COMPUTER DRILL

Introduction

Are computers important?

Computers are all around us. Like cars, phones and TVs, computers will help us improve our lives. Millions of computers are used everyday. Doctors use computers to treat patients. Police use computers to track criminals. Stores use computers to add up prices quickly. Scientists use computers for research. Banks use them to keep track of our money. Jets need computers to fly. Airports use computers to know where the jets are flying. Schools use computers to keep track of your work. Businesses use computers so they can do more work faster. Without computers we'd all be in a big mess.

What are computers?

A computer is a tool that helps us to do things better and faster. They work without getting tired. They operate at very high speed. They follow instructions perfectly. And they never forget; like we sometimes do.

The electronic computer has been around now since the late 1940's. The first ones were huge electronic mazes of wires and tubes. They were very expensive so that only big businesses and governments could afford to buy them and keep them running. The early computers required large amounts of electrical energy to run them and generated vast quantities of heat. More importantly they required a specialized team of experts called programmers to tell them what to do.

This is a far cry from how we see a computer nowadays. We see them mostly as watches, calculators, microwave ovens, VCR's, phones, TV's, CD's, and Personal Computer workstations. This is a little different from the computers used in business which are larger and require more space to operate.

We generally work with a very small version of a computer called a **microcomputer**. A microcomputer is a computing device where the **central processor** and **memory** (RAM) are contained on a very small integrated circuit "chip" called a **microprocessor**. The central processor is the device that performs arithmetic and makes decisions. The memory is where information in the form of numbers and letters are stored. A microcomputer therefore contains both a microprocessor and some ways to communicate with a person.

All micros have a keyboard for you to type in your instructions. This is called "input". Some have individual keyboards, like a typewriter; others have plastic membranes, like microwave ovens.

All micros have a device which is used to communicate back to a person. This is called "output". Some micros use your TV set, while others require a special type of TV called a "monitor". Some use paper in a device called a "printer" and some use a liquid crystal display (LCD) which is the same display that is used on some time pieces.

What is Basic?

BASIC is a computer language. The name BASIC stands for Beginner's All-purpose Symbolic Instruction Code. It was developed by professors at Dartmouth College in the mid-1960s. It was designed for students who had no previous experience in programming computers.

BASIC is basic; however there are differences in the language based upon the computer that uses it in much the same way that English is different depending on which country it is spoken in (Great Britain or the United States) or which part of the country the language is spoken in (East Coast, Midwest, South or "fer sure" the West Coast).

The differences in the language will depend on the computer that is being used and on the company that produced the language for the computer manufacturer. Microsoft BASIC, so-called because it was developed by the Microsoft Corporation of Belleview, Washington, is generally found on Apple II+, IBM PC and PC jr., Laser Computers, Commodore VIC20 and Commodore 64 and others. Other computers such as the Timex Sinclair ZX80 and the Texas Instruments 99/4 use versions of BASIC similar to MS-BASIC.

What is a program?

A computer program is a series of instructions that tells a computer what tasks you want it to perform. Computer programs are written in a **Programming Language**. BASIC is one of the languages that a human and a computer understand and so BASIC can be used to write a program that a computer can carry out for you.

Why should I learn to use a computer?

The future belongs to the people who understand how to use computers. Computers can do many things if they are given instructions. For a particular job to be done, by a person or a machine, we must specify the steps which must be done to do the job. The computer becomes an extension of our capabilities. To do this requires us to learn how to talk to our computer.

This book will help you to get started. All you need is your **PRECOMPUTER 1000** and some time.

What is this book for?

The **PRECOMPUTER 1000** manual is a step-by-step guide for understanding programs and learning to use the computer language called BASIC. Not everyone wants to write programs but once you understand how BASIC works, you'll be able to write your own programs that are similar to the ones in this book.

Who's this book for?

This guide is written for students and adults who haven't used a computer before. With a little time and effort you will discover that there is nothing difficult about learning to talk to a computer.

What if I make a mistake?

Don't worry about making a mistake. The more mistakes you make the more you will learn. This is called discovery! The computer doesn't care how many mistakes you make. There is nothing you can do to damage the computer - except smashing it. Experimenting is a very good way to make your programs do exactly what you want.

The keyboard

Since the **PRECOMPUTER 1000** has many other functions besides Computer Drill, only the special keys pertinent to this section will be explained here.

* The numeric (top row) keys are used for entering numbers. The letter keys are used for entering letters.

* The cursor is a special symbol on the display that indicates where the next character that is pressed on the keyboard will be entered in the computer. The character used on the **PRECOMPUTER 1000** is the blinking underline symbol,"_".

* The **SHIFT** key is used with another key to enter a character into the computer. For example, to enter the "+" character, you must press **SHIFT** and the key marked "+ =". If you just press the "+ =" key then the "=" character will be entered.

* Since the **PRECOMPUTER 1000** keyboard is in the CAPS LOCK mode when Computer Drill is selected, it is not necessary to use the **SHIFT** key to enter capital letters. If you press the **SHIFT** key with a letter key in this mode the key pressed will be in lower case.

* The **LEFT-ARROW** key ← and the **RIGHT-ARROW** key → are used to position the cursor left or right, respectively. They do not erase any character that is displayed, but you can type over anything in their positions.

* To enter a space, simply press the **SPACE** bar. Each time you press the **SPACE** bar one space is inserted on the display line.

* The **INS** and **DEL** keys are used to insert or delete one character at a time on the display where the cursor is positioned. The **INS** key shifts all characters on the display one character to the right to make room for the new character. You can then type in the character in the space provided. Be sure to press the **INS** key and the character for each character you want to insert.

  The **DEL** key works in a similar fashion to the **INS** key. The **DEL** key will shift all characters on the display one position to the left, thereby erasing the character under the cursor.

* Before anything on the display is sent to the **PRECOMPUTER 1000,** you must press the **ENTER** key. This key acts much like a Return key on a typewriter. You press all the keys that comprise a statement to the **PRECOMPUTER 1000** and then press **ENTER**. **PRECOMPUTER 1000** will take it from there.

* For BASIC only . . .
In BASIC, the "^" character (REPEAT ^ key) is used to tell the computer you want to raise the number preceding it to a power of the number following it. More on arithmetic later.

The **LOCK BREAK** key is used as a **BREAK** which means it can do one of two things. When you are typing a line but before you press **ENTER**, it will erase all that you have typed; or, if you are running a BASIC program, you can interrupt it by pressing this key. You can restart a program by typing **CONT** and pressing the **ENTER** key. (The **ERASE** key has the same function as the **BREAK** key)

When using the **INPUT** command, the **PRECOMPUTER 1000** will ask you to supply it with some data from the keyboard. The **PRECOMPUTER 1000** tells you it needs input with a prompt. The prompt is the "]" symbol and it appears on the display indicating that the computer is waiting for you to type something. The letters and numbers that you press appear on the display.

The display

The **PRECOMPUTER 1000** uses a Liquid Crystal Display (LCD). It is 60 characters wide but you can only see 20 characters at a time. The left arrow and right arrow are used to scroll the 20 character display window left or right, respectively.

In BASIC, you will need to press the **ENTER** key to see the next printed statement that your program is displaying. You need to do this so that the output doesn't scroll off the end of the line.

Getting started

When you select the Computer Drill Activity, you get two choices: 1) Example Programs and 2) Basic Computer.

Example Programs

You can choose from the nine programs which are a part of the **PRECOMPUTER 1000.** After you have selected one of the programs you can issue the **RUN** command (type **RUN** and press **ENTER**) to see the program work for you. Remember to press the **ENTER** key to see the next printed line on the display. You can also modify the program and see what will happen as a result of your changes.

Follow these simple steps:
1) Turn on the **PRECOMPUTER 1000**
2) Press the **Computer Drill** key
3) Press 1 to select the **Example Programs**
4) Press 1 to 9 to select your choice from the list below:

1) Computer Talk
2) French Fries
3) Rock'n Roll
4) Alarm Clock
5) Number Guessing
6) Party Game
7) Satellite
8) Animal Farm
9) Nim

The program listings are found in the **Appendix** (Pg. 79-82). After running a program you need to go back to step 2 to load in another example program. You should complete the Basic Computer Tutorial and then choose some of the programs to experiment with and learn.

Basic Computer

The activity lets you type in your own programs and run them. Follow these simple steps:
1) Turn on the **PRECOMPUTER 1000**
2) Press the **Computer Drill** key
3) Press 2 to select the **Basic Computer**

There are two modes available Command and Run. Type in this program:

1Ø PRINT "HI THERE" press the **ENTER** key

This is the usual way to make a program. You enter the program lines. Remember to press the **ENTER** key. Each line starts with a number and is followed by a statement. The **PRECOMPUTER 1000** stores the line with the other lines into memory. Later you execute the program by typing the command, **RUN.** Don't forget to press the **ENTER** key after typing **RUN.** The program that you have typed into memory will begin. The results appearing on the LCD display will be the message: HI THERE.

There is a short cut. Enter the following (no line number in front):

PRINT "HI THERE"

This time **PRINT** was used as a command and the results appeared immediately on the display. The computer executes the command right away, without waiting for you to type **RUN.** When you do this the statements are not saved for future re-use. They are executed immediately and discarded. This is not recommended for creating programs but highly recommended for use as a calculator.

Here are some other commands that can be used:

NEW

This command clears the memory of any BASIC statements that have been previously entered.

LIST

This command displays each line of your program starting with the lowest line number. Each time you press **ENTER,** the next line is displayed. You can stop this by pressing the **BREAK** key. If you enter the command with a line number after the word like this, LIST 50, the **PRECOMPUTER 1000** will begin listing the statements one after another starting with the statement at line 50.

RUN

This command instructs **PRECOMPUTER 1000** to begin executing each BASIC statement with a line number that was typed into memory. The computer will start with the lowest line number and proceed up to the highest numbered statement.

EDIT

Use the **EDIT** command when you want to change a statement that has been typed into memory without re-typing the entire line. Just type **EDIT** and the line number and press the **ENTER** key. The statement will appear on the display. Use the **LEFT-ARROW** and **RIGHT-ARROW** keys to move the cursor. Use the **INS** and **DEL** keys to make room for a new character or to remove an unwanted one.

You can type in line numbered BASIC statements in any order. The **PRECOMPUTER 1000** will sort them out for you and **LIST** them or **RUN** starting with the lowest numbered one to the highest numbered one. It is a good idea to number your lines in increments of 10 (10, 20, 30 . . .) or 100 (100, 200, 300 . . .). This way you have room to add more statements if you decide to change your program in the future.

You can insert an entirely new line in a program by using a line number that doesn't exist between two existing ones. You can delete an existing line-numbered statement by merely typing the line number and pressing the **ENTER** key. Of course, you can change an existing line by merely retyping the entire line.

CONT

This command causes the program to resume executing after encountering the BASIC command called **STOP.** The program will carry on with the next statement after **STOP.** You can type **CONT** to resume the program after you hit the **BREAK** key.

Basic computer tutorial

1. A real small program.

You and Taraesa went to the local video store to rent some movies. Taraesa brought back 12 but 4 were to gory for you and so you took them back. On that trip you got carried away and came home with 7 new ones. How many are you going to watch this afternoon? You could work this out yourself but here is a small BASIC program that can do the arithmetic for you. Type in:

1Ø PRINT 12-4+7 and press **ENTER**
2Ø END and press **ENTER**

Not much to it. Now type **RUN** and press **ENTER.** What happens? The answer, 15, appears on the display.

In BASIC, you write a series of line-numbered statements that tell the **PRECOMPUTER 1000** what to do . The first statement, numbered 10, tells the computer to work out the sum of three numbers and then **PRINT** the answer in the display. The next statement, number 20, tells the computer that this is the end of the program and that it can stop **RUNNING.** The last statement is not necessary in this program because you only wanted to print out the sum of 3 numbers and it could be removed.

Remove the last line by typing 20 and press **ENTER,** then type **RUN** and press **ENTER.** What happens? That's right, you get the same results.

Suppose you typed:

1Ø PRONG 12-4+7 press **ENTER**

Now type **RUN** and press **ENTER** what happens? You get a strange message, **"?SN ERROR"** in the display. That means you made a mistake in BASIC grammar and this is **PRECOMPUTER 1000's** way of telling you this. You need to **EDIT** the line or re-type it, changing the **"PRONG"** word to **"PRINT"** and then **RUN** the program to get the correct answer.

You can use **PRECOMPUTER 1000** as a calculator in arithmetic statements like the one above by not using a line number. Just type:

PRINT 12-4+7 and press **ENTER**

The answer will appear on the display.

Another short cut. You can use the **"?"** symbol to stand for the word **"PRINT".** When you are using BASIC as a calculator in "command mode" think of the question mark as meaning "What is 12-4+7"? When you are using the question mark as a **"PRINT"** command in a program, the computer will replace the **"?"** with the word **"PRINT"** for you.

2. Let's do some Arithmetic.

In the previous section you did a sum of three numbers. The numbers like 1, 3, 27, 14.3, etc., are called **CONSTANTS.** The program added and subtracted the constants 12, 4 and 7. The order that you do addition is unimportant: 6+10 is the same as 10+6. In subtraction, the order is important: 10-6 is not the same as 6-10. So the order that you write numbers and do arithmetic operations is important. In BASIC, operations are done from left to right.

The * symbol is used to represent multiplication. Like addition, the order of the numbers is unimportant. Here's an example. There are 2.204 pounds to a kilogram. How many pounds does a 6 kilogram parakeet weigh? Try this:

```
1Ø PRINT 6*2.2Ø4
```

The symbol / is used for division. Here, like in subtraction, the order of the numbers is important since 15/3 is 5 and 3/15 is .2. How many kilograms does a 6 pound parakeet weigh? Try this:

```
1Ø PRINT 6/2.2Ø4
```

You can raise a number to a power. To do this you need the ^ sign. The expression 5^3 means 5*5*5 or 125; similarly, 3^5 means 3*3*3*3*3 or 243. There are fractional powers; for example 2^.5 is the square root of 2 or 1.414.... Here's an example: I bet you 1 doubloon and throw the dice 10 times; each time you double your money. How much have you won? Type:

```
1Ø PRINT 2^1Ø
```

Now is the time to throw all the operations together and it gets a bit complicated. After a winning baseball game, your 7 person team is awarded 3 cans of pop for the game plus another 11 cans because you finished first in the league to share among all players. How many cans does each player get? Type this:

```
1Ø PRINT 3+11/7
```

Is it really evaluated from left to right? Yes, but was it the answer you were expecting? If the answer came out to 4.57, the computer first divided 11 by 7 getting 1.57 and then it added 3 to the result giving 4.57. You probably were expecting the answer to be 2 because 3+11 is 14, then divide by 7 gives 2.

You can see that expressions are not simply evaluated from left to right. It is necessary to have an exact set of rules to follow.

Here they are:

( )	things is parenthesis (brackets)	**highest**
^	raising to a power	
* /	multiplication and division	
+ -	addition and subtraction	**lowest**

The computer will always look at an expression in parenthesis first and do the things you put there. It could be an expression or more sets of brackets. Among the expressions, raising to a power is done first, followed by multiplication and division from left to right, followed by addition and subtraction from left to right. If you're not sure of how the computer will evaluate the expression, simply put brackets around the expressions you want done first. Remember the brackets must always be in pairs, one right bracket for each left one. If you don't then you will get a **"?SN ERROR"**.

3. Numeric Variables.

You have seen how you can use BASIC to print numbers and messages. You can also give values to a BASIC program yourself. The computer uses a letter to stand for a value. This is called a variable. For example LET D = 609. This statement puts the value 609 into a box in memory called D.

Try this; type:

```
1Ø LET A=8
```

press **ENTER**

The computer makes a box in memory and calls it A. Into this box it puts the number 8. Now type:

```
2Ø LET B=1Ø
```

press **ENTER**

The computer makes a box called B and places 10 into it. Type:

```
3Ø LET B=15
```
press **ENTER**

The computer tries to make a box called B, but it already found one called B. It has a value of 10 in it. No matter; it puts the number 15 into it. The number 10 has been replaced by the number 15. Now type:

```
4Ø LET C=A+B
```
press **ENTER**

This statement is a bit more complicated. Here's how it works. First, the computer searches for a memory box called A and finds in it the number 8. Next, it looks for the box called B and finds in it the number 15. The + sign tells it to add the numbers found in A and B together. It does that and the answer is 23. Now, where to put the answer? No problem. The = tells the computer to store the answer in a memory box called C. The computer searches for a box called C. It doesn't find it so it makes one in memory and then puts the answer into it.

Of course, if there was a memory variable called C that had a number in it from a previous operation, the old number would be replaced by the number computed by this statement.

Now let's finish this by typing:

```
5Ø PRINT "FIRST NUMBER"; A
6Ø PRINT "SECOND NUMBER"; B
7Ø PRINT "THE SUM IS"; C
RUN
```

You will see the following on the display:

```
FIRST NUMBER 8
SECOND NUMBER 15
THE SUM IS 23
```

(Don't forget to press **ENTER** after you have seen each line to get the next one displayed.)

Let's take a look at the print statements on lines 50 through 70. The words in quotes are called a **STRING** or a **LITERAL.** What ever you put between a set of quotation marks will appear on the display exactly the way you typed them. Don't forget that quotation marks come in pairs!

Next, the semicolon tells the print statement to print the next thing immediately to the right of the end of the string. This is the number 8 which is the value in memory box A.

Suppose that you asked the computer to multiply 9 million times 9 million. Let's try; type:

```
1Ø PRINT 9ØØØØØØ*9ØØØØØØ
RUN
```

The answer on the display is 8.1E+13. This is the computer's way of showing extremely large numbers; a number with 13 zeroes to the right of the decimal point. This is called scientific or exponential notation. You find the decimal point and move the number of places specified after the E to the right filling in with zeroes. The long way to write out the above answer is 81,000,000,000,000. The procedure works in reverse for very small numbers. The number 8.1E-13 is .00000000000081. This is a very small positive number. The number -8.1E+13 is a very large negative number while the number -8.1E-13 is a very small negative number.

4. String Variables.

String variables are similar to the numeric variables that we have been working with so far except that the variables contain alphabetic characters (numbers and letters and symbols). The name of the memory variable always contains a $ to distinguish it as a string variable. String variables are not used in arithmetic but allow you to store things like your pet bird's name. Here are some examples:

VARIABLE NAMES	CONTENTS
A	675
B	2.3434
C$	CURTIS
A$	ALPHABET SOUP

You can store a string variable like this:

```
1Ø LET A$="ALPHABET SOUP NO 123"
```

You can have a numeric variable called A and a string variable called A. The $ lets BASIC know how to tell them apart.

NOTE: For both numeric and string variables, only the first 2 characters of the variable names are valid. Also, they cannot resemble the name of any command.
Examples: TOTAL is interpreted as the TO command and
POTATO is also interpreted as the TO command.

5. Constants.

A constant is something that doesn't change. The computer stores it in memory just like a variable; however, since a constant doesn't change it has no variable name. The number 6 is a numeric constant; so is 4.567 or 28967.35. **"QUICK BROWN FOX"** is a string constant.

You can store constants to a variable. That's usually done to start out a program; later the contents of the variables change. Let's see. Suppose we want to count to 10. Type in this program:

```
NEW
1Ø  N=Ø
2Ø  N=N+1
3Ø  PRINT N
4Ø  IF N<1Ø THEN 2Ø
5Ø  PRINT "THE ANSWER IS" N
RUN
```

The statement at line 10 stores the numeric constant zero into N a numeric variable. In statement 20 the variable N will have a numeric constant, 1, added to itself and then the sum of that operation will be stored in place of the value previously stored there.

The program in the above example contains a programming construction called a loop. A loop is one or more BASIC statements, usually called a set, that can be executed as many times as you want. The statement in line 40 contains an **IF** statement. It is called a conditional statement and will be discussed in detail under the topic called Making decisions.

6. INPUTTING data

We have been putting information in the computer by typing constants into programs using statements like the following:

```
1Ø LET N=1Ø
2Ø LET A$="LANETTE LIKES CHOCOLATE CUPCAKES"
```

Another way to supply data is by using an **INPUT** statement. Try this out by typing:

```
NEW
1Ø INPUT "GIVE ME A NUMBER"; N
2Ø PRINT "YOUR NUMBER IS"; N
```

The computer will print the literal following the **INPUT** command in line 10 on the display. It will prompt you for a number with the] character. You can type any number of digits that you like and press **ENTER** to let the computer know you have finished. Ready? Try it, type: **RUN** and press **ENTER.**

You can also input alphabetic data into a string variable like this program:

```
1Ø INPUT "WHAT IS YOUR NAME"; N$
2Ø PRINT "HI THERE "; N$
```

You can use as many input statements as you need to get all the values into your program. Try this:

```
NEW
1Ø INPUT "GIVE ME A NUMBER"; N1
2Ø INPUT "AND ANOTHER"; N2
3Ø PRINT "I WILL ADD THEM"
4Ø C=N1+N2
5Ø PRINT "THE ANSWER IS"; C
```

Type **RUN** and **ENTER** to see what happens. If you want to experiment some more, see **Example Program 1 "Computer Talk".**

7. Making decisions

The **IF** statement tells the computer that it has to make a decision. It does this by comparing two numbers, or arithmetic expressions or string variables. It uses special symbols to represent a condition.

The symbol > means greater than.

The symbol < means less than.

The symbol = means equal to.

Here are some conditions:

5 < 10 20 > 10

They are both true because

5 is less than 10

20 is greater than 10

A condition is either **TRUE** or **FALSE**. The **IF** statement has a condition. The computer evaluates the condition and decides that if a condition is **TRUE** it will do something for the **TRUE** condition. Take the following:

```
IF A > 9Ø THEN PRINT "YOU ARE SMART"
```

The A > 9Ø is a condition. The PRINT "YOU ARE SMART" is the thing it does for the true condition. In the above example, let's give A the value 80. Would the message be printed? No, it would not be printed because the condition is false (80 is less than 90).

Let's try a program to compute the area of a rectangle. Type:

```
NEW
1Ø INPUT "ENTER THE LENGTH";L
2Ø INPUT "ENTER THE WIDTH";W
3Ø LET A=L*W
4Ø PRINT "THE AREA IS";A
5Ø INPUT "ANOTHER? (Y/N)";R$
6Ø IF R$="Y" THEN 1Ø
7Ø PRINT "THAT'S ALL FOLKS!"
8Ø END
```

This program uses the **INPUT** statement to get values for numeric variables and print a computed result. It then prompts for a value for a string variable and makes the decision to begin again if the value typed in is a Y.

Here's another one. Type:

```
NEW
 1Ø  INPUT "GIVE ME A NUMBER";A
 2Ø  INPUT "AND ANOTHER";B
 3Ø  PRINT "ADD, SUBTRACT, MULTIPLY OR DIVIDE"
 4Ø  PRINT "TYPE IN THE FIRST LETTER OF THE"
 5Ø  PRINT "ARITHMETIC OPERATION YOU WANT"
 6Ø  PRINT "ME TO DO FOR YOU"
 7Ø  INPUT R$
1ØØ  IF R$="A" THEN C=A+B : GOTO 2ØØ
11Ø  IF R$="S" THEN C=A-B : GOTO 2ØØ
12Ø  IF R$="M" THEN C=A*B : GOTO 2ØØ
13Ø  IF R$="D" THEN C=A/B : GOTO 2ØØ
14Ø  PRINT "YOU MADE A MISTAKE"
15Ø  PRINT "TRY AGAIN"
16Ø  GOTO 3Ø
2ØØ  PRINT "THE ANSWER IS";C
3ØØ  END
```

This is a long program but it has a lot of new things in it that are worthwhile to know about. Line 70 contains an **INPUT** statement without the prompt string. Lines 100 through 130 contain two statements for the thing to do when the condition is true. One is a **LET** statement without the word LET (C=A+B) and the other is a **GOTO** statement. Both of the statements are separated from each other by the colon (: symbol). The **GOTO** statement tells the computer the number of the next statement to do. Line 140 is an error trap. It tells you that you have not followed instructions.

You can experiment some more with this program and put in a decision to start the program all over again.

What happens if you change line 300 to be 300 GOTO 10? Yes, you are right. The program continues forever. You can stop it by pressing the **BREAK** key and resume it by typing in **CONT** and press **ENTER.**

8. Looping

A loop is a set of one or more instructions. These instructions can be repeated as many times as you want. You can create a loop with a **GOTO** statement. A program like the following will run forever:

```
1Ø INPUT "GIVE ME A NUMBER";N
2Ø PRINT "YOUR NUMBER IS";N
3Ø GOTO 1Ø
```

You can modify the program to get out of a loop by placing an **IF** statement to control the loop like the program that calculates the area of a rectangle.

Another way to create a loop is through the use of the **FOR** and **NEXT** statements. These statements surround the instructions that you want to repeat. For example, type the following:

```
NEW
1Ø FOR N=1 TO 1Ø
2Ø PRINT "HELLO"
3Ø NEXT N
RUN
```

How many times does the word HELLO get printed? You can change line 20 to print the value of N every time the loop repeats. Type:

```
2Ø PRINT "N IS";N
```

We could always rewrite the program like this:

```
1Ø N=1
2Ø PRINT "N IS";N
3Ø IF N > 1Ø THEN END
4Ø N=N+1
5Ø GOTO 2Ø
```

Are they the same? We can also count by 2's. Let's bring back the previous program and change line 10. The entire program looks like this:

```
NEW
1Ø FOR N=2 TO 1Ø STEP 2
2Ø PRINT "N IS";N
3Ø NEXT N
RUN
```

The STEP 2 part changes the meaning hidden in the **NEXT N** statement to be N = N + 2 or any other number that gets put in the STEP part.

We can even count backwards. Try this:

```
NEW
1Ø FOR N=1Ø TO 1 STEP -1
2Ø PRINT N
3Ø NEXT N
4Ø PRINT "BLAST OFF"
RUN
```

9. More about groups - GOSUB ... RETURN

How much is 10 degrees Celsius in Fahrenheit? What is 100 degrees Fahrenheit in Celsius? Here's a program that gives you the answers. It uses the **GOSUB** and **RETURN** statements to create a group of instructions that can be executed from various parts of a program. Remember when you use a **GOTO** the program branches to the line number that you specify in the statement. In a **GOSUB,** the line number you specify begins a set of statements that will be executed sequentially until a **RETURN** statement is encountered. At that point the program will resume at the next statement following the **GOSUB.** Confusing? Not really but first some more background about our problem.

The formula for converting Celsius to Fahrenheit is:

F=(9/5*C)+32

The formula for Fahrenheit to Celsius is:

C=(F-32)*5/9

Now for the program! Type this:

```
NEW
 1Ø  INPUT "ENTER THE CELSIUS TEMP";C
 2Ø  GOSUB 5ØØ
 3Ø  INPUT "ENTER THE FAHRENHEIT TEMP";F
 4Ø  GOSUB 6ØØ
 5Ø  END
5ØØ  F=(C*9/5)+32
 51Ø  PRINT "THE FAHRENHEIT TEMP IS";F
 52Ø  RETURN
6ØØ  C=(F-32)*5/9
 61Ø  PRINT "THE CELSIUS TEMP IS";C
 63Ø  RETURN
```

10. Other Projects

There are many more things that you can do now that you have the BASICS down. More example programs are found in the **Appendix**. The Reference section contains a description of each of the BASIC commands found in the **PRECOMPUTER 1000.** Things like sound, special functions for random numbers and geometric functions, string manipulation and using arrays are found there.

Your enjoyment is only limited by your imagination, so have fun!

Reference section

The **REFERENCE SECTION** contains brief explanations of all the commands and statements you will use in this manual. Refer to this when you need help.

CLEAR

The **CLEAR** statement is used to assign more memory space for the string variables.

Example: 1Ø CLEAR 1ØØ

This command will assign 100 bytes of memory for strings. If the **CLEAR** command is not used, the computer will assume the number of bytes of memory for strings to be 50. The use of the **CLEAR** command only, will reserve the same number of bytes as the default value. However, if a value follows the **CLEAR** command, the computer will assign the number of bytes of that value. If you want to use more strings in your program, set this number to a larger one but, at the same time, you will have less space for your program.

CONT

Also see **STOP**

This command causes the program to resume executing after encountering the BASIC command called **STOP**. The program will carry on with the next statement after **STOP**. You can type **CONT** to resume the program after you hit the **BREAK** key.

DIM

Line # DIM array-name (array-size).

The **DIM** command reserves space for one dimensional numeric or string arrays. The array-name may be up to 6 characters long, only the first 2 characters are valid. For a string array the last character must be a $. The array can contain up to 100 elements (0-99).

Example: 1Ø DIM FRIEND$(5Ø)

1Ø DIM PRICE(19)

EDIT

Use the **EDIT** command when you want to change a statement that has been typed into memory without re-typing the entire line. Just type **EDIT** and line number and press the **ENTER** key. The statement will appear on the display. Use the **LEFT-ARROW** and **RIGHT-ARROW** keys to move the cursor. Use the **INS** and **DEL** keys to make room for a new character or to remove an unwanted one.

Example: 1Ø INPUT B (press ENTER)

EDIT 10 (press ENTER)

1Ø INPUT B

1Ø INPUT B

1Ø INPUT C

You can type in line-numbered BASIC statements in any order. The **PRECOMPUTER 1000** will sort them out for you and **LIST** them or **RUN** them starting with the lowest numbered one to the highest numbered one. It is a good idea to number your lines in increments of 10 (10, 20, 30...) or 100 (100, 200 300...). This way you have room to add more statements if you decide to change your program in the future.

You can insert an entirely new line in a program by using a line number that doesn't exist between two existing ones. You can delete an existing line-numbered statement by merely typing the line number and pressing the **ENTER** key. Of course, you can change an existing line by merely retyping the entire line.

END

Line # END

The **END** statement is used as the last command in a program. It stops the program.

Examples: 16Ø END

999 END

FOR ... TO ... STEP ... NEXT

Line# FOR variable=initial value TO final value STEP stepsize (optional)
Line#
Line#
Line#
Line# NEXT variable

The **FOR ... NEXT** statement repeats a task a set number of times without having to rewrite it. All statements between the **FOR** and **NEXT** commands are repeated based on the initial value, the final value and the stepsize. If the **STEP** portion of the command is not used it is assumed to be 1.

Example: 6Ø FOR ITEMS=1 TO 2Ø

7Ø INPUT PRICE

8Ø SUM=PRICE+SUM

9Ø NEXT ITEMS

FUNCTION

A function is a mathematical procedure which when applied to a certain value will give a new value. We call the value in brackets ( ), the argument and the new value the result.

Example: SQR is the square root function. So if we type PRINT SQR (9) (press **ENTER**) we will get the answer 3.

Example: Y=9 : X=SQR(Y) : PRINT X (press ENTER)

We will get the answer 3.

In these two examples 9 is the argument, SQR is the function and 3 is the result.

A LIST OF NUMERIC FUNCTIONS

Function	What it does
ABS (X)	Returns the absolute (positive) value of X
SGN (X)	Returns the sign of the argument X negative returns - 1 X positive returns + 1 X zero returns Ø
SQR (X)	Returns the square root of X. X cannot be negative.
LOG (X)	Gives the natural logarithm of X, i.e., the logarithm to the base e (=2.71828). The value of the argument must be greater than zero.
EXP (X)	Gives you the value e^x - i.e., the natural antilogarithm of X.
INT (X)	Gives the greatest integer which is less than or equal to X.
RND (X)	Gives a random whole number between 1 and X. If X equals zero, RND (X) returns a random number between Ø and 1. X cannot be negative.
SIN (X) COS (X) TAN (X)	The argument of the trigonometrical functions is taken to be in radians (1 radian=36Ø/2π =57.296 degrees). The range of X is -999 < (X) < 999.
ATN (X)	This gives the result of ARC TANGENT in radians.

ARITHMETIC FUNCTIONS

ABS (X)
This gives the absolute (positive) value of the argument. So ABS (-7) = 7.

Example: PRINT ABS (7 - 2 * 4) press **ENTER**

1

SGN (X)
This function will give the value of + 1 if X is positive, Ø if X is zero, and -1 if X is negative. So SGN (4.3) = 1; SGN (Ø) = Ø; SGN (-.276) = -1.

Example: A = -6 press **ENTER**

PRINT SGN (A); SGN (A-A) press **ENTER**

-1 press **ENTER** Ø

INT (X)
This converts arguments which are not whole into the largest whole number below the argument. So INT (5.9) = 5; also INT (-5.9) = -6. Note that with negative arguments, the absolute value of the result returned by INT will be greater than that of the argument.

Example: PRINT INT (-6.7) press **ENTER**

-7

RND (X)
This will produce a random number between 1 and X if X is positive.

Example: PRINT RND (19) press **ENTER**

You will get a number between 1 and 19. RND (Ø) will give you a number between Ø and 1.

Note: X cannot be negative.

STRING FUNCTIONS

We can also use functions to act on strings. Have a look at the following:

Note: From now on, the **ENTER** key sign will be deleted for simplicity. Remember to press the **ENTER** key after each line of entry.

LEN

This function computes the length of the string argument, which must be in brackets. So if you type PRINT LEN ("JOHN") the computer will return the result 4. This is telling you that there are 4 characters in the string "JOHN". Blank spaces have the value of a character. Thus if you put in spaces "J O H N" it comes out as 7 characters.

STR$

The **STR$** function changes a number argument into a string. Let us take a look at the following example and see how it works.

Example: A$=STR$ (73)

This is the same as saying

A$="73"

Here is an example program

Example:

```
1Ø  A$=STR$ (7 * 3)
2Ø  B$=A$+"BIG"
3Ø  PRINT B$
RUN
21BIG
```

VAL

VAL works like **STR$** but in reverse. It changes a string argument into a number. Look at the following short program.

Example:

```
1Ø  A$="33"
2Ø  B$="2Ø"
3Ø  C=VAL (A$+B$)
4Ø  PRINT C; C+1ØØ
RUN
332Ø    342Ø
```

SUBSTRINGS

It is also possible to get substrings of strings. A substring is, as you might guess a part of a string. For example: "**ABC**" is a substring of "ABCDE". The following three functions operate on substrings.

LEFT$ (A$, N)

This will return the substring from the leftmost of string A$, the first character to the Nth character.

Example:

```
1Ø  A$="ABCDE"
2Ø  B$=LEFT$ (A$+"FGH", 6)
3Ø  PRINT B$
RUN
ABCDEF
```

RIGHT$ (A$, N)

This will return a substring but starting from the Nth character from the end and running to the last one - the right most character in the string A$.

Example:

```
1Ø  A$="WHY"
2Ø  B$=RIGHT$ (A$+"ME", 4)
3Ø  PRINT B$
RUN
HYME
```

MID$ (A$, M, N)

This function returns a substring of the string A$ starting from the Mth character with a length of N characters.

Example:

```
1Ø  A$="ABCDEFGH"
2Ø  B$=MID$ (A$, 2, 3)
3Ø  PRINT B$
RUN
BCD
```

ASC (A$)

The **ASC** statement will return the **ASCII code** (in decimal) for the FIRST character of the specified string. Brackets must enclose the string specified. Refer to the appendix for the **ASCII code table** (Pg. 83). For example the ASCII decimal value of "X" is 88. If A$="XAB", then ASC (A$)=88.

Example:

```
1Ø  X=ASC ("ROY")
2Ø  PRINT X
RUN
82
```

CHR$ (N)

This statement works the opposite way as the **ASC** statement. The **CHR$** statement will return the string character which corresponds to the given **ASCII code**. The argument may be any number from 32 to 127 or any variable expression with an integer value within that range. Brackets must be put around the argument.

Example:

```
3Ø  PRINT CHR$ (68)
RUN
D
```

GOSUB ... RETURN

Line# GOSUB first line number of subroutine.
Line#
Line#
Line#
Line#first line number of subroutine
Line#
Line# RETURN

The **GOSUB** command tells the computer to **GOTO** another line number. After the process has been completed a **RETURN** statement is used to send the computer back to the line immediately after the **GOSUB**. These statements are used to process a series of commands that are frequently used.

Example:

```
3Ø  GOSUB 12Ø
 .
 .
other program lines
 .
 .
12Ø  PRINT "WELCOME TO THE"
13Ø  PRINT "PRECOMPUTER 1ØØØ"
14Ø  RETURN
```

GOTO

Line # GOTO line number

The **GOTO** statement is used to change the normal flow of the program (which is from lowest statement number to highest). A **GOTO** command transfers control to the specified line.

Examples:

```
1Ø  GOTO 13Ø
2ØØ  GOTO 65
```

IF ... THEN ... ELSE

In general terms, the **IF...THEN...** statement is used for CONDITIONAL BRANCHING. It uses the general form "**IF** (condition) **THEN** (action clause)". A condition is made up of an expression, a relation and an expression.

Any BASIC expression, either numeric or string, may be used, but both expressions must be the same type.

Relations or comparisons used in the **IF ... THEN** statement are the following:

= Equal to
< = Less than or equal to
< > Not equal to
> = Greater than or equal to
< Less than
> Greater than

Examples of how you can use conditionals:

IF . . . . THEN A = B
IF . . . . THEN GOTO
IF . . . . THEN GOSUB
IF . . . . THEN PRINT
IF . . . . THEN INPUT

Example: 3Ø IF X > 25 THEN 6Ø

Here if the condition X > 25 is true, the computer is told to jump to line 6Ø (Note: the **GOTO** is optional after **THEN**).

If the condition is not true, that is, if X is not greater than 25 then the computer simply carries on with the normal line number order in the program. Notice that it is not necessary to use the **ELSE** part of the command here as this is optional.

Example:
```
1Ø INPUT A, B
2Ø IF A > B THEN 5Ø
3Ø IF A < B THEN 6Ø
4Ø IF A = B THEN 7Ø
5Ø PRINT A; "IS GREATER THAN"; B: END
6Ø PRINT A; "IS LESS THAN"; B: END
7Ø PRINT A; "IS EQUAL TO"; B
8Ø END
   RUN
   ]7
   ]3
   7 IS GREATER THAN 3
```

Example: 4Ø IF P=6 THEN PRINT "TRUE" ELSE PRINT "FALSE"

In this example if P = 6 the computer will print **TRUE**. Any other value will produce a **FALSE**. In either case the computer will carry onto the next line.

It is possible for more than one statement to follow the **THEN** or **ELSE** command, a colon separates the statements.

Example: 5Ø IF A=5 THEN PRINT "TRUE": S=S-3:
GOTO 9Ø ELSE PRINT "FALSE": K=K+8

So if A equals 5 the computer will print TRUE, subtract 3 from the variable S and go to line 9Ø. If A does not equal 5 the computer will print FALSE, add 8 to the variable K and then carry on with the next normal line.

LOGICAL OPERATORS

Logical operators are used **IF ... THEN ... ELSE** and such statements where a condition is used to determine subsequent operations within the user program. The logical operators are: **AND, OR, NOT**.

For purposes of this discussion A and B are relational expressions having only TRUE and FALSE. Logical operations are performed after arithmetical and relational operations.

Operator	Example	Meaning
NOT	**NOT** A	If A is true, **NOT** A is false.
AND	A **AND** B	A **AND** B has the value true, only if A and B are both true. A **AND** B has the value false if either A or B is false.
OR	A **OR** B	A **OR** B has the value true if either A or B or both are true. It has the value false if both are false.

TRUTH TABLES

The following tables are called **TRUTH TABLES**. They illustrate the results of the previous logical operations with both A and B given for every possible combination of values.

TRUTH TABLE FOR "**NOT**" FUNCTION	
A	**NOT** A
T	F
F	T

TRUTH TABLE FOR "**AND**" FUNCTION		
A	B	A **AND** B
T	T	T
T	F	F
F	T	F
F	F	F

Note that T = TRUE and F = FALSE.

TRUTH TABLE FOR "**OR**" FUNCTION		
A	B	A **OR** B
T	T	T
T	F	T
F	T	T
F	F	F

Example:

```
1Ø  INPUT A, B, C
2Ø  IF A=B AND B=C THEN PRINT "A=B=C"
3Ø  IF (NOT A=B) OR (NOT B=C) THEN 5Ø
4Ø  END
5Ø  PRINT "A=B=C IS FALSE"
6Ø  END
    RUN
    ] 1Ø
    ] 5
    ] 7
    A=B=C IS FALSE
```

Moreover **AND, OR, NOT** can be used to manipulate numerical values. These operations are based on binary numbers with 1 and Ø representing TRUE and FALSE respectively. For example:

i) **NOT** 1=-2 — [1=binary ØØØØØØØ1 and -2=binary 1111111Ø, so it just changes the 1 to Ø and Ø to 1. In other words, TRUE (1) changed to FALSE (Ø) and FALSE (Ø) is changed to TRUE (1).]

ii) 6 **OR** 13=15 — [6=binary ØØØØ11Ø and 13=binary ØØØ11Ø1, so, with reference to the **OR** truth table, 6 **OR** 13 =15=binary ØØØ1111]

iii) 6 **AND** 13=4 — [6=binary ØØØØ11Ø and 13=binary ØØØ11Ø1, so with reference to the **AND** truth table, 6 **AND** 13=4 binary ØØØØ1ØØ]

INPUT

Line # INPUT "(optional character string)"; variable 1, variable 2, ...

INPUT allows the user to type in the value of a variable at the time the program is **RUN**. If an optional character string is used, this message will be printed before the question is asked. The type of data to be **INPUTTED** varies according to the type of the variable.

Examples:

```
35  INPUT AMOUNT
14Ø INPUT "WHAT IS YOUR NAME:"; NAME$
```

LET

Line # **LET** variable = variable expression

LET variable = variable expression

The variable expression is calculated and the result is stored under the variable. The word **LET** is optional.

Examples:
```
4Ø  LET SUM=A+B+C
25  COST=PRICE-PROFIT
```

LIST

LIST (optional line number)

LIST is used to display the active program. If the optional line number is omitted the first line of the program will be displayed. If the line is longer than 20 characters, you must use ----> to move to the right. To continue the **LISTING** press **ENTER** key. To discontinue the **LISTING** press **BREAK**.

Examples:
```
LIST
LIST 5Ø
```

NEW

NEW erases all program lines from the active program area.

PRINT

Line # **PRINT** expression and/or character string

The **PRINT** statement is used to produce output on the display panel. The **PRINT** command will print one or several items--either expressions or strings. Each item in the list should be separated by a comma or a semicolon.

Examples:
```
8Ø  PRINT NAME$; "IS COMING"
23Ø PRINT "SUM="; A+B+C
```

READ AND DATA

When it is necessary to enter a lot of information or data into the computer, using the **INPUT** statement can be very time consuming. To help out use the **READ** and **DATA** commands.

Example:

```
1Ø  DATA 1Ø, 6Ø, 7Ø, 8Ø, 9Ø
2Ø  READ A, B, C, D, E
3Ø  PRINT A; B; C; D; E
RUN
1Ø  6Ø  7Ø  8Ø  9Ø
```

The **READ** statement consists of a list of variable names with commas between each variable.

The **DATA** statement consists of a list of expressions separated by commas. These expressions can be either numeric or strings. The **READ** statement makes the computer look up the value of its variables from the **DATA** statement. When the computer goes to **READ** first it will assign the first expression from the **DATA** list. The next time it goes to **READ** it will assign the second value - And so on. If the **READ** runs out of **DATA** you will get **'? OD ERROR'** i.e. **'? OUT OF DATA ERROR.'**

RESTORE

If you want to use the same data later on in the program you can do so by using the **RESTORE** statement.

Example:

```
1Ø  DATA 1, 3, 8, 9
2Ø  READ A, B, D
3Ø  RESTORE
4Ø  READ X, Y
5Ø  PRINT A; B
6Ø  PRINT X; Y
7Ø  END
    RUN
    1  3
    1  3
```

The **RESTORE** command makes subsequent **READ** statements get their values from the start of the first **DATA** statement.
Now see if you can work out what is happening here.

Example:

```
1Ø  REM FIND AVERAGE
2Ø  DATA Ø.125, 3, Ø.6, 7
3Ø  DATA 23, 9.3, 25.2, 8
4Ø  S=Ø
5Ø  FOR I=1 TO 8
6Ø  READ N
7Ø  S=S+N
8Ø  NEXT
9Ø  A= S/8
1ØØ  PRINT A
RUN
9.52813
```

REM

Line # REM text

REM is used to add comments in your program which are ignored when the program is **RUN**.

Examples:

```
1 REM**GUESSING GAME**
2 REM TEST OF SORTING
```

RUN

RUN tells the computer to begin to perform your program beginning with the lowest statement number.

SOUND

Line # SOUND frequency (0-31), pitch (1-9)

Use **SOUND** to create tones for whatever reason. The frequency of the tone is specified by the pitch value (0-31). The duration is specified by the duration value (1-9).

Example:

```
1Ø FOR DUR=1 TO 9
2Ø FOR FREQ=Ø TO 31
3Ø SOUND FREQ, DUR
4Ø NEXT FREQ
5Ø NEXT DUR
RUN
```

Example:

```
1Ø FOR I=1 TO 8
2Ø READ X
3Ø SOUND X, 7
4Ø NEXT I
5Ø DATA 16, 18, 2Ø, 21, 23, 25, 27, 28
RUN
```

This will produce 8 notes going up the scale. In this program the variable X is the frequency and the constant 7 is the duration of the note.

It is possible to get 31 different frequencies and 9 different note durations. The following tables show which codes produce the different frequencies and durations.

By varying the notes and durations therefore, and using the tables it is possible to produce tunes of your choice.

DURATION
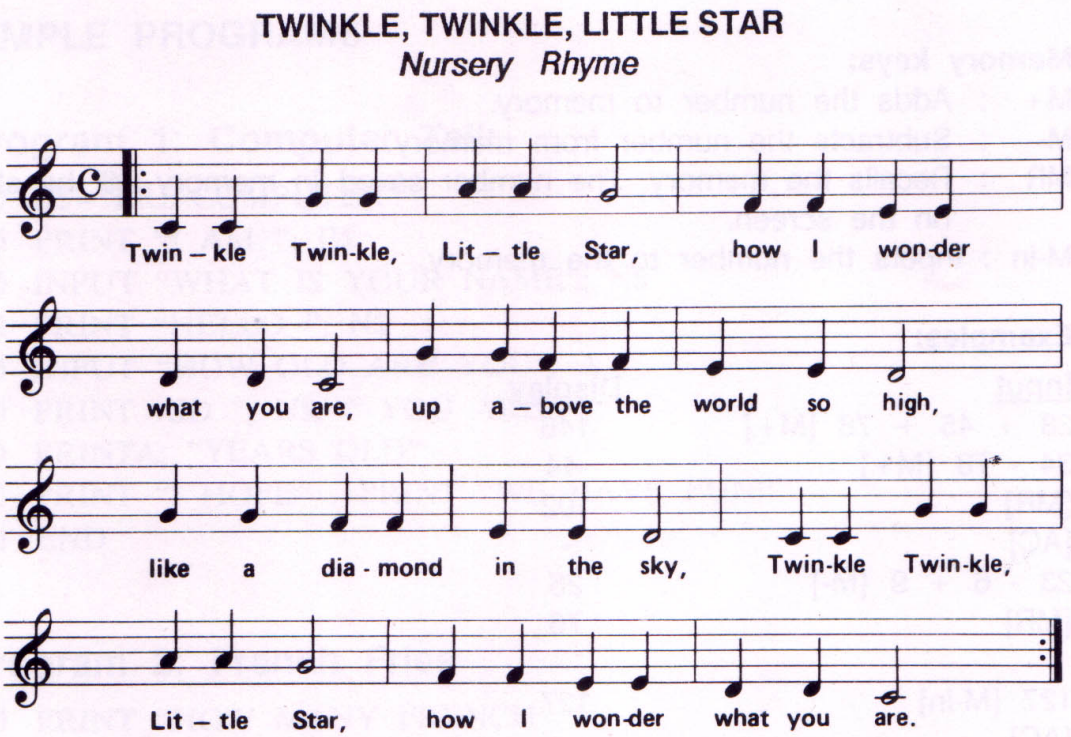
Code	Note	Note length
1		$\frac{1}{8}$
2		$\frac{1}{4}$
3		$\frac{3}{8}$
4		$\frac{1}{2}$
5		$\frac{3}{4}$
6		1
7		$1\frac{1}{2}$
8		2
9		3

FREQUENCY

Code	Pitch	Code	Pitch
Ø	rest	16	C4
1	A2	17	C#4
2	A#2	18	D4
3	B2	19	D#4
4	C3	2Ø	E4
5	C#3	21	F4
6	D3	22	F#4
7	D#3	23	G4
8	E3	24	G#4
9	F	25	A4
1Ø	F#3	26	A#4
11	G3	27	B4
12	G#3	28	C5
13	A3	29	C#5
14	A#3	3Ø	D5
15	B3	31	D#5

MUSIC

Below you can see how a musical score is transposed into Data for the computer.



TWINKLE, TWINKLE, LITTLE STAR

Example:

```
2 DATA 21, 4, 21, 4, 28, 4, 28, 4, 3Ø, 4, 3Ø, 4, 28, 6, 26, 4, 26, 4, 25, 4
4 DATA 25, 4, 23, 4, 23, 4, 21, 6, 28, 4, 28, 4, 26, 4, 26, 4, 25, 4,25, 4, 23, 6
6 DATA 28, 4, 28, 4, 26, 4, 26, 4, 25, 4, 25, 4, 23, 6, 21, 4, 21, 4, 28, 4, 28, 4
8 DATA 3Ø, 4, 3Ø, 4, 28, 6, 26, 4, 26, 4, 25, 4, 25, 4, 23, 4, 23, 4, 21, 6
1Ø FOR I=1 TO 42: READ F, D: SOUND F, D:NEXT: END
```

STOP ... CONT

Line # STOP

CONT

The **STOP** command halts the **RUNNING**of a program at that line. This is helpful for debugging. To **CONTINUE** at the next line after the **STOP** command, type **CONT** directly without a line number.

Example: 8ØØ STOP

Appendix

EXAMPLE PROGRAMS

■ Program 1: Computer Talk

```
1Ø  B$="PRE-COMPUTER"
2Ø  PRINT "I AM "; B$
3Ø  INPUT "WHAT IS YOUR NAME"; N$
4Ø  PRINT "HELLO "; N$
5Ø  INPUT "HOW OLD ARE YOU"; A
6Ø  PRINT "SO "; N$; " YOU ARE"
7Ø  PRINTA; "YEARS OLD"
8Ø  PRINT "I HOPE" : PRINT "WE HAVE FUN!!"
9Ø  END
```

■ Program 2: French Fries

```
1Ø  PRINT "HOW MANY FRENCH"
2Ø  INPUT "FRIES DID YOU EAT"; F
3Ø  PRINT "HOW MANY INCHES"
4Ø  PRINT "WAS EACH"
5Ø  INPUT "FRENCH FRY"; L
6Ø  LET T=F*L
7Ø  PRINT "YOU JUST ATE"
8Ø  PRINT "A"; T; "INCH"
9Ø  PRINT "FRENCH FRY"
```

■ Program 3: Rock'n Roll

```
1Ø  P1=16:P2=23
2Ø  D1=3:D2=2
3Ø  FOR I=1 TO 5
4Ø  SOUNDP1,D1
5Ø  SOUNDP2,D2
6Ø  SOUNDP2,D2
7Ø  NEXT
```

■ Program 4: Alarm Clock

```
1Ø  INPUT "MINUTES"; M
2Ø  INPUT "SECONDS"; S
3Ø  FOR A1=1 TO M*6Ø+S
4Ø  FOR A2=1 TO 63Ø
5Ø  NEXT
6Ø  NEXT
7Ø  SOUND31,9
```

■ Program 5: Number Guessing

```
 1Ø  PRINT "PICK FROM 1-1ØØ"
 2Ø  N=RND(1ØØ)
 3Ø  INPUT "TYPE YOUR GUESS"; G
 4Ø  IF G=N THEN 1ØØ
 5Ø  IF G<N THEN 8Ø
 6Ø  PRINT "HIGH GUESS"
 7Ø  GOTO 3Ø
 8Ø  PRINT "LOW GUESS"
 9Ø  GOTO 3Ø
1ØØ  PRINT "CORRECT!"
```

■ Program 6: Party Game

```
1ØØ  PRINT "I WILL GUESS"
11Ø  PRINT "YOUR AGE"
12Ø  PRINT "FIRST, DOUBLE"
13Ø  PRINT "YOUR AGE, THEN"
14Ø  PRINT "ADD 1Ø, TIMES IT"
15Ø  PRINT "BY 1Ø, DIVIDE IT"
16Ø  PRINT "BY 2, AND FINALLY"
17Ø  PRINT "SUBTRACT 1ØØ"
18Ø  PRINT "NOW TYPE IN THE"
19Ø  INPUT "NUMBER"; A
2ØØ  PRINT "YOUR AGE IS"; A/1Ø+5
21Ø  INPUT "AGAIN (YES/NO)"; Y$
22Ø  IF Y$="YES" THEN 1ØØ
 999 END
```

■ Program 7: Satellite

```
1Ø PRINT "STAR SATELLITE"
2Ø INPUT "HEIGHT OF SAT."; H
3Ø PRINT "DIAMETER OF"
4Ø INPUT "PLANET (M)"; D
5Ø PRINT "SPEED OF SAT."
6Ø INPUT "(M PER SEC)"; S
7Ø T=2*3.14*(D/2+H)/S
8Ø PRINT "TOTAL"; 24*6Ø*6Ø/T
9Ø PRINT "ROUNDS A DAY"
```

■ Program 8: Animal Farm

```
  5 CLEAR 1ØØ
 1Ø DIM A$(1Ø)
 2Ø FOR A=1 TO 1Ø
 3Ø INPUT "ANIMAL"; A$(A)
 4Ø NEXT
 5Ø FOR A1=1 TO 1Ø-1
 6Ø FOR A2=A1+1 TO 1Ø
 7Ø IF A$(A1)<A$(A2) THEN 11Ø
 8Ø T$=A$(A1)
 9Ø A$(A1)=A$(A2)
1ØØ A$(A2)=T$
11Ø NEXT
12Ø NEXT
13Ø FOR A=1 TO 1Ø
14Ø PRINT A$(A)
15Ø NEXT
16Ø END
```

■ Program 9: Nim

```
1ØØ  REM==INTRO==
11Ø  PRINT "YOU MAY TAKE"
12Ø  PRINT "FROM 1 TO 3"
13Ø  PRINT "STICKS AT ONCE."
14Ø  PRINT "OBJECT IS NOT TO"
15Ø  PRINT "TAKE THE LAST"
16Ø  PRINT "ONE. START WITH"
17Ø  INPUT "HOW MANY STICKS"; T
18Ø  IF T<1 THEN 17Ø
19Ø  INPUT "YOU 1ST (YES/NO)"; Y$
2ØØ  IF Y$="YES" THEN 3ØØ
21Ø  REM==COMPUTER TURN==
22Ø  C=T-1-INT((T-1)/4)*4
23Ø  IF C=Ø THEN C=1
24Ø  PRINT "I TAKE"; C
25Ø  T=T-C
26Ø  IF T>Ø THEN 3ØØ
27Ø  PRINT "I LOSE"
28Ø  GOTO 37Ø
29Ø  REM==HUMAN TURN==
3ØØ  PRINT "STICKS LEFT"; T
31Ø  INPUT "TAKE HOW MANY"; H
32Ø  IF H<1 OR H>3 THEN 31Ø
33Ø  T=T-H
34Ø  IF T>Ø THEN 22Ø
35Ø  PRINT "YOU LOSE"
36Ø  REM==END==
37Ø  INPUT "AGAIN(YES/NO)"; Y$
38Ø  IF Y$="YES" THEN 11Ø
 999  END
```

The ASCII code table

Decimal Value	Printable Character	Decimal Value	Printable Character	Decimal Value	Printable Character
32	(Space)	64	@	96	‘ (Accent)
33	!	65	A	97	a
34	“	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	‘ (Apostrophe)	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	, (Comma)	76	L	108	l
45	-	77	M	109	m
46	. (Fullstop)	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	¦
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_ (Under score)	127	■

THE ERROR MESSAGES

CN **CANNOT CONTINUE**
An attempt is made to continue a program that:
1. has halted due to an error,
2. has been modified during a break in execution, or
3. does not exist.

Ø/ **DIVISION BY ZERO**
A division by zero is encountered in an expression, or the operation of involution results in zero being raised to a negative power.

EXTRA IGNORED
More than one parameter is entered to 'INPUT' command.

ID **ILLEGAL DIRECT**
A statement that is illegal in direct mode is entered as a direct mode command. Example: INPUT

FC **ILLEGAL FUNCTION CALL**
A parameter that is out of range is passed to a math or string function. This error may also occur as the result of:
1. a negative or unreasonably large subscript
2. a negative or zero argument with LOG
3. a negative argument to SQR
4. a negative mantissa with a non-integer exponent
5. an improper argument to MID$, LEFT$, RIGHT$.

L3 **INVALID ERROR CODE**
An error message is not available for the error condition which exists.

MO **MISSING OPERAND**
The operand of some commands are missed. Example: SOUND.

NF **NEXT WITHOUT FOR**
A variable in a NEXT statement does not correspond to a previously executed unmatched FOR statement variable.

OD **OUT OF DATA**
A READ statement is executed when there are no DATA statements with unread data remaining in the program.

OM **OUT OF MEMORY**
A program is too large, has too many FOR loops or GOSUB, too many variables, or expressions that are too complicated.

OS **OUT OF STRING SPACE**
String variables have caused BASIC to exceed the amount of free memory remaining.

OV **OVERFLOW**
The result of a calculation is too large to be represented in the number format. If underflow occurs, the result is zero and execution continues without an error.

DD **REDIMENSIONED ARRAY**
Two DIM statements are given for the same array, or a DIM statement is given for an array after the default dimension of 10 has been established for that array.

REDO **REDO**
A string is assigned to a numeric variable during the execution of the INPUT command.

RG **RETURN WITHOUT GOSUB**
A RETURN statement is encountered for which there is no previous unmatched GOSUB statement.

ST **STRING FORMULA TOO COMPLEX**
A string expression is too long or too complex. The expression should be broken into smaller expressions.

LS **STRING TOO LONG**
An attempt is made to create a string more than 255 characters long.

BS **SUBSCRIPT OUT OF RANGE**
An array element is referenced either with a subscript that is outside the dimensions of the array or with the wrong number of subscripts.

SN **SYNTAX ERROR**
A line is encountered that contains some incorrect sequence of characters (such as unmatched parenthesis, misspelled command or statement, incorrect punctuation, etc.).

TM **TYPE MISMATCH**
A string variable name is assigned a numeric value or vice versa; a function that expects a numeric argument is given a string argument or vice versa.

UL **UNDEFINED STATEMENT**
A line reference in a GOTO, GOSUB or IF ... THEN ... ELSE is to a nonexistent line.