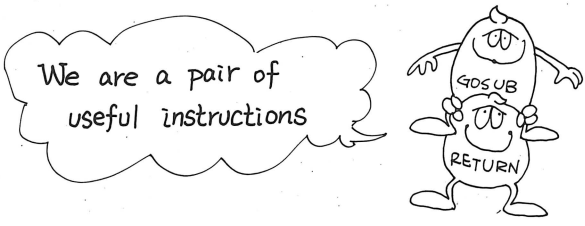
]RUN
12>=10          17>=10          14>=10
7<10            3<10            10>=10
13>=10          17>=10          10>=10
17>=10
```

GO TO

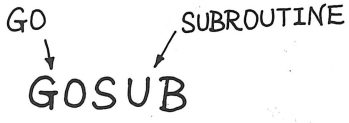
## 10 • 2 GOSUB and RETURN

### 1. GOSUB

A pair of useful instructions in BASIC are GOSUB




In some programs, there are tasks we have to perform repeatedly, under such circumstances, GOSUB and RETURN are the most suitable instructions. In order to avoid writing some similar smaller programs over and over again in various places, we may treat those smaller programs as subroutines. GOSUB is the acronym of GO and SUBROUTINE




### 2. RETURN

After the computer goes to a subroutine (GOSUB), it has to come back to its original place to go on completing the unfinished program. Therefore, in a program with GOSUB's we have to use RETURN.

### 3. Examples and explanation

This program can be divided into two parts: a main program and a subroutine. Line numbers 10-70 are the main program; line numbers 100-140 are the subroutine. The instruction GOSUB is in the main program, while RETURN is in the subroutine.

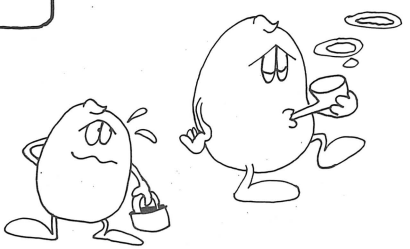
main program—GOSUB sub-program—RETURN

```
10 X = 5:Y = 1
20  GOSUB 100
30 X = 7:Y = 3
40  GOSUB 100
50 X = 4:Y = 5
60  GOSUB 100
70  END
100  REM  SUBROUTINE
110  FOR I = 1 TO X
120  PRINT Y;
130  NEXT I
140  PRINT
150  RETURN
```

GOSUB in main program

RETURN in sub-program

```
]RUN
11111
3333333
5555
```



Now, let's examine the execution of this program:

a. In line number 10, assign X=5, Y=1.
b. Line number 20 GOSUB to the beginning of the subroutine in line number 100.
c. Execute subroutine, after execution, return to line number 30 and assign X=7, Y=3.
d. Line number 40 GOSUB to 100.
e. Again at line 50 assign new values to X and Y and GOSUB to 100 at line 60, line 70 end of program.

```
10 X = 5:Y = 1
20  GOSUB 100
30 X = 7:Y = 3
40  GOSUB 100
50 X = 4:Y = 5
60  GOSUB 100
70  END

100  REM  SUBROUTINE
110  FOR I = 1 TO X
120  PRINT Y;
130  NEXT I
140  PRINT
150  RETURN
```

**4. Examples**

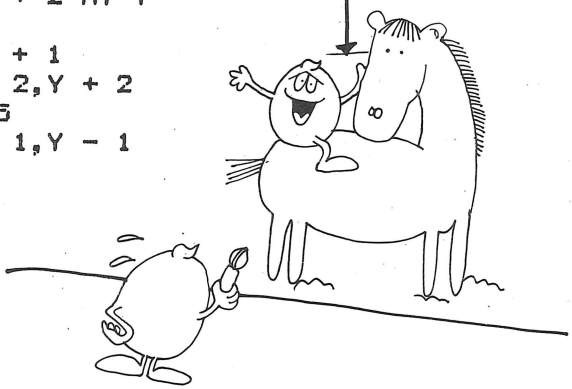
a. The following program uses a subroutine to activate a horse with orange legs, a white head and a blue trunk.

```
5   HOME
20   GR
30   REM   FIRST HORSE CENTER
40  X = 12:Y = 35
60   GOSUB 1000
70   REM   SECOND HORSE CENTER
80  X = 33:Y = 2
100   GOSUB 1000
200   END
1000   REM   PUT A HORSE ANYWHERE O
       N THE SCREEN
1010   COLOR= 7
1020   PLOT X,Y - 1
1030   HLIN X,X + 2 AT Y
1040   COLOR= 7
1050   PLOT X,Y + 1
1060   PLOT X + 2,Y + 2
1070   COLOR= 15
1080   PLOT X - 1,Y - 1
1090   RETURN
```

The first position to draw a horse

The second position

Draw a horse



b. When you are executing the above program drawing a horse, you might make some mistakes. This is because the beginning position assigned is too close to the edge of the screen, thus the coordinate values go beyond the limit of the screen. In order to check out such a mistake in the subroutine, we may add the following statement in the subroutine:

```
]1012 IF X<1 THEN X=1

]1014 IF X>37 THEN X=37

]1016 IF Y<1 THEN Y=1

]1018 IF Y>38 THEN Y=38
```

After the addition, the total program turns out to be:

```
5  HOME
20  GR
30  REM  FIRST HORSE CENTER
40 X = 12:Y = 35
60  GOSUB 1000
65  FOR K = 1 TO 500: NEXT K
70  REM  SECOND HORSE CENTER
80 X = 33:Y = 2
85  FOR K = 1 TO 500: NEXT K
100  GOSUB 1000
200  END
1000  REM  PUT A HORSE ANYWHERE O
      N THE SCREEN
1010  COLOR= 7
1012  IF X < 1 THEN X = 1
1014  IF X > 37 THEN X = 37
1016  IF Y < 1 THEN Y = 1
1018  IF Y > 38 THEN Y = 38
1020  PLOT X,Y - 1
1030  HLIN X,X + 2 AT Y
1040  COLOR= 7
1050  PLOT X,Y + 1
1060  PLOT X + 2,Y + 2
1070  COLOR= 15
1080  PLOT X - 1,Y - 1
1090  RETURN
```

First position to draw a horse

delay time

Second position

delay time

Draw a horse

a. In this program, line numbers 65, 85 are added to delay the movement of the horse.
b. Line numbers 1012-16 are to prevent the errors in plotting.




**5. Subroutine within a subroutine**

Sometimes it is desirable to have a subroutine within a subroutine. Try the following:

```
10  REM  MAIN PROGRAM
20  DIM A(100)
30  GOSUB 2000
40  PRINT "END"
50  END
2000  REM  FIRST LEVEL SUBROUTINE
2010  FOR I = 1 TO 5
2020 A(I) = 5 * I
2030  GOSUB 3000
2040  NEXT I
2050  RETURN
3000  REM  NEXTED SUBROUTINE
3010 B(I) = A(I) * 2 + 1
3020  PRINT "A("I")=";A(I),"B("I"
     )="B(I)
3030  RETURN
```

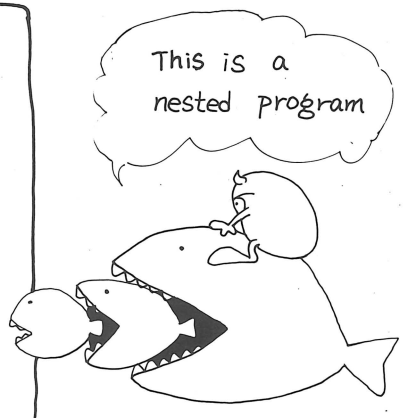
main program

Subprogram

Secondary subprogram

Please study the program and write down the result.

RUN



## 10 • 3 ON-GOSUB

### 1. ON-GOTO and ON-GOSUB

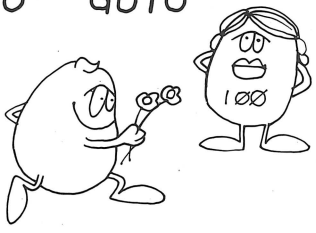
ON.....GOTO bears a resemblance to ON.....GOSUB. BUT ON.....GOTO jumps to a certain line number, while ON.....GOSUB jumps to subroutine. Remember, ON..... ..GOSUB has to be used with RETURN. Now try this:

ON……GOTO

```
10  FOR I = 1 TO 100
20 B% =  RND (1) * 3 + 1
30  ON B% GOTO 100,200,300
100 A(1) = A(1) + 1
110  GOTO 400
200 A(2) = A(2) + 1
210  GOTO 400
300 A(3) = A(3) + 1
400  NEXT I
410  FOR I = 1 TO 3
420  PRINT "A("I")=";A(I)
430  NEXT I
```

Try to run, everytime you'll get different result, please make a record for it.

NO····· GOTO



```
10  FOR I = 1 TO 3
20 B% =  RND (1) * 3 + 1
30  ON B% GOSUB 100,200,300
40  NEXT I
50  PRINT
60  FOR I = 1 TO 3
70  PRINT "A("I")=";A(I)
80  NEXT I
90  END
100  PRINT B%;
110 A(1) = A(1) + 1
120  RETURN
200  PRINT B%;
210 A(2) = A(2) + 1
220  RETURN
300  PRINT B%;
310 A(3) = A(3) + 1
320  RETURN
400  NEXT I
```

ON ····· GOSUB

```
]RUN
31233231122122332123222122 3123
A(1)=7
A(2)=14
A(3)=9
```



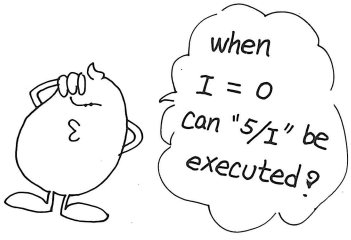

In the above program, we assign I=1, 2 or 3 by means of RND. If I=1, then GOSUB will be 100; I=2, GOSUB 200; I=3, GOSUB 300. We execute them one by one up to 30 times, then we print out the statistical figures of 1, 2, 3, etc.

## 10 • 4 ONEER·····GOTO and RESUME

### 1. ONEER·····GOTO

If there is a mistake occurs during the computer execution, the computer will interrupt the execution of program and print out error message. For example:

```
10  HOME
20  FOR I = 0 TO 10
30  PRINT I,5 / I
40  NEXT I
```




When the computer starts to run, it will stop immediately, because I=Ø and the computer is not able to execute 5/Ø.

```
]RUN
 0
?DIVISION BY ZERO ERROR IN 30
```

In BASIC language, the instruction ONERR.....GOTO is the combination of ON, ERROR and GOTO. When there is a mistake occurs, ONERR.....GOTO can prevent the execution of the program from being interrupted, because this instruction will produce an unconditional GOTO to jump to the assigned line number.

ON+ERROR GO TO
=
ONERR

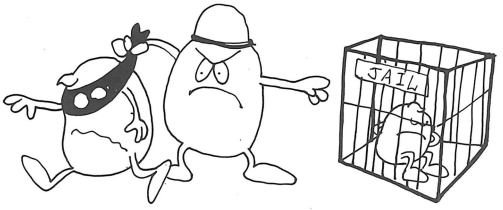
```
10  HOME
20  ONERR  GOTO 100
30  INPUT "ENTER NUMBER  ";X
40 Y = 1 / X
50  PRINT "X=";X,"Y=";Y
60  GOTO 30
100  PRINT "DON'T INPUT ZERO  "
110  GOTO 20


]RUN
ENTER NUMBER  56
X=56              Y=.0178571429
ENTER NUMBER  12
X=12              Y=.0833333333
ENTER NUMBER  0
DON'T INPUT ZERO
ENTER NUMBER
```

If error, go to 100

## ONERR GOTO

If error, go to .....

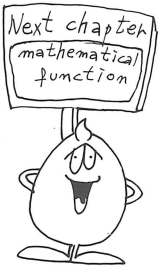



❷

```
10  HOME
20  ONERR  GOTO 100
30  INPUT "ENTER NUMBER  ";X
40  PRINT X; SPC( 5) SQR (X)
50  GOTO 30
100  PRINT "IMAGINARY ROOT  " SQR
     ( - X)
110  GOTO 30

]RUN
ENTER NUMBER  56
56      7.48331478
ENTER NUMBER  -7
-7      IMAGINARY ROOT  2.64575131
ENTER NUMBER  -5
-5      IMAGINARY ROOT  2.23606798
```

If error, go to 100



# 11

# MATHEMATICAL FUNCTIONS

130,90

In programming, we have to make use of mathematical operations from time to time. In addition to basic operations such as, +, -, *, /, ↑ more advanced operations called functions are needed.

In BASIC language, functions are generally divided into arithmetical functions and string functions. We have studied string functions in chapter 7. In this chapter we will discuss specifically the functions which have been supplied in the MPF-II BASIC language.

We have used the functions INT(X), RND(1) and SQR(X), now we will discuss the others.

MPF-II BASIC contains the following function:

1. SIN(X): Sine function
2. COS(X): cosine function
3. TAN(X): tangent function
4. ATN(X): actangent function
5. INT(X): ..... integer
6. RND(1): ..... random function
7. RND(0): random function
8. RND( ): random function
9. SGN(X): Sign function
10. ABS(X): absolute function
11. SQR(X): ..... Square root function
12. EXP(X): exponentiation function
13. LOG(X): logrithm function



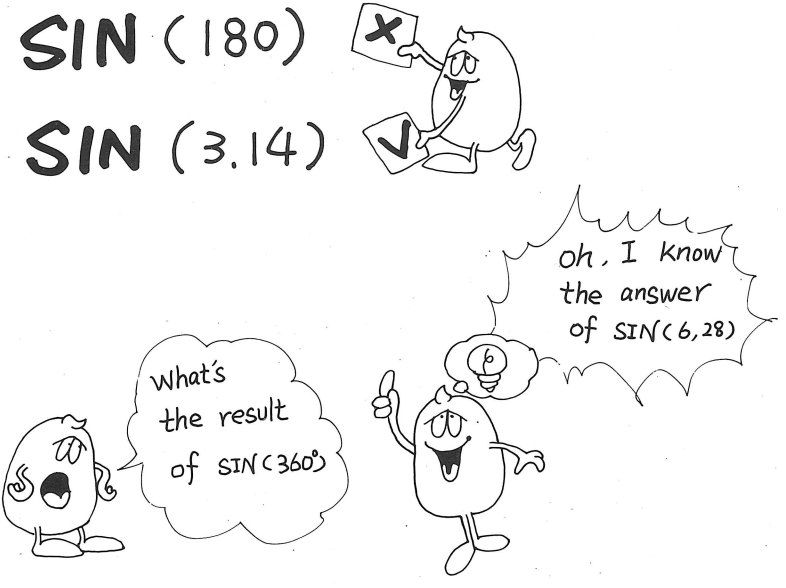
## 11 • 1 Trigonometric Functions

SIN(X), COS(X) and TAN(X) are rather important trigonometric functions. This section explains only the usage of these three functions.

### 1. Sine function SIN(X)

In the BASIC language used by MPF-II, the operation of trigonometric function is based on the unit radian. Thus the argument in the brackets must be a radian not a degree, because 360° is equal to 6.28 radians, each radian is equal to 57.324804 degrees.

Each radian is equal to 57.324804 degrees



Now, try the following program, this program is meant to get the SIN(X). From Ø" to 36Ø", we first use line number 3Ø to get the number of degrees in a radian, and then convert the degress to radians, (I/R of line number 5Ø). A conversion in made for every 15 degree.

```
10  PRINT "DEG"; TAB( 8);"RADIANS
    "; TAB( 24);"SIN(X)"
20  PRINT
30 R = 360 / 6.28
40  FOR I = 0 TO 360 STEP 15
50 Y =  SIN (I / R)
60  PRINT I; TAB( 8);I / R; TAB(
    24);Y
70  NEXT I
```

Print the title

the degree of each radian

convert to radian

```
]RUN
DEG     RADIANS          SIN(X)

0       0                0
15      .261666667       .258690844
30      .523333333       .499770103
45      .785             .706825181
60      1.04666667       .865759839
75      1.30833333       .96575386
90      1.57             .999999683
105     1.83166667       .966165865
120     2.09333333       .8665558
135     2.355            .707950909
150     2.61666667       .501148958
165     2.87833334       .260228956
180     3.14             1.59265337E-03
195     3.40166667       -.257152076
210     3.66333333       -.498389979
225     3.925            -.705697661
240     4.18666667       -.864961682
255     4.44833333       -.965339405
270     4.71             -.999997146
285     4.97166667       -.966575419
300     5.23333333       -.867349563
315     5.495            -.709074841
330     5.75666667       -.502526543
345     6.01833333       -.261766407
360     6.28             -3.1853027E-03
```

DEG ↑ Degree

RADIANS ↑ Radiam

SIN(X) ↑ Sine value



**2. Cosine function COS(X)**

The following program is computes the cosine of X, it with the above program.

```
10  PRINT "DEG"; TAB( 8);"RADIANS
      "; TAB( 24);"COS(X)"
20  PRINT
30 R = 360 / 6.28
40  FOR I = 0 TO 360 STEP 15
50 Y =  COS (I / R)
60  PRINT I; TAB( 8);I / R; TAB(
     24);Y
70  NEXT I
```

```
]RUN
DEG     RADIANS          COS(X)

0       0                1
15      .261666667       .965960169
30      .523333333       .866158094
45      .785             .707388269
60      1.04666667       .500459689
75      1.30833333       .259459981
90      1.57             7.96326206E-04
105     1.83166667       -.257921541
120     2.09333333       -.4990802
135     2.355            -.706261645
150     2.61666667       -.865361035
165     2.87833334       -.965546939
180     3.14             -.999998732
195     3.40166667       -.966370948
210     3.66333333       -.866952956
225     3.925            -.708513099
240     4.18666667       -.501837909
255     4.44833333       -.260997763
270     4.71             -2.38897806E-03
285     4.97166667       .256382449
300     5.23333333       .497699443
315     5.495            .70513323
330     5.75666667       .864561781
345     6.01833333       .96513126
360     6.28             .999994927
```



### 3. Tangent function TAN(X)

The following program to compute the tangent of X. The program is very similar to programs for sine and cosine.

```
10  PRINT "DEG"; TAB( 8);"RADIANS
     "; TAB( 24)"TAN(X)"
20  PRINT
30 R = 360 / 6.28
40  FOR I = 0 TO 360 STEP 15
50 Y =  TAN (I / R)
60  PRINT I; TAB( 8);I / R; TAB(
     24);Y
70  NEXT I
```

```
]RUN
DEG     RADIANS          TAN(X)

0       0                0
15      .261666667       .267806947
30      .523333333       .5769964
45      .785             .99920399
60      1.04666667       1.72992922
75      1.30833333       3.72216884
90      1.57             1255.76523
105     1.83166667       -3.74596809
120     2.09333333       -1.73630571
135     2.355            -1.00239184
150     2.61666667       -.579121243
165     2.87833334       -.269514557
180     3.14             -1.59265539E-03
195     3.40166667       .266100793
210     3.66333333       .574875459
225     3.925            .996026273
240     4.18666667       1.72358777
255     4.44833333       3.69865009
270     4.71             418.587575
285     4.97166667       -3.77005301
300     5.23333333       -1.74271757
315     5.495            -1.00558988
330     5.75666667       -.581250008
345     6.01833333       -.271223634
360     6.28             -3.18531886E-03
```

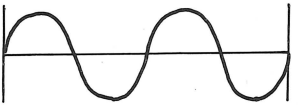


## 11 • 2 The Plotting of Functions

### 1. Simple diagram

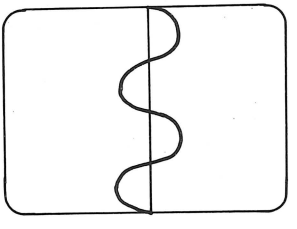
The MPF-II is able to calculate values of numbers as well as to plot some diagrams of functions. In plotting trigonometric functions we have to revolve the coordinate axis 90" clockwise, because the number of characters is limited in the screen and the printer. We have known that value of the sine ranges between + 1 and - 1. The screen of MPF-II has only 40 characters for each line, so we have to use 20 as the medium point. Try the following program. The key point of this program lies in line numbers 70 and 80.

Paper work



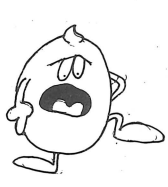
Rotate 90° degree on the screen

```
50  FOR I = 0 TO 360 STEP 10
60 Y =  SIN (I / 57.29578)
70 K% = Y * 19 + 20
80  PRINT  TAB( K%);"*"
90  NEXT I
```

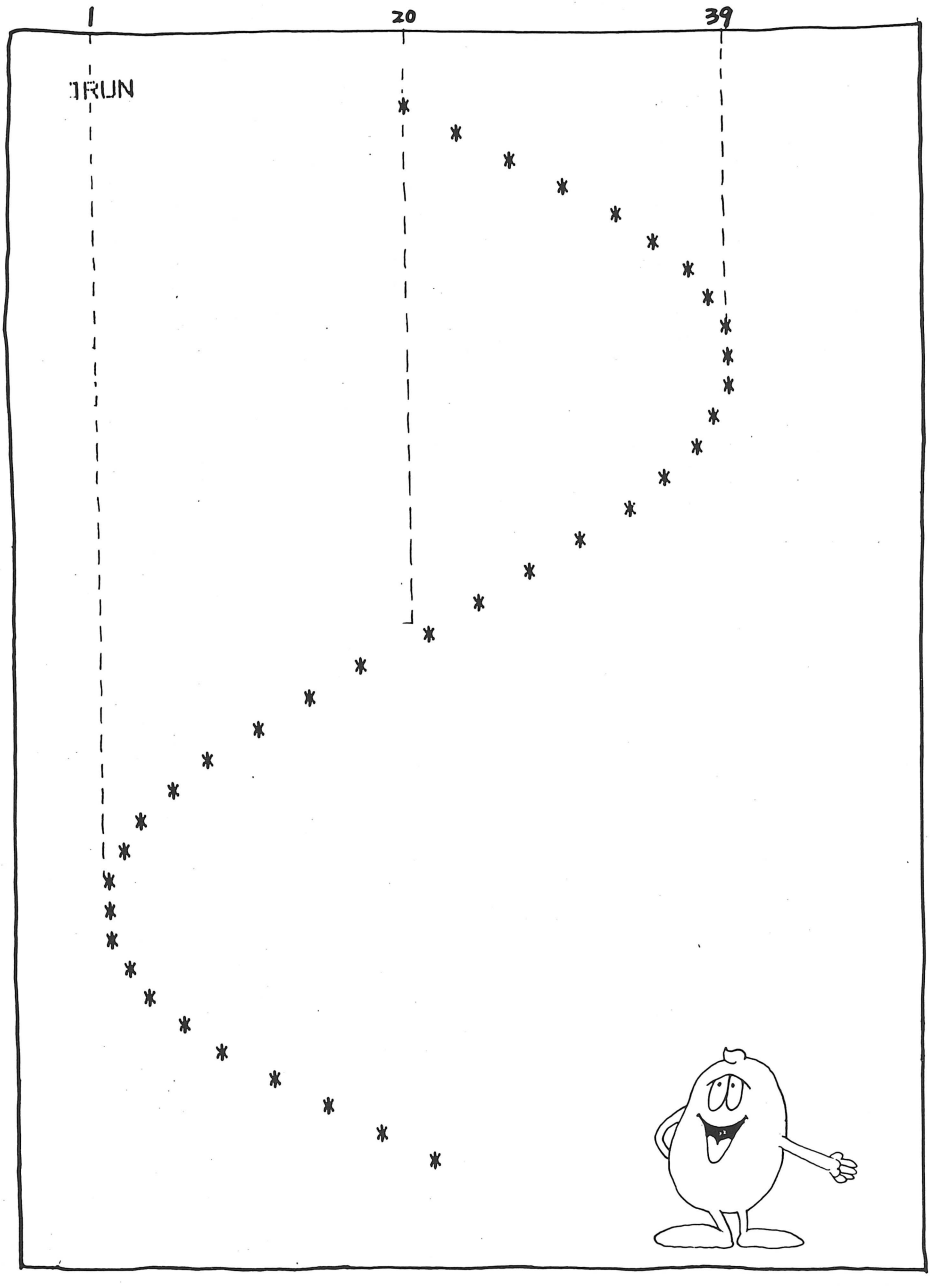


In line number 70, Y is the sine value, and its value is between - 1 and + 1; after Y is multiplied by 19, its value falls between - 19 and + 19, and then we add 20 to it, the value of K% will fall between 1 and 39, and which matches the size of the screen. Line 80 prints out the symbol " * " at TAB (K%), therefore we will have the following diagram.

$$70\ K\% = \underbrace{Y * 19}_{\text{enlarge 19 times (between -19 and +19)}} + 20$$



graphics is on the next page


1
20
39
RUN

**2. Sine function diagram**

After we revise the above program, we may have the following diagram, compare the differences between this and the previous sine plot them.

```
10  FOR I = 0 TO 39
20  PRINT ".";
30  NEXT I
40  PRINT
50  FOR I = 0 TO 360 STEP 10
60 Y =  SIN (I / 57.29578)
70 K% = Y * 19 + 20
80  IF K% > 20 THEN 120
90  PRINT  TAB( K%);"*"; TAB( 20)
    ;"."
100  GOTO 130
120  PRINT  TAB( 20);"."; TAB( K%
    );"*"
130  NEXT I
```

increment the X coordinate

Print "*"; then "."

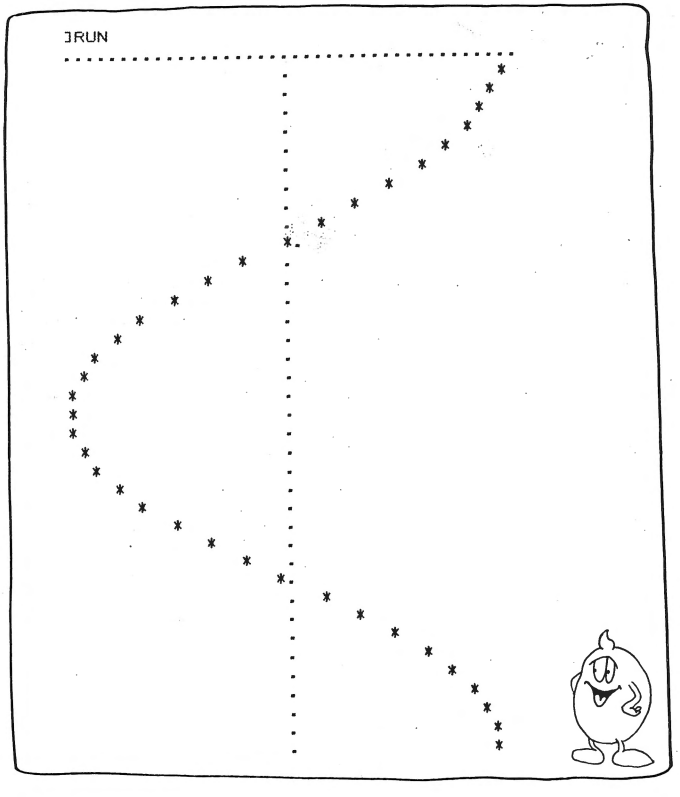
Print "."; then "*"

```
]RUN
.....................................
                    *.
                  *  .
                *    .
              *      .
            *        .
          *          .
        *            .
      *              .
    *                .
  *                  .
 *                   .
 *                   .
 *                   .
  *                  .
    *                .
      *              .
        *            .
          *          .
            *        .
              *      .
                *    .
                  *  .
                    *.
```

**3. Cosine function diagram**

Compare this program with the above one.

```
10  FOR I = 0 TO 39
20  PRINT ".";
30  NEXT I
40  PRINT
50  FOR I = 0 TO 360 STEP 10
60 Y =  COS (I / 57.29578)
70 K% = Y * 19 + 20
80  IF K% > 20 THEN 120
90  PRINT  TAB( K%);"*"; TAB( 20)
    ;"." <-------------------- Print "*"; then "."
100  GOTO 130
120  PRINT  TAB( 20);"."; TAB( K%
    );"*" <-------------------- Print "."; then "*"
130  NEXT I
```



## 11 • 3 Other Functions

### 1. Random Function

There are three kinds of random functions in MPF-II: RND(1), RND(Ø) and RND(-1). RND(1) will return a decimal number between Ø and 1.
Try the following program:

```
5  PRINT "RND(1)"
10  FOR I = 1 TO 3
20  PRINT  RND (1),
30  NEXT I
40  PRINT
45  PRINT "RND(0)
50  FOR I = 1 TO 3
60  PRINT  RND (0),
70  NEXT I
80  PRINT
85  PRINT "RND(-2)"
90  FOR I = 1 TO 3
100  PRINT  RND ( - I),
110  NEXT I
```

Print RND (1)

Print RND (0)

Print RND (-I)

```
]RUN
RND(1)
.235586735        .186784665
.37278107
RND(0)
.37278107         .37278107
.37278107
RND(-2)
2.99196472E-08   2.99205567E-08
4.48217179E-08
```

Same

From the above number we observe that:

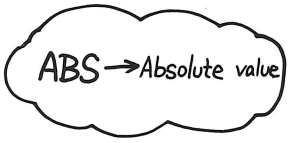
a. RND(1) will return a number between Ø and Ø.999999999
b. RND(Ø) will repeat the previous number the RND has generated.
c. Each RND(-X) gets a different number, but the value is quite small.

**2. Absolute function**

In mathematics, there must be a methord of computing absolute value. For instance:

```
10  FOR I = - 10 TO 10 STEP 5
20  PRINT "ABS("I")=" ABS (I)
30  NEXT I

]RUN
ABS(-10)=10
ABS(-5)=5
ABS(0)=0
ABS(5)=5
ABS(10)=10
```




**3. Anti-trigonometric function**

There is only one anti-trigonometric function -- ANT(X) -- in MPF-II. Try this:

```
10  FOR I = - 10 TO 10 STEP 5
20  PRINT "ATN("I")=" ATN (I)
30  NEXT I

]RUN
ATN(-10)=-1.47112768
ATN(-5)=-1.37340077
ATN(0)=0
ATN(5)=1.37340077
ATN(10)=1.47112768
```

**4. Sign function SGN(X)**

The following program is an example of SGN(X), if the value of X is negative, the computer prints out -1; if the value is zero, it prints out 0; if the value is positive, it prints out 1.

```
10  FOR I = -10 TO 10 STEP 5
20  PRINT "SGN("I")=";SGN (I)
30  NEXT I

]RUN
SGN(-10)=-1
SGN(-5)=-1
SGN(0)=0
SGN(5)=1
SGN(10)=1
```

**5. Exponential function**

The use of mathematics in areas such as engineering and physics require using natural exponents which has e as its radix. If e to x is equal to Y, then we may use the following formula to compute Y: $e^x = Y$

But in BASIC language, we can express it by Y=EXP(X), try this:

```
10  FOR I = 1 TO 5
20  PRINT "EXP("I")=";EXP (I)
30  NEXT I

]RUN
EXP(1)=2.71828183
EXP(2)=7.3890561
EXP(3)=20.0855369
EXP(4)=54.5981501
EXP(5)=148.413159
```

### 6. Logarithm function LOG(X)

LOG(X) is logarithm function which has 10 as its radix. In computing logarithms, X must be larger than 0. Try the following program:

```
10  FOR I = 1 TO 5
20  PRINT "LOG("I")=" LOG (I)
30  NEXT I

]RUN
LOG(1)=0
LOG(2)=.693147181
LOG(3)=1.09861229
LOG(4)=1.38629436
LOG(5)=1.60943791
```



Published by
Micro-Professor Publishing Corporation
977 MIN SHEN E.ROAD,
TAIPEI, TAIWAN, R.O.C.
TEL:(02)7691225
TLX: "19162 MULTIIC", "23756 MULTIIC"
Printed in Taipei
US$10.00
copyright 1982, in Taiwan, Republic of China
by Micro-Professor Publishing Corporation
Assigned Republic of China Copyright NO:2678

**MULTITECH INDUSTRIAL CORPORATION**

OFFICE/ 977 MIN SHEN E. ROAD, TAIPEI, 105,
TAIWAN, R.O.C.
TEL:(02)769-1225(10 LINES)
TLX:23756 MULTIIC, 19162 MULTIIC.

FACTORY/5, TECHNOLOGY ROAD III,
HSINCHU SCIENCE-BASED INDUSTRIAL PARK
HSINCHU, TAIWAN, 300. R.O.C.
TEL:(035)775102(3 LINES)


M2M01-209B

Micro-Professor™ THE INTRODUCTION TO BASIC PROGRAMMING

MULTITECH