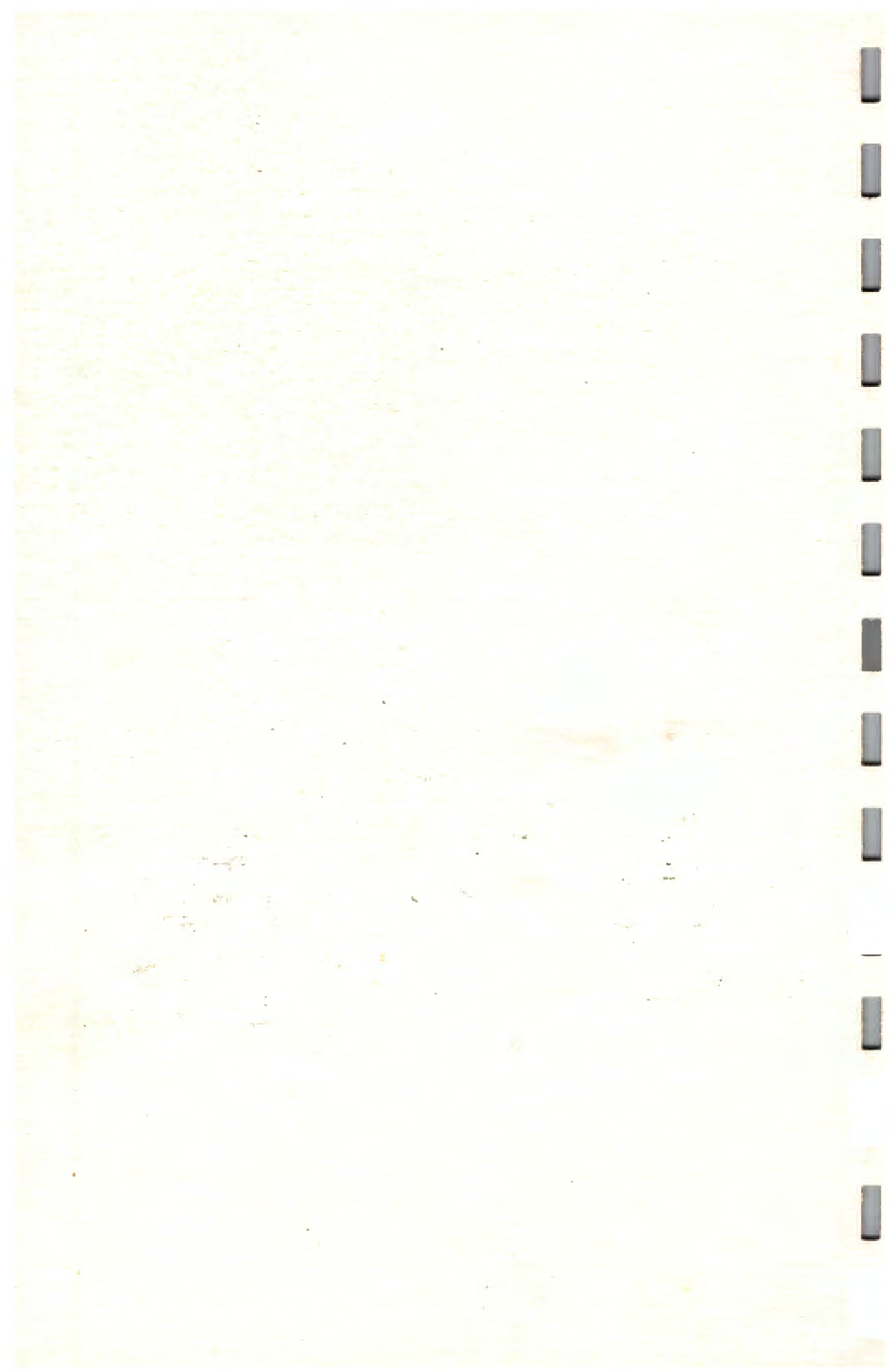
COLOR COMPUTER

BASIC
REFERENCE MANUAL

VTECH



Information in this document is subject to change without notice and does not represent a commitment on the part of Video Technology Ltd. It is against the law to copy this color computer's BASIC on cassette tape, disk, ROM, or any other medium for any purpose without the written consent by Video Technology Ltd.

© Video Technology Ltd. 1983

LIMITED WARRANTY

Video Technology Ltd. shall have no liability or responsibility to purchaser or any other person or entity with respect to any liability, loss or damage caused or alleged to be caused directly or indirectly by this product, including but not limited to any interruption of service, loss of business or anticipatory profits or consequential damages resulting from the use or operation of this product.

FIRST EDITION – 1983

All rights reserved. Reproduction or use, without express permission, of editorial or pictorial content, in any manner, is prohibited. No patent liability is assumed with respect to the use of the information contained herein. While every precaution has been taken in the preparation of this book, the publisher assumes no responsibility for errors or omissions. Neither is any liability assumed for damages resulting from the use of the information contained herein.

© Copyright 1983, Video Technology Ltd.

INTRODUCTION

This manual is intended for people who want to learn to program in BASIC on the Color Computer. With a little time and effort you will very soon discover then there is nothing very difficult about learning how to program your computer. You will be introduced to the fundamentals of BASIC and to the procedures of programming. Nothing is taken for granted. No prior knowledge is presumed. Things are explained one at a time and step by step. All you have to do is to start at the beginning and make sure you try everything as it comes up. Take your time. Understand one step before going to the next.

The key to success is to try everything. It is not enough to read about it. You must do it. You don't learn to play the piano, type or swim by reading a book. You learn by doing. Don't worry about making mistakes. It is part of the learning process. If you do make a mistake, just correct the mistake and continue. The computer doesn't worry about it, why should you? There is nothing that can be done from the keyboard that can damage your computer. Cautions are included in the text when statements that might DESTROY data files are introduced. Thus, feel free to try things out with your computer at every stage of learning.

In general, you should follow the SEQUENCE of presentation given in the text. However, Chapter 15 which discusses the use of tape storage may be read at anytime when you wish to save a program on the cassette tape. While this manual is written for one who wishes to learn to program in BASIC on this Computer, it can also serve as a general introduction to programming in BASIC on any system. Just remember that the BASIC language has many forms. There are slight differences between one implementation of BASIC and another.

Finally, this manual will not only help you to understand BASIC but it will also help you to understand the fundamentals of computer programming in general.

Have fun with your color computer.

TABLE OF CONTENTS

1. **THE COMPUTER** Page 9
   - WHAT IS A COMPUTER?
   - WHAT MAKES UP A COMPUTER SYSTEM?
   - WHAT IS A PROGRAM
   - COMPUTER LANGUAGE
   - BASIC

2. **HOW TO USE YOUR COLOR COMPUTER** Page 15
   - TO START
   - HOW TO OPERATE THE KEYBOARD
   - PRINT AND [RETURN]
   - SYNTAX ERROR
   - EDITING
   - INSERT
   - CLS
   - A LOOK AHEAD

3. **YOUR COLOR COMPUTER AS A SIMPLE CALCULATOR** Page 29
   - SIMPLE INSTRUCTIONS
   - ORDER OF NUMERIC OPERATIONS
   - BRACKETS

4. **CONSTANTS AND VARIABLES** Page 33
   - CONSTANTS
   - VARIABLES
   - LET
   - SEMI-COLONS AND COMMAS
   - COLONS

5. **PROGRAMMING** Page 39
   - INPUT
   - REM
   - NEW
   - RUN
   - LIST
   - PAUSE IN LISTING
   - DELETING A LINE

6. NUMERIC FUNCTIONS Page 47
- WHAT IS A FUNCTION
  ABS
  SGN
  SQR
  LOG
  EXP
  INT
  RND
  SIN
  COS
  TAN
  ATN

7. STRINGS Page 53
- STRING VARIABLES
- STRING FUNCTIONS
  LEN
  STR$
  VAL
  LEFT$
  RIGHT$
  MID$
  ASC
  CHR$
- STRING COMPARISONS
- INKEY$

8. COMPUTER PROGRAMMING REVISITED Page 63
- GOTO
- BREAK
- CONT
- STOP
- END
- CLEAR

9. CONDITIONS Page 69
- IF . . . THEN . . . ELSE
- CONDITIONAL BRANCHING
- LOGICAL OPERATORS
- TRUTH TABLES

10. LOOPING — Page 77
- FOR . . . TO
- NEXT
- STEP

11. SUBROUTINE — Page 83
- GOSUB
- RETURN

12. LISTS AND TABLES — Page 87
- ARRAYS
- DIM

13. READ, DATA, RESTORE — Page 93
- READ
- DATA
- RESTORE

14. PEEK AND POKE — Page 99
- PEEK
- POKE

15. STORING PROGRAM ON TAPE (CASSETTE INTERFACE) — Page 107
- SETTING IT UP
- CLOAD
- VERIFY
- CSAVE
- CRUN
- PRINT #
- INPUT #

16. GRAPHICS — Page 115
- MODES
- GRAPHIC CHARACTERS
- INVERSE
- SET
- RESET
- POINT

17. MORE COMMANDS AND INFORMATION Page 121
- PRINT @
- PRINT TAB
- PRINT USING
- INP
- OUT
- USR

18. SOUND AND SONGS Page 129
- SOUND
- SONGS

19. COLOR Page 135
- COLOR

20. THE PRINTER Page 139
- LLIST
- LPRINT
- COPY

APPENDIX Page 143
- ERROR MESSAGE
- ASCII CODE TABLE
- SUMMARY OF BASIC COMMANDS

THE COMPUTER

- WHAT IS A COMPUTER?
- WHAT MAKES UP A COMPUTER SYSTEM?
- WHAT IS A PROGRAM?
- COMPUTER LANGUAGE
- BASIC

WHAT IS A COMPUTER?

A computer is a device which performs various operations based on instructions given by the one who uses it. The computer cannot tell the user how to solve a problem. It has to be told what to do. The computer cannot think. Yet it is a very effective tool in the hands of a competent and experienced user.

A computer system consists of a number of machines or devices where operations are coordinated by a central control unit. These machines when working together are able to perform simple logical and arithmetic processes such as comparing two numbers. They can also read in information, store this information and give out results in a form understandable by us.

WHAT MAKES UP A COMPUTER SYSTEM?

Generally a computer system consists of the following units:

i) Central Processor Unit (CPU)
This can be considered the brain of the computer system. It performs operations specified in the instructions, such as arithmetic and logical operations.

ii) Memory Unit – Information and instructions given by the user or generated by the computer are stored here. This unit is inside the computer. The CPU gets information from it directly.

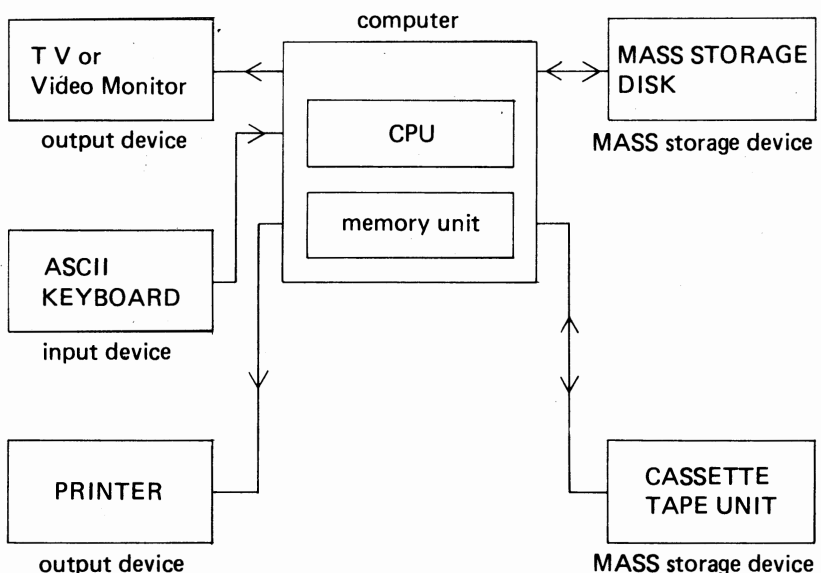
iii) Mass Storage Unit – This unit is outside the computer. It stores instructions and information given by the user or generated by the computer. The tape storage unit and the floppy disk units are examples of mass storage units. Information stored in these units has to be transferred to the memory unit before the CPU can process it.

iv) Input Device – As the name suggests this allows the user to enter instructions or information to the computer. The keyboard is an example of an input device.

v) Output Device – This receives information or results sent from the computer. Examples are the printer and the TV screen.

The input and output devices together act as a two-way communication channel between the computer user and the computer system.

Although computer systems vary in size, all practical computer systems require the above mentioned units.



Configuration of a Computer System in general

WHAT IS A PROGRAM?

A program is a set of instructions. The process of specifying a set of instructions for a computer is called programming. The individual preparing a program is called a programmer. The programmer 'inputs' a series of instructions (program) which tells the computer the steps to take to complete the task required of it.

COMPUTER LANGUAGE

There are two steps involved in preparing a program for a computer. First the programmer must know what instructions to specify and the order in which to specify them. Second, he must be able to communicate his instructions to the computer. Communication is accomplished by means of a programming 'language' which the programmer writes, and the computer 'reads'.

There are many programming languages in use today. Some are designed for very specialised applications. Others are designed for more general use. BASIC is a language in the latter category.

BASIC

BASIC, an acronym of Beginners' All-Purpose Symbolic Instruction Code, is a powerful programming language. BASIC has a simple English vocabulary, few grammatical rules and it resembles ordinary mathematical notation. To instruct your computer you must know BASIC. It will be introduced gradually and explained at each step.

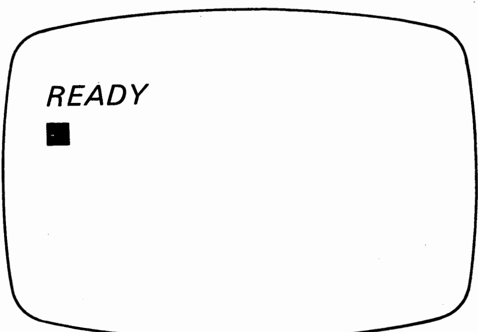
Programs written in BASIC are translated by a language translation program into a language that the central processor unit understands. This language translation program is called the BASIC Interpreter, and is contained in the main console.

HOW TO USE YOUR COLOR COMPUTER

— TO START
— HOW TO OPERATE THE KEYBOARD
— PRINT AND [RETURN]
— SYNTAX ERROR
— EDITING
— INSERT
— CLS
— A LOOK AHEAD

TO START

When you have set up your color computer and switched it on, your TV screen should look like this.

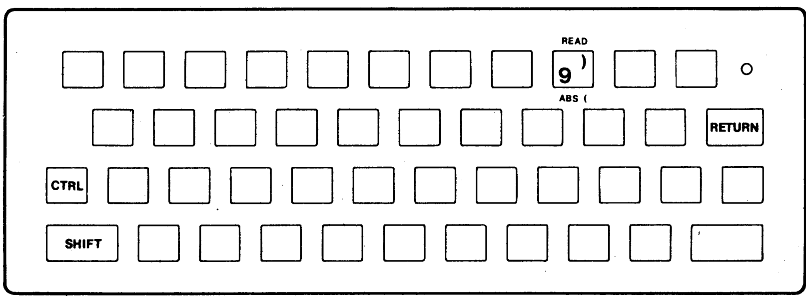


READY tells you that the computer is waiting and as the word suggests is ready to receive your instructions. The flashing green square, the CURSOR, tells you exactly where you are on the screen.

HOW TO OPERATE THE KEYBOARD?

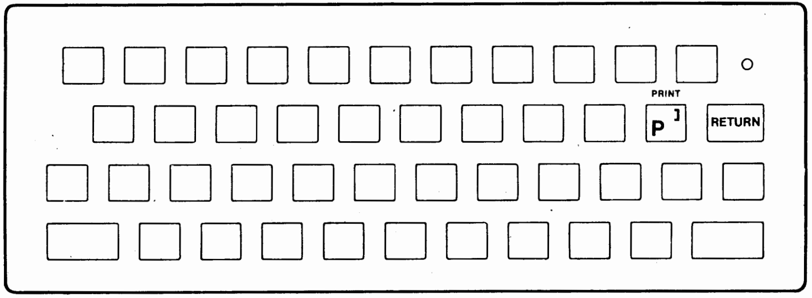
The keyboard of your computer looks complicated but it is really not too difficult to operate.

Let us take the [9] key as an example. To get the number 9 just press the key. If you wish to get **READ**, press the [CTRL] key and at the same time press the [9] key. If you wish to get the bracket on the [9] key, press the [SHIFT] key and at the same time press the [9] key. To get **ABS(**, which is below the [9] key, is a little more complicated. First press the [CTRL] key and keep pressing while you press the [RETURN] key once and then press the [9] key. **ABS(** will appear on your screen. All it takes is a little practice. If you are wondering "What would happen if I did . . . . ." Go ahead and do it and find out for yourself.



PRINT AND RETURN

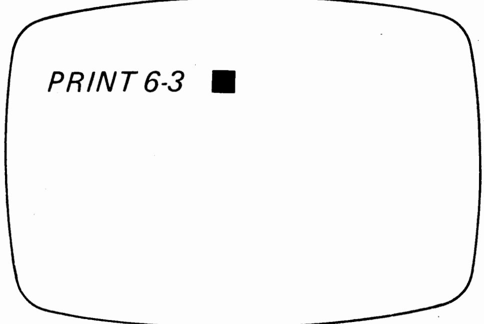
You instruct your computer using the language of BASIC. The computer can obey at once, or it can store the instructions and run them later as a program. Let us now instruct the computer to act at once. To do this, we need our first two words from the BASIC language:



When using these or any other statements you can type out each letter, e.g. **P, R, I, N, T** or you can just press the appropriate combination of keys, in this case CTRL and P to get the Keyword. Try it out on your computer.

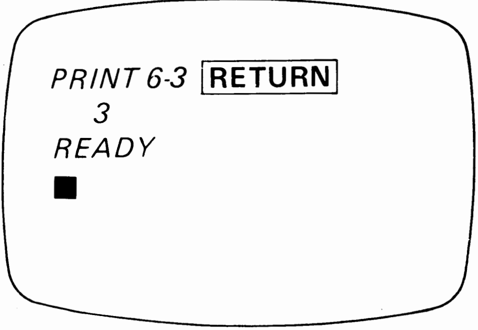
When you type **PRINT** the computer knows that it has to print what follows on the screen.

For example, type *PRINT 6-3* and the screen should look like this.



Notice that when you press a key there is a (BEEP) sound. This tells you that the key has registered and is helpful in that you do not constantly have to check the screen.

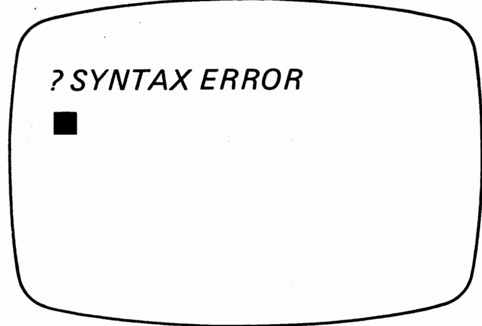
Now press RETURN and your screen should look like this.



By pressing the RETURN you have told the computer that the message is completed and you want the line executed. Remember then to press RETURN after each completed message.

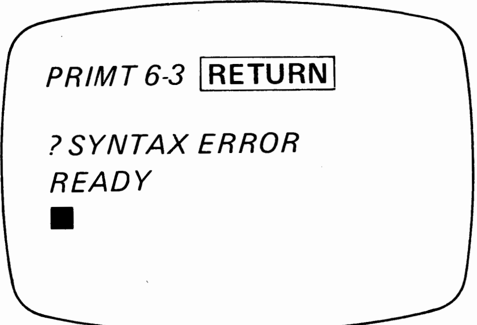
SYNTAX ERROR

You may find the following appear on your screen.



This means **SYNTAX ERROR.** A syntax error is usually due to incorrect punctuation or a typing error.

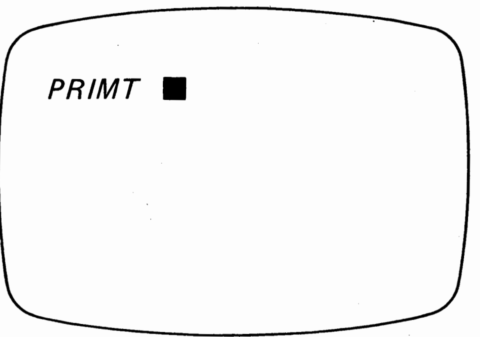
Suppose you type in *PRIMT 6-3* and then press the RETURN key your screen will look like this.



In addition to SYNTAX error there are a number of other errors that may occur, the various error types are listed in an appendix.

EDITING

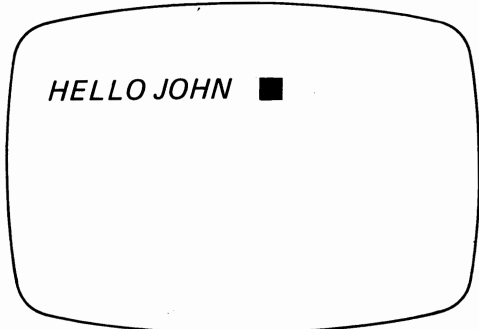
If you make a mistake while you are entering a program statement, and you press the RETURN key, you can use the CTRL and the appropriate key to move the cursor back to the wrong entry to make a correction.



When the cursor is over **T** press CTRL and RUBOUT. **T** will disappear. Move the cursor over **M.** Depress CTRL and RUBOUT again **M** will disappear. Then type in **NT** – TO GET **PRINT.**

There are four directions in which you can move the CURSOR: left, right, up, down as indicated by the black arrows on the keyboard. You will find these on the bottom right hand corner of your keyboard.

Let us take an example. If you have typed in a line, then the CURSOR is at the right side of the screen. Let us suppose you have made a mistake at the beginning of the line. You want to delete a WORD. PRESS the **CTRL** key and the **M** key. Keep both pressed until the CURSOR has moved back to where you want it. Then press the **CTRL** key and **RUBOUT** and keep pressed until the line is erased.



You want to eliminate *JOHN.* Move the CURSOR to J. Press [CTRL] and [RUBOUT]. Keep pressed and see what happens.

It is also possible to type in a letter over another letter. Let us suppose you want to change JOHN to JAMES. Position the CURSOR over O. Type in AMES. and you get JAMES.

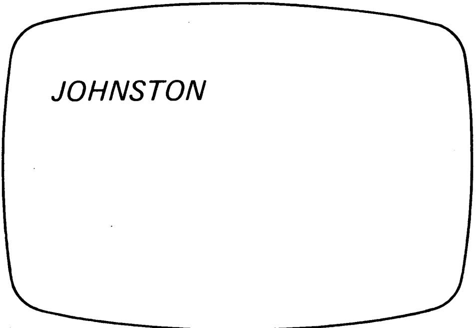
Suppose that after having typed HELLO JAMES you decide to change the name again. However this time suppose the CURSOR is at a lower line. Well all you do is to press [CTRL] and the key [•]. You will find that the CURSOR will move up to the line you want to change. Each time you press these two keys the CURSOR will move up one line. Once you have reached the line you wish to edit you can just carry on the same way as the example above.

To familiarise yourself with editing you need to experiment with it. Here as elsewhere the old adage holds true: Practice Makes Perfect.

INSERT

This allows you to insert characters starting at the position the CURSOR is in without changing what is already there. For example: you wish to insert S in JOHNTON between the N and the T.

Well you move the CURSOR to the first T, press the [CTRL] and [L] keys (which will put you into the INSERT Mode) and then type S. Your display should now look like this:



Besure to press [RETURN] after you finished editing. This will update the current line where the CURSOR is located. If you forget to do so, the original line is still kept in the program.

CLS

As you have guessed by now [CTRL] means control. If you want to clear the whole screen, press [CTRL] and [CLS] at the same time and then press [RETURN]. This will clean the screen but it will not clean the memory. The program will be wiped out from memory if you press [CTRL] and [NEW]. It will also be wiped out if you disconnect the computer.

A LOOK AHEAD

At this stage you are probably very eager to jump ahead and see what your computer is capable of. So using your newly acquired ability to try typing in these programs.

Be careful to type in everything. Do not worry about understanding the commands at the moment.

1) To get all the characters on the screen type this.

```
10  FOR I = 0 TO 1  [RETURN]
20  FOR J = 0 TO 255  [RETURN]
30  POKE 28672 + I*256 + J, J  [RETURN]
40  NEXT  [RETURN]
50  NEXT  [RETURN]
60  GOTO 60  [RETURN]
    RUN  [RETURN]
```

2) To see some of the color possibilities: try this one.

```
10 MODE (1)  [RETURN]
20 FOR Y = 0 TO 15  [RETURN]
30 FOR X = 1 TO 4  [RETURN]
40 COLOR X  [RETURN]
50 FOR K = 0 TO 31  [RETURN]
60 SET (X* 32 – 32 + K, Y)  [RETURN]
70 NEXT  [RETURN]
80 NEXT  [RETURN]
90 NEXT  [RETURN]
100 GOTO 100  [RETURN]
    RUN  [RETURN]
```

To return to Text mode type [CTRL] - [BREAK].

If you would like to change your background color you should type COLOR,1. To return to the original green just type COLOR,Ø. Note that when you do this in mode 1 the foreground colors will also change. All these will be explained later.

CHAPTER 3

YOUR COLOR COMPUTER AS A SIMPLE CALCULATOR

— SIMPLE INSTRUCTIONS
— ORDER OF NUMERIC OPERATIONS
— BRACKETS

SIMPLE INSTRUCTIONS

To use the computer as a calculator simply type **PRINT** followed by the problem and then **RETURN** . Your computer, of course, cannot only add, using +, but it can also subtract using −, multiply using *, divide using / and raise one number to the power of another using ↑. +, −, *, /, are called operations, and they operate on numbers called operands.

For example type

```
PRINT 3 ↑ 2  [RETURN]
```

and the answer 9 will appear.

ORDER OF NUMERIC OPERATIONS

When operations are combined, care must be taken to note the order in which the computer carries out the operations. The order is as follows:

1) Minus sign − used to indicate negative numbers.
2) Exponentiation starting at the left and moving right.
3) Multiplication and division (which are given the same order of precedence). Here too the computer moves from left to right.
4) Subtraction and addition moving from left to right.

Examples: A)

```
PRINT 3 ↑ 2 ↑ 2 ↑ 2  [RETURN]
6561
```

This is done by squaring 3 to get 9. Then squaring 9 to get 81, and then squaring 81 to get 6561.

B)
```
PRINT 6 * 2 + 3  [RETURN]
15
```
Here the computer first multiplies 6 x 2 and then adds 3.

C)
```
PRINT 6 + 3 * 4 + 6/3  [RETURN]
2Ø
```
First the computer carries out the multiplication and division and then adds to give 6 + 12 + 2.

BRACKETS

All operations within brackets will be carried out first before the other operations.

Example:
```
PRINT 18/(3 + 3)  [RETURN]
3
```

Where brackets are placed within brackets the innermost brackets are calculated first.

Example:
```
PRINT 2Ø/(1 + (3 ↑ 2))  [RETURN]
2
```

CHAPTER 4

CONSTANTS AND VARIABLES

— CONSTANTS
— VARIABLES
— LET
— SEMI-COLONS AND COMMAS
— COLONS

CONSTANTS

In the previous chapters we have been using constants. A constant is, of course, something which does not change and can be either positive or negative. The number 6.32 is a constant. Moving onto 6.33 just gives a different constant.

The range of a number in the computer is $-1Ø^{38} \leqslant x \leqslant 1Ø^{38}$.

The lowest positive number is $1Ø^{-38}$.

VARIABLES

A variable, as you might surmise, is something which changes. In Y = X + 3 both X and Y are variables as they have many possible values. A variable can be denoted by any letter of the alphabet, any two letters, or by a letter and a number providing the letter comes first. For example, A, AB, A6.

The variable name can be any length but only the first two characters will be recognised by the computer. For example, the computer will consider HELLO MR BLOGGS to be the same variable as HE.

A variable name cannot be any of the command words like LET or PRINT.

LET

The command **LET** can be used to assign a value to a variable. If a variable is not assigned a value it is held to be equal to zero. The variable will keep its assigned value until another **LET**, **READ** or an **INPUT** command is used to change the value.

Example:

```
LET A = 7      [RETURN]
LET B = 9      [RETURN]
PRINT A + B    [RETURN]
16
```

In BASIC the = sign does not mean the same as it usually does but tells the computer to give the value on the left hand side the same value as the right hand side.

The left hand side of the statement must always be a variable.

Have a look at the next example.

Example:

```
LET A = 2   RETURN
LET B = 3   RETURN
LET C = 2Ø   RETURN
LET A = 2 + A   RETURN
LET D = 3 + D   RETURN
PRINT A; B; C; D   RETURN
4  3  20  3
```

In the fourth line we see that A is assigned a new value of 2 plus the old value of A which was also 2 giving a total of 4.

In the fifth line we see that D is given the value 3 + D, as zero is the value given to any variable without an assigned value.

Note that in your computer it is not strictly necessary to use **LET** to assign a value to a variable, and A = 7 will carry out the same function as LET A = 7.

SEMI-COLONS, COMMAS

If more than one item is included in a **PRINT** statement the items should be separated by either a (,) or (;). Note the use of the semi-colon in the **PRINT** statement above. This causes the results to be printed immediately after each other with a space in between. When we use the semi-colon with strings there is no space.

A comma causes the result to be printed as follows. Think of your screen divided up into 2 sections of 16 characters. A comma will cause the first result to be printed at the beginning of the first section – on the left side – the second result to be printed at the beginning of the second section. The third result will go back to the first section and will be printed under the first result. If a result is longer than 16 characters it will overlap into the next section. The next result will ignore this section and start at the beginning of the following one.

COLONS

If you have more than one statement on a line you must separate them by using colons.

Example:

```
10 FOR I = 1 TO 5 : PRINT I; : NEXT RETURN
RUN RETURN
1 2 3 4 5
```

Lists of PRINT Statements

If you type

```
10 PRINT 4  RETURN
20 PRINT 6  RETURN
30 PRINT 8  RETURN
RUN RETURN
```

Your screen will show

```
4
6
8
```

Semi Colons at the end of PRINT Statements

If you type

```
10 PRINT 4; [RETURN]
20 PRINT 5; [RETURN]
30 PRINT 6; [RETURN]
RUN [RETURN]
```

Your screen will show

```
4    5    6
```

CHAPTER 5

PROGRAMMING

- INPUT
- REM
- NEW
- RUN
- LIST
- PAUSE IN LISTING
- DELETING A LINE

OK, so now let's try some simple programming. In the last few chapters we have been dealing with "immediate execution" – with the computer obeying immediately. We now want the computer to store statements so that they can be executed later on – "deferred execution".

INPUT

Have a look at this program.

```
10  REM RAISE TO THE POWER OF 3 [RETURN]
20  INPUT A [RETURN]
30  PRINT A;A ↑ 3 [RETURN]
```

The **INPUT** in line 20 asks you to assign a value to the variable A. When you run this program, a question mark "?" will display. The computer will wait until you type in a value for this variable A.

Notice that each line begins with a number. These numbers tell the computer not to obey immediately but to store away. The line number governs the order in which the line will appear on the screen. It is useful to write the numbers in tens as new lines can be later fitted into any part of the program by giving it a value of say 15 or 25. The range of possible line numbers is from 0 to 65529.

REM

The **REM** in line 10 is simply there to remind you later on of the purpose of the program. The computer will ignore it. The **REM** statements use memory space so if you are short of space you can delete **REM** statements.

The **INPUT** statement in line 20 tells the computer (when the program is being run) not to execute until the variable value(s) have been typed in.

The **PRINT** statement has already been met.

NEW and RUN

So how do we feed the program into the computer. Well first press the [NEW] key and [RETURN]. This will wipe out any old programs and variables. Remember the **NEW** command clears the memory of the computer.

Now type

```
10 REM RAISE TO THE POWER OF 3  [RETURN]
20 INPUT A  [RETURN]
30 PRINT A; A ↑ 3  [RETURN]
```

You can now run the program by pressing [RUN] and [RETURN]. The sign ? will now appear under line 30. This is the result of the INPUT statement and the computer is now waiting for you to give a value to the variable A. This value should be typed next to?.

Let's type 2 and [RETURN]

The screen will look like this.

```
10 REM RAISE TO THE POWER OF 3 [RETURN]
20 INPUT A [RETURN]
30 PRINT A; A ↑ 3 [RETURN]
RUN [RETURN]
?   2 [RETURN]
2   8
```

RUN

By pressing RUN and RETURN the whole program will be executed. If however you press RUN then a line number then RETURN the computer will start execution at the line specified.

Have a look at this program

```
10  INPUT A, B     RETURN
20  PRINT A + B    RETURN
RUN  RETURN
```

What will happen here is first of all you will get one '?' for you to put a value of A beside. On the next line you will get '??' for you to put the value of B beside. So carrying on with the program above.

```
10  INPUT A, B     RETURN
20  PRINT A + B    RETURN
RUN  RETURN
?      3  RETURN
??     6  RETURN
       9
```

LIST

If you want the whole program to be displayed in an ascending line number order then just press [LIST] and [RETURN].

Example:

```
10  INPUT A  [RETURN]
20  INPUT B  [RETURN]
30  PRINT A; B; C; A + B + C  [RETURN]
25  INPUT C  [RETURN]
```

```
LIST  [RETURN]
10 INPUT A
20 INPUT B
25 INPUT C
30 PRINT A; B; C; A + B + C
```

If you only want one line to be displayed then type

```
LIST (Line number)  [RETURN]
```

Example:

```
LIST 20  [RETURN]
20 INPUT B
```

To list part of a program say line 20 — 30 type

```
LIST 2Ø – 3Ø  [RETURN]
2Ø INPUT B
25 INPUT C
3Ø PRINT A; B; C; A + B + C
```

If you type *LIST – 3Ø* you will get the program listed up to line 3Ø from the start.

If you type *LIST 3Ø –* you will get the program listed from line 3Ø to the end.

PAUSE IN LISTING

If you have a very long program you might wish to have a look at a particular line while it is being listed. To do this just press the [SPACE] key when you wish the listing program to stop.

DELETING A LINE

To get rid of any program line just type the line number and [RETURN].

CHAPTER 6

NUMERIC FUNCTIONS

— WHAT IS A FUNCTION
- ABS
- SGN
- SQR
- LOG
- EXP
- INT
- RND
- SIN
- COS
- TAN
- ATN

WHAT IS A FUNCTION?

A function is a 'law' which when applied to a certain value will give a new value. We call the first value the argument and the new value the result.

SQR is the square root function. So if we type

```
PRINT SQR (9) RETURN
```

we will get the answer 3.

In this example 9 is the argument, **SQR** is the function and 3 is the result.

Below we give a list of the numeric functions and a brief explanation. Any function which we consider to be new to the reader will be explained in more detail afterwards. The functions will appear later on in programs so we don't give example program for each one.

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. The value of the argument must be greater than zero.
EXP (X)	Gives you the value of e^x. e = 2.71828
INT (X)	Gives the greatest integer which is less than or equal to X.

RND (X)	Gives random whole numbers between 1 and X. If X equals zero RND (X) returns random numbers between Ø and 1. X cannot be negative.
SIN (X) **COS (X)** **TAN (X)**	The argument of the trigonometrical functions is in radians. The range of X is −9999999 ⩽ (X) ⩽ 9999999.
ATN (X)	This gives the result of ARC TANGENT in RADIANS.

A FURTHER LOOK AT ABS : SGN : INT : RND

ABS (X)

This gives the absolute (positive) value of the argument. So *ABS (−7) = 7.*

Example:

```
PRINT ABS (7 - 2 * 4)  RETURN
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  RETURN
PRINT SGN (A); SGN (A - A)  RETURN
-1      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.*

Example:

```
PRINT INT (–6.7)  [RETURN]
–7
```

RND (X)

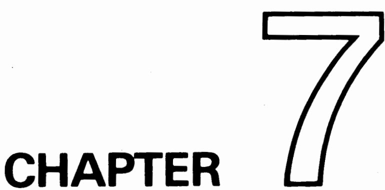
This will produce a random number between 1 and X if X is positive.

Example:

```
PRINT RND (19)  [RETURN]
```

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

X cannot be negative.



CHAPTER 7

STRINGS

- STRING VARIABLES
- STRING FUNCTIONS
  - LEN
  - STR$
  - VAL
  - LEFT$
  - RIGHT$
  - MID$
  - ASC
  - CHR$
- STRING COMPARISONS
- INKEY $

We assume that you are now familiar with the use of the RETURN key so we will not keep reminding you of it.

A string is any combination of CHARACTERS that is treated as a unit.

Strings must be enclosed in inverted COMMAS.

Example: *"HELP"*

When using the **PRINT** statement the semi-colon will not cause a space to appear between the results. They will appear immediately next to each other.

STRING VARIABLES

Any letter of the alphabet can be used as a string variable but must be followed by a $ sign. The computer accepts these characters as the variant name.

Example: *A$ = "ONE DOZEN EGGS"*

You can add strings to each other. This is called concatenation. You cannot subtract, divide or multiply.

Now let us try out this program.

```
10 A$ = "I A"
20 B$ = "M 15 YE"
30 C$ = "ARS OLD"
40 PRINT A$ + B$ + C$
   RUN
   I AM 15 YEARS OLD
```

Notice the spacing of the string characters here.

STRING FUNCTIONS

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

LEN

This function works out the length of the string 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 numbers into strings. Let us take a look at the following example and see how it works.

```
A$ = STR$ (73)
```

This is the same as saying

```
A$ = "73"
```

Here is an example program

```
10 A$ = STR$ (7 * 3)
20 B$ = A$ + "BIG"
30 PRINT B$
   RUN
   21 BIG
```

VAL

VAL works like **STR$** but in reverse. It changes strings into numbers. If only works on numbers not on operators or characters.

```
VAL ("61") = 61
```

Look at the following short program

```
10 A$ = "33"
20 B$ = "20"
30 C  = VAL (A$ + B$)
40 PRINT C; C + 100
RUN
3320   3420
```

SUBSTRINGS

It is also possible to get substrings of strings. A substring is as you might guess a part of a string. For example: "ASC" is a substrung of "ASCDE".

LEFT$ (A$, N)

This will return the substring from the leftmost CHARACTER — the first character — to the Nth CHARACTER.

Example:

```
10 A$ = "ABCDE"
20 B$ = LEFT$ (A$ + "FGH", 6)
30 PRINT B$
   RUN
   ABCDEF
```

RIGHT$ (A$, N)

This will return a substring as in the above example but starting from the Nth CHARACTER from the end and running to the last one — the right most CHARACTER in the string A$.

Example:

```
10 A$ = "WHY"
20 B$ = RIGHT$ (A$ + "ME", 4)
30 PRINT B$
   RUN
   HYME
```

MID$

MID$ (A$, M, N) returns a substring of the string A$ starting from the Mth CHARACTER with a length of N CHARACTERS.

Example:

```
10 A$ = "ABCDEFGH"
20 B$ = MID$ (A$, 2, 3)
30 PRINT B$
   RUN
   BCD
```

ASC

The **ASC** statement which is written as **ASC (A$)** where A$ is a variable string expression, 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. For example the ASCII decimal value of "X" is 88. If *A$ = "XAB"*, then *ASC (B$) = 88.*

Example:

```
10 X = ASC ("ROY")
20 PRINT X
   RUN
   82
```

CHR$

This statement works the opposite way around to the **ASC** statement. The **CHR$** statement will return the CHARACTER of the given ASCII code. The argument may be any number from Ø to 255 or any variable expression with a value within that range. Brackets must be put around the argument.

Example:

```
30 PRINT CHR$ (68)
```

STRING COMPARISONS

Relational operators can be applied to string expressions to compare the strings for equality or alphabetic precedence. As far as equality is concerned all the characters (and any blanks) must be identical.

Example:

```
10 A$ = "AA"
20 B$ = "BA"
30 IF A$ = B$ then PRINT 20
40 IF A$ < B$ then PRINT 30
50 IF A$ > B$ then PRINT 40
   RUN
   30
```

The comparisons are done by taking the value of the string characters from the table in the appendix and then comparing these values. The table gives us the value for 'A' as 65 and 'B' as 66. The program above is therefore asking for confirmation that 65 is less than 66.

If the first two CHARACTERS of a string are equal the computer will search for the third CHARACTER and do the comparison on this.

Example:

```
10 A$ = "ABC"
20 B$ = "ABD"
30 IF B$ > A$ then PRINT 40
   RUN
   40
```

The CHARACTERS compared here are C and D. The table value of C is 67 and the table value of D is 68. B$ is therefore greater than A$.

INKEY$

INKEY$ returns either a one-character string containing a character read from the keyboard or a null string if no character is pending at the keyboard. All characters are passed through to the program except for Control-C, which terminates the program.

Example:

```
10 A$ = INKEY$
20 PRINT A$;
30 GOTO 10
```

CHAPTER 8

COMPUTER PROGRAMMING REVISITED

- GOTO
- BREAK
- CONT
- STOP
- END
- CLEAR

Here are some more commands to help you write more interesting programs.

GOTO

This command tells the computer to go backwards or forwards to the line number following the GOTO statement and then to carry on executing the program from that line number.

Example:

```
10 INPUT A
20 PRINT A, A ↑ 3
30 GOTO 10
   RUN
   ?
```

If you give the value say 2 to A the computer will return the results 2 and 8. However a question mark will again appear on the screen asking you to give A another value. This procedure is the result of the GOTO statement tells the computer not to end at line 30 but to go back to line 10 and start again.

BREAK

When you get tired of putting in different values of A you can press [BREAK] next to the question mark. The computer will now stop executing the program and *BREAK IN 10* will appear on your screen. It should be noted that [BREAK] is not a command as such. You can break any continuous program by implementing [BREAK].

CONT

If however, after stopping the program execution you feel there are still some values of A you would like to try you can type **CONT**, and the computer will start to execute the program once more.

STOP

A useful statement in programming is **STOP.** This causes the program to stop at the line printed after the **STOP** statement and can help you to examine the results of the variables at various stages in the program. It is also extremely useful when it comes to locating mistakes (debugging). A liberal supply of **STOP** statements throughout a program is therefore a good idea.

You can restart the program by typing **CONT.** The program will carry on from the next line after the **STOP.**

END

The **END** statement is used to terminate execution. Unlike with **STOP** execution cannot be continued after an **END** statement.

Example:

```
10 INPUT A
20 IF A > 0 THEN PRINT "A IS POSITIVE":
   END
30 IF A < 0 THEN PRINT "A IS NEGATIVE":
   END
40 PRINT "A IS ZERO"
50 END
```

NOTICE the **STOP** statement will give you the line number when you stop. This will not happen with the **END** statement.

CLEAR

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

Example:

```
10 CLEAR 100
```

The value following is the number of bytes of memory. This value may be omitted and the computer will assign the default value. If you want to use more strings in your program, set this number to a larger one but you will have less spaces for your program.

CONDITIONS

- IF . . . THEN . . . ELSE
- CONDITIONAL BRANCHING
- LOGICAL OPERATORS
- TRUTH TABLES

IF . . . THEN . . . ELSE

As we make our way through BASIC, we find that we gain more control over the computer, that is, we are able to do more with the computer. In this chapter we are going to take a look at the "**IF . . . THEN . . . ELSE**" statement. This is, perhaps, one of the two most important programming concepts in BASIC. The other one is "**FOR . . . NEXT**". We will look at this in the following chapter.

Let us look at this example.

```
6Ø IF A$ > B$ THEN PRINT A$ ELSE PRINT B$
```

This tells the computer that if the expression A$ is greater than B$ to carry out the statement *PRINT A$* and if not to carry out the statement *PRINT B$*.

CONDITIONAL BRANCHING

A condition is made up of:

An expression, a relation and an expression.

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

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

Here are some more examples of how we can use conditionals.

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

Example 1:
```
30 IF X > 25 THEN 60
```

If the statement 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 2:
```
10 INPUT A, B
20 IF A > B THEN 50
30 IF A < B THEN 60
40 IF A = B THEN 70
50 PRINT A; "IS GREATER THAN"; B: END
60 PRINT A; "IS LESS THAN"; B: END
70 PRINT A; "IS EQUAL TO"; B
80 END
   RUN
    ?     7
   ??     3
   7 IS GREATER THAN 3
```

Example 3:

```
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.

Example 4:

```
5Ø IF A = 5 THEN PRINT "TRUE": S = S – 3:
GO TO 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, and add 8 to the variable K.

LOGICAL OPERATORS

Logical operators are used in **IF . . . THEN . . . ELSE** and such statements where 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 (1) and FALSE (Ø) values. 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 above 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

NOTE: T – TRUE
F – FALSE

Example:

```
10 INPUT A, B, C
20 IF A = B AND B = C THEN PRINT "A = B = C"
30 IF (NOT A = B) OR (NOT B = C) THEN 50
40 END
50 PRINT "A = B = C IS FALSE"
60 END
   RUN
     ?   10
    ??    5
    ??    7
   A = B = C IS FALSE
```

LOOPING

- FOR . . . TO
- NEXT
- STEP

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

Often we need the computer to perform repetitive tasks. The looping process enables us to do just this without having to type in similar information many times.

For example if we want many numbers cubed and then divided by 3 we don't have to type in various values; all we have to do is this:

Example:

```
10 FOR X = 1 TO 10
20 PRINT X; X ↑ 3/3
30 NEXT X
40 END
   RUN
    1   .333333
    2   2.66667
    3   9.00001
    4   21.3333
    5   41.6667
    6   72
    7   114.333
    8   170.667
    9   243
    10   333.333
```

From the above example we can see that the computer has cubed and divided by 3 numbers between 1 and 10. The **FOR . . . . TO** statement, therefore, stipulates the range of numbers you wish to act on.

You will notice that the increments of the numbers were one each time. The increments can be changed by using the **STEP** statement and a positive number. If a negative number follows the **STEP** statement we will get a decrement. It is also possible to use a decimal, as expression or a variable.

Example:

```
10 FOR X = 1 TO 10 STEP 2
20 PRINT X; X ↑ 3/3
30 NEXT X
40 END
   RUN
```

This will act on all the odd numbers between 1 and 10 and your screen will show the following numbers:

```
1  .333333
3  9.00001
5  41.6667
7  114.333
9  243
```

You will also notice that each loop must be closed with a **NEXT** statement. The variable name of the **NEXT** statement must be the same as the variable name of the **FOR** statement – in this case X. On your computer the variable name following the **NEXT** statement can be omitted.

Loops are very useful for writing tables as you will see from the following example.

Example:

```
10 REM TO PRINT A SINE AND COSINE TABLE
20 PRINT "SIN (X)", "COS (X)"
30 FOR X = 0 TO 2 STEP 0.5
40 PRINT SIN (X), COS (X)
50 NEXT X
60 END
   RUN

SIN (X)              COS (X)
0                    1
.479426              .877582
.841471              .540302
.997495              .0707371
.909298              .416147
```

If you do use a variable name following your **NEXT** statement and you use 2 loops you must be careful not to cross your loops.

Example:

```
     10 . . . . . . . . . .
  ┌─20 FOR X = 0 TO 10
  │ 30 . . . . . . . . . .
┌─┼─40 FOR Y = 0 TO 5
│ │ 50 . . . . . . . . . .
│ └─60 NEXT X
└───70 NEXT Y
```

This causes crossed loops and will result in a NEXT WITHOUT FOR error message. The correct way is as follows:

```
   10..........
┌─20 FOR X=Ø TO 1Ø
│  3Ø..........
│┌─4Ø FOR Y=Ø TO 5
││ 5Ø..........
│└─6Ø NEXT Y
└──7Ø NEXT X
```

This gives you the nested loop for Y inside the loop for X.

CHAPTER 11

SUBROUTINES

- GOSUB
- RETURN

GOSUB – RETURN

A program has a beginning and an end. It has a structure. This structure is made up of smaller building blocks. You may need some of these blocks or sections of the program many times in various places in the overall program. To help us deal with these similar smaller parts of the program we can use subroutines. The statements we use are GOSUB and RETURN.

```
10 PRINT "HELLO"
20 GOSUB 50
30 PRINT "GOODBYE"
40 END
50 PRINT "HOW ARE YOU?"
60 GOSUB 80
70 RETURN
80 PRINT "SEE YOU"
90 RETURN
   RUN
   HELLO
   HOW ARE YOU?
   SEE YOU
   GOODBYE
```

The lines are executed in this order 10, 20, 50, 60, 80, 90, 70, 30, 40. You can see from this example how the GOSUB statement works . . . .

. . . The **GOSUB** statement tells the computer to move on to the line number indicated, that is to the line number following **GOSUB.** However unlike with the **GOTO** command the **GOSUB** command makes the computer come back again to the statement that immediately follows the **GOSUB** statement.

The computer will carry on with the subroutine until the **RETURN** statement is met. It is the **RETURN** statement that makes the computer go back to the statement following the **GOSUB** statement.

For another example:

2Ø GOSUB 6Ø tells the computer to jump to line 6Ø. The computer will start executing at line 6Ø until it meets a **RETURN** statement. On meeting the **RETURN** statement the computer will go back and start executing at the statement following the **GOSUB** statement in line 2Ø.

Have a look at this example and see if you can work out what's happening.

```
1Ø FOR X = 1 TO 5
2Ø GOSUB 6Ø
3Ø PRINT X; S
4Ø NEXT X
5Ø END
6Ø S = Ø
7Ø FOR J = 1 TO 4
8Ø S = S + J
9Ø NEXT J
1ØØ RETURN
   RUN
   1      1Ø
   2      1Ø
   3      1Ø
   4      1Ø
   5      1Ø
```

CHAPTER 12

LISTS and TABLES

— ARRAYS
— DIM

ARRAYS AND DIM

There are two types of variables – Simple Variables and Array Variables. Up to now we have been dealing with simple variables. Let us now take a look at the Array type.

An array is an organised list of values which provide an efficient way of handling large amounts of information. The values can be either numbers or strings. To set up an array you have to give the array a name and a size. The name can be either a letter or a string e.g. A$(5).

It is easy to distinguish an ARRAY from a simple variable. The ARRAY variable is always followed by brackets containing a number. e.g. A(2), B(7), G5$(7). This number in the brackets is called a subscript.

Why Use ARRAYS?

Let us suppose you have a number of books at home – say 1ØØ books – and you want to index all your books. If we assign a variable to each book name for example:

Example:

```
1Ø A$ = "GONE WITH THE WIND"
2Ø B$ = "OLIVER TWIST"
 .
 .
 .
 .
 .
 .
1ØØØ Y$ = "BASIC PROGRAMMING"
```

This would be very inefficient and time consuming. A better way of dealing with this list is to use an ARRAY. The variable A$ will stand for the list of books. Let us look at the following example:

Example:

```
10 REM BOOK NAMES
20 DIM A$ (100)
30 FOR X = 1 TO 100
40 INPUT "WHAT NUMBER"; A$ (X)
50 NEXT X
   RUN
   WHAT NUMBER?
```

After you give a code to the question mark, 'WHAT NUMBER?' will appear again underneath. This will carry on until your list is completed.

Before you can use an ARRAY it is necessary to use the **DIM** statement. Above we have **DIM A$ (100)**. This tells the computer to reserve space for the ARRAY called A and that the ARRAY has 101 subscript variables. It is possible at a later time to sort, rearrange or print out this set of data. This type of ARRAY is called a one dimensional ARRAY and it deals with lists.

It is also possible to have a two dimensional ARRAY where we have two subscripts and we are dealing with numbers in the matrix form.

DIM stands for dimension.

Let us suppose we have 5 students doing 3 exams. The results of the exams look like this

	EXAM (1)	EXAM (2)	EXAM (3)
STUDENT 1	50%	70%	90%
STUDENT 2	63	42	36
STUDENT 3	20	62	50
STUDENT 4	70	75	84
STUDENT 5	93	82	68

These results can be recorded on the computer using a two dimensional ARRAY. We would have to start with the statement: *DIM A(5, 3).* Here 5 is the number of students and 3 is

the number of columns and in this case exams. So A(4, 1) will be 70. This is the score of the fourth student in the first exam.

It is possible to have up to a three dimensional ARRAY – DIM A(3, 6, 2). The size of each dimension is limited by the memory size of the computer so remember; A(X) is a one dimensional array variable, A(X, Y) is a two dimensional array variable, and A(X, Y, Z) is a three dimensional one.

Note that if you do not use the **DIM** statement the subscripts Ø – 1Ø are allowed for each dimension of each array used.

CHAPTER 13

READ, DATA, RESTORE

- READ
- DATA
- RESTORE

READ, DATA

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

Example:

```
10 DATA 10, 60, 70, 80, 90
20 READ A, B, C, D, E
30 PRINT A; B; C; D; E
   RUN
   10  60  70  80  90
```

The **READ** statement consists of a list of variables 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? **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:

```
10 DATA 1,3,8,9
20 READ A, B, D
30 RESTORE
40 READ X, Y
50 PRINT A; B
60 PRINT X; Y
70 END
   RUN
   1   3
   1   3
```

The **RESTORE** command makes subsequent **READ** statements get their values from the first **DATA** statement.

Now see if you can work out what is happening here.

Example:

```
 10 REM FIND AVERAGE
 20 DATA 0.125, 3, 0.6, 7
 30 DATA 23, 9.3, 25.2, 8
 40 S = 0
 50 FOR I = 1 TO 8
 60 READ N
 70 S = S + N
 80 NEXT
 90 A = S/8
100 PRINT A
    RUN
    9.52813
```

Now using our student's examinations results from the chapter on arrays (chapter 12) see how the **READ** and **DATA** commands can be used.

Example:

```
 10 CLS: DIM A(5, 3)
 20 PRINT "RESULT": PRINT
 30 PRINT TAB(8); "EX(1)   EX(2)   EX(3)"
 40 PRINT
 50 FOR J = 1 TO 5
 60 PRINT "STUDENT"; J;
 70 FOR I = 1 TO 3:READ A(J, I):PRINT A(J, I);:
    NEXT: PRINT
 80 NEXT
 90 END
100 DATA 50, 70, 90, 63, 42, 36, 20, 62, 50, 70, 75
110 DATA 84, 93, 82, 68
    RUN
```

CHAPTER 14

PEEK and POKE

- PEEK
- POKE

PEEK (address)

This **PEEK** function will return the value stored at the specified address in the memory of the computer. The value will be displayed in the form of a decimal. The value is in the range of Ø – 255.

Example:

```
3Ø A = PEEK (28672)
```

This returns the value the program has at 28672 and gives this value to A. It should be noted that the address need not be a value it can be an expression.

POKE address value, expression

The **POKE** function complements the **PEEK** function. It sends a value to the stated address location. You need therefore an address and value, and again the value has to be between Ø and 255.

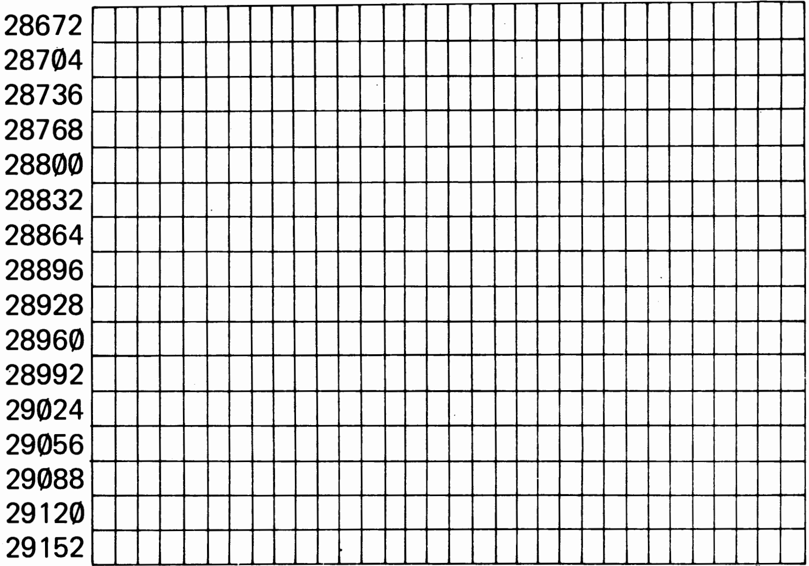
Example:

```
1Ø A = 1
2Ø POKE 29ØØØ, A
3Ø B = PEEK (29ØØØ)
4Ø PRINT B
   RUN
   1
```

When using this command we must be very careful as it can destroy your program. It is wise to save the program before you execute **POKE.** It is not recommended for newcomers without prior knowledge of what it does. You can only **POKE** to the Random Access Memory (RAM) that is to the place where the computer stores the information it wants to keep like your

BASIC program, variables, the picture for the television and the various musical notes. So may be the address you indicate may not be in the memory of the computer. The address is in the range of –32768 to +32767.

Screen locations for **POKE** and **PEEK** commands are hexidecimal 7000 – 71FF (Decimal 28672 – 29183) in TEXT mode and 7000 – 77FF (Decimal 28672 – 36863) in GRAPHIC mode. Starting location is decimal 28672.



Example:

POKE

```
10 CLS : SC = 28672
20 FOR I = 1 TO 9
30 READ A
40 POKE SC + I * 32, A
50 NEXT
60 GOTO 60
70 DATA 80, 79, 75, 69, 32, 80, 69, 69, 75
```

TYPE [CTRL] [BREAK] *to break this program*

PEEK

```
10 CLS : SC = 28672
20 PRINT "PEEK" : PRINT
30 FOR I = 0 TO 3
40 PRINT PEEK (SC + I);
50 NEXT
```

Your computer has some different character codes to most other computers. They are shown below:

NEW CHARACTERS CODE (FOR POKE & PEEK)

(A)		(A)		(A)		(A)	
Ø	@	16	P	32		48	Ø
1	A	17	Q	33	!	49	1
2	B	18	R	34	"	5Ø	2
3	C	19	S	35	#	51	3
4	D	2Ø	T	36	$	52	4
5	E	21	U	37	%	53	5
6	F	22	V	38	&	54	6
7	G	23	W	39	'	55	7
8	H	24	X	4Ø	(	56	8
9	I	25	Y	41	)	57	9
1Ø	J	26	Z	42	*	58	:
11	K	27	[	43	+	59	;
12	L	28	\	44	,	6Ø	<
13	M	29	]	45	—	61	=
14	N	3Ø	↑	46	.	62	>
15	O	31	←	47	/	63	?

(A) 1ST COL – CHARACTER CODE
2ND COL – ASCII REPRESENTED

FOR A = Ø – 63
gives normal characters.

By adding an offset
A = Ø – 63 (+ 64) ⇒ 64 – 127
gives inverse characters.

(A)		(A)	
128		136	
129		137	
13Ø		138	
131		139	
132		14Ø	
133		141	
134		142	
135		143	

FOR A = 128 – 255 the codes are divided into 8 (graphic characters) groups each with different color code. (The shaded area has color.)

FOR A = 128 – 143 GREEN
= 144 – 159 YELLOW
= 16Ø – 175 BLUE
= 176 – 191 RED
= 192 – 2Ø7 BUFF
= 2Ø8 – 223 CYAN
= 224 – 239 MAGENTA
= 24Ø – 255 ORANGE

CHAPTER 15

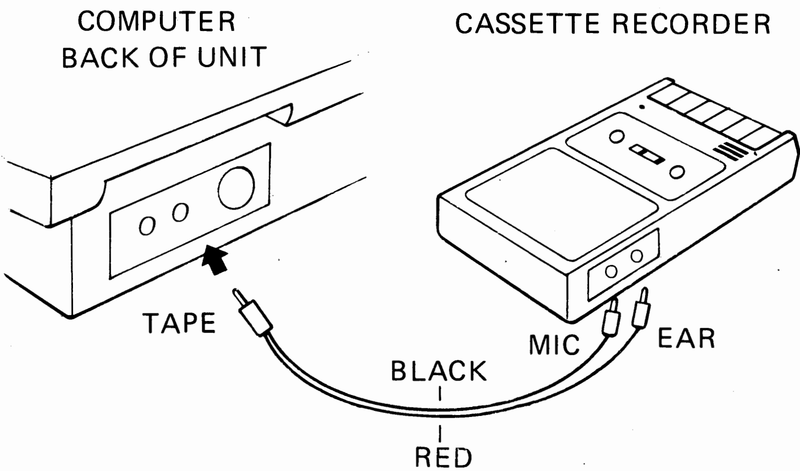
STORING PROGRAM ON TAPE
(CASSETTE INTERFACE)

— SETTING IT UP
— C LOAD
— VERIFY
— C SAVE
— C RUN
— PRINT #
— INPUT #

You may have developed some programs which you want to retain. It is too much trouble to type in a program, especially if it is long, every time you want to use it. This problem is easily solved. You can store your programs on tape and whenever you need them call them into memory.

SETTING IT UP

To do this you need an ordinary cassette tape recorder, a cassette tape and interconnecting cords. Connect the recorder as shown in the picture.



You need to be familiar with three commands, namely, **CSAVE, CLOAD** and **VERIFY.**

You have received a cassette tape with your computer. On one side of this tape there is a program and the other side is blank. It is suggested that you start by loading this program into the computer.

There is a file name for each program on the tape. A file name is a "must" for saving a program but not absolutely necessary for loading and verifying a program.

The file name can be one to sixteen characters in length. The first character must be a letter; the rest can be any character.

For our purpose saving a program means transferring a program that you have typed in, to a tape.

Verifying a program means checking to make sure that the program on the tape is the same as the program in the computer.

Loading a program means transferring a program from the tape to the computer.

CLOAD "FILE NAME"

The procedure for loading the program from the tape to the computer is as follows:

1. Load the tape containing the required program into the recorder.
2. Rewind the tape to the start of the required program.
3. Type the COMMAND *CLOAD "FILE NAME".*

   BE SURE NOT TO PRESS THE RETURN KEY
4. Press the play button on the recorder.
5. Press the RETURN key.
6. If the computer finds no program on the tape, the statement **WAITING** is displayed on the screen. If you want the computer to come out of WAITING, PRESS the CTRL BREAK keys before stopping the cassette recorder.
7. If the incoming program has the file name which does not match with the specified one, then the statement *'FOUND T: FILE NAME'* appears and the program is skipped.
8. The desired program is loaded with the statement *'LOADING T: FILE NAME'* appears.
9. When the statement READY is displayed, press the STOP button on the recorder.

Let us suppose there are three programs on the tape and you have given them file names: PROGRAM 1, PROGRAM 2, PROGRAM 3. You want PROGRAM 3 and it is at the end of the tape. You can type: *CLOAD "PROGRAM 3".* Then your screen will show:

```
CLOAD "PROGRAM 3"
WAITING
FOUND T: PROGRAM 1
FOUND T: PROGRAM 2
LOADING T: PROGRAM 3
READY
```

If you know the location of PROGRAM 3 and you set the tape at the beginning of this program the screen will show:

```
CLOAD "PROGRAM 3"
WAITING
LOADING: PROGRAM 3
READY
```

NOTE T: stands for TEXT file.

VERIFY "FILE NAME"

To verify a program on tape the procedure is as follows:

1. List the program in the computer to make sure that the program still exists.
2. Type the COMMAND *VERIFY "FILE NAME"*

   BE SURE NOT TO PRESS THE [RETURN] KEY.
3. Press the play button on the recorder.
4. Press the [RETURN] key. The flashing CURSOR will disappear and the verifying begins.

Example:

VERIFY "PROGRAM 2"
WAITING
FOUND T: PROGRAM 1
LOADING T: PROGRAM 2
VERIFY OK
READY

5. The **'OK'** statement tells us that the program on the tape is the same as the program in the computer.
6. If the verifying is incorrect, the statement **'Verify Error'** will be displayed on the screen. This statement shows that the program on the tape is different from the one in the computer. In this case the user should store the program and verify it once again.

You can verify that there is a program on the tape by listening. If there is a program on the tape and you run it on the cassette recorder, the recorder will give out a distinctive sound.

CSAVE "FILE NAME"

If you wish to save a program, make sure that you use a good quality tape. The quality of the tape can affect the quality of your recording.

It is important to set the volume of the cassette recorder within a proper range. This range will vary from one recorder to another. The tone should be set to MAXIMUM level.

The procedure for storing/saving a program is as follows:

1. Type in the complete program. It is advisable to use a short program at the start. Longer programs can be saved once you have achieved success with a short program.
2. Type in the COMMAND *CSAVE "FILE NAME"*
Remember a File Name is a "MUST" for saving a program.
BE SURE NOT TO PRESS THE RETURN KEY

3. Load the recorder with a good quality tape.
4. Press the PLAY and RECORD buttons on the recorder.
5. Press the RETURN key.

   The flashing CURSOR will disappear and the storing begins.
6. When the flashing CURSOR reappears, the storing is completed.
7. Press the STOP button on the recorder.

The program that you typed in is now stored on a tape. To make sure that it is stored the user may verify this for himself.

CRUN "FILE NAME"

One more powerful COMMAND **'CRUN'** can be used. This command is similar to **'CLOAD'** except that the loaded program will start execution automatically after the loading is completed.

The four cassette interface COMMANDS, **CSAVE, VERIFY, CLOAD** and **CRUN** help the user to save his programs, verify them, and get them back into the computer to execute. The user should pay attention to the volume level of the recorder. Cassette interface is a means of inexpensive MASS STORAGE.

There are two more COMMANDS that the user should become familiar with. There are:

PRINT

PRINT # "FILE NAME", item list sends the values of the specified variables or data onto a cassette tape. It is understood that the recorder must be properly connected and set in record mode when this statement is executed.

INPUT

INPUT # "FILE NAME", item list.

Inputs the specified number of values stored on cassette and assigns them to the specified variable names.

Example:

```
10 PRINT # "KAM", 1, 2, 3, 4, 5
   RUN
```

The data 1 to 5 are saved in the data file "KAM". BE SURE to put your cassette recorder in RECORD mode BEFORE execution.

Example:

```
10 INPUT # "KAM", A, B, C, D, E
20 PRINT A; B; C; D; E
   RUN
   FOUND D: KAM
   1  2  3  4  5
```

The data in the data file "KAM" are assigned to the variables A to E. BE SURE to put your cassette recorder in PLAY mode BEFORE execution.

NOTE D: stands for DATA file.

CHAPTER 16

GRAPHICS

- MODES
- GRAPHIC CHARACTERS
- INVERSE
- SET
- RESET
- POINT

MODES

There are 2 different modes of screen display. Once the power is turned on, the computer is in the normal Text mode, **MODE (Ø)**, with 32 characters multiplied by 16 lines. This displays 64 x 32 Dots with 9 colors. It is best used when some low resolution graphics and text are required as the video output is more attractive and appealing.

The user can switch to the High Resolution Graphic mode, which has 128 x 64 Dots and 8 colors by making use of the **MODE (1)** command. This mode provides extra-fine graphics and is very suitable for games purposes.

To get back to normal Text mode you should use the **MODE (Ø)** command.

GRAPHIC CHARACTERS

There are 16 graphic characters in your computer which can be typed by pressing the **SHIFT** key and any of the corresponding keys. These characters are useful for drawing pictures and graphs.

Have a look at the program beneath to see how you can use these characters.

Example:

```
1Ø REM COLOR
2Ø FOR I = 1 TO 52
3Ø FOR J = 1 TO 8
4Ø COLOR J
5Ø PRINT "■";
6Ø NEXT
7Ø NEXT
8Ø GO TO 8Ø
```

This example uses the graphic character shown in the program to plot some graphics. This example also shows all the 8 different colors in the TEXT mode by using graphic character.

This example only uses one graphic character. For the rest of them, you may observe your keyboard to see where they are located.

INVERSE

The inverse characters can be produced by simply pressing the [CTRL] and [INVERSE] keys. Your computer will remain in the inverse mode until the [CTRL] and [INVERSE] or [RETURN] key is pressed.

SET (X, Y)

On your computer this is used when dealing with colors. It plots a dot at a specified location on the screen which is determined by the values of X and Y. The value of X can be from Ø to 127 and the value of Y can be from Ø to 63.

RESET (X, Y)

This is used to wipe out a point switched on by the **SET** command. The X and Y values determine the location of the point to be wiped out. This is done by making the point the same color as the background.

POINT (X, Y)

This tells you if a specific point has been fixed by the **SET** command. Usually it is used with the **IF-THEN-ELSE** statement.

Like this:

```
80 SET (40, 20) : IF POINT (40, 20) THEN
   PRINT "YES" ELSE PRINT "NO"
```

Here is another example of the use of **SET** and also one using **POINT.**

Example of **SET** and **RESET**:

```
10 MODE (1) : COLOR 2
20 FOR I = 0 TO 31
30 SET (I, I/2)
40 NEXT
50 FOR I = 0 TO 31
60 RESET (I, I/2)
70 NEXT
80 MODE (0)
```

This example will plot a diagonal line on the screen and then rub this line out. When this program is completed, the computer will return back to **MODE (0)**, that is TEXT mode.

NOTE: If you want to keep the computer in graphic mode, replace line 80 by: *80 GO TO 80.*

Example of **POINT**:

```
 1Ø MODE (1) : DIM A(4)
 2Ø FOR I = 1 TO 4
 3Ø COLOR I
 4Ø SET (I, I)
 5Ø A(I) = POINT (I, I)
 6Ø NEXT
 7Ø MODE (Ø)
 8Ø FOR I = 1 TO 4
 9Ø PRINT A(I)
1ØØ NEXT
```

This example will plot points on the screen with four colors. This command **POINT** will check the color of that point and hold this information in the array A(I). Afterwards, the program will print out the color of that point checked by the command **POINT.**

CHAPTER 17

MORE COMMANDS AND INFORMATION

— PRINT @
— PRINT TAB
— PRINT USING
— INP
— OUT
— USR

PRINT @

This command causes the output to be produced at a particular point on the screen. When dealing with the **PRINT @** command consider your screen to be a grid divided up into 32 x 16 spaces. Therefore there are 512 possible positions. The command takes this form

```
PRINT @ position, item list.
```

The position can be a number, variable or arithmetic expression. The value must be between Ø and 511.

Example:

```
6Ø PRINT @ 6Ø, 6ØØ;
```

Notice the use of the semi-colon at the end of the statement. This stops the rest of the line from being rubbed out.

PRINT TAB (expression)

This command works in very much the same way as the TAB on a typewriter. In this case it moves the cursor to a particular point on a line. The value of the expression must be between Ø to 255 inclusive. If it is over 63 the cursor will move to the position that exceeds the maximum integer multiples of 64.

Example:

```
4Ø PRINT TAB (6); 1; TAB (2Ø); 1
   RUN
   1                             1
```

PRINT USING string; item list

This command is useful in allowing you to state how you want your lines printed. It is very useful for reports and tables.

It takes the form of

```
PRINT USING string; value or string
```

This string or value can be either a variable or constant. What will happen is that the string or value to the right of the semi-colon will be inserted as specified by the field specifiers, the preceding string.

A) "!" This specifies that only the first character in the given string is to be printed.

Example:

```
1Ø  A$ = "ASDF"
2Ø  PRINT USING "!"; A$
RUN
A
```

B) "#" A number sign is used to represent each digit position. Digit positions are always filled. If the number to be printed has fewer digits than positions specified, the number will be right justified (perceded by spaces) in the field.

"." A decimal point may be inserted at any position in the field. If the format string specifies that a digit is to precede the decimal point, the digit will always be printed (as Ø if necessary). Numbers are rounded as necessary.

Example:

```
PRINT USING "##.##"; .78
Ø.78
```

C) "+" A plus sign at the beginning or end of the format string will cause the sign of the number (plus or minus) to be printed before or after the number.
"–" A minus sign at the end of the format field will cause negative numbers to be printed with a trailing minus sign.

Examples:

```
PRINT USING "+##.##"; –68.95
–68.95
PRINT USING "##.##–"; –68.95
68.95–
```

D) "**" A double asterisk at the beginning of the format string causes leading spaces in the numeric field to be filled with asterisks. The ** also specifies positions for two more digits.

Example:

```
PRINT USING "**#.#"; –Ø.9
*–Ø.9
```

E) "$$" A double dollar sign causes a dollar sign to be printed to the immediate left of the formatted number. The $$ specifies two more digit positions, one of which is the dollar sign. The exponential format cannot be used with $$. Negative numbers cannot be used unless the minus sign trails to the right.

Example:

```
PRINT USING "$$###.##"; 456.78
$456.78
```

"**$" The **$ at the beginning of a format string combines the effects of the above two symbols. Leading spaces will be asterisk-filled and a dollar sign will be printed before the number. **$ specifies three more digit positions, one of which is the dollar sign.

F) "," A comma that is to the left of the decimal point in a formatting string causes a comma to be printed to the left of every third digit to the left of the decimal point. A comma that is at the end of the format string is printed as part of the string. A comma specifies another digit position.

Example:

```
PRINT USING "####,.##"; 1234.5
1,234.5Ø
```

G) "%" If the number to be printed is larger than the specified numeric field, a percent sign is printed in front of the number. If rounding causes the number to exceed the field, a percent sign will be printed in front of the rounded number.

Examples:

```
PRINT USING "##.##"; 111.22
%111.22
PRINT USING ".##"; .999
%1.ØØ
```

INP (I)

This returns the byte read from port I. I must be in the range Ø to 255. INP is the complementary function to the **OUT** statement (see below).

Example:

```
1ØØ A = INP (255)
```

OUT I, J

This sends a byte to a machine output port. I and J are integer expressions in the range Ø to 255. I is the port number and J is the data to be transmitted.

Example:

```
1ØØ OUT 32,1ØØ
```

USR (X)

This calls the user's assembly language subroutine with the argument X. This subroutine could be taken from tape or made by **POKE**ing instruction into memory locations. It is advised to take great care when using this function as it can have disastrous effects on any stored program.

Example:

```
11Ø A = USR (B/2)
```

CHAPTER 18

SOUND AND SONGS

- SOUND
- SONGS

SOUND

Another interesting feature of the computer is its ability to produce sound. Here is an example program.

Example:

```
10 FOR I = 1 TO 8
20 READ X
30 SOUND X, 7
40 NEXT
50 DATA 16, 18, 20, 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 tables below show how which codes produce the different frequencies and duration.

By varying the notes and duration therefore, and using the tables below, 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

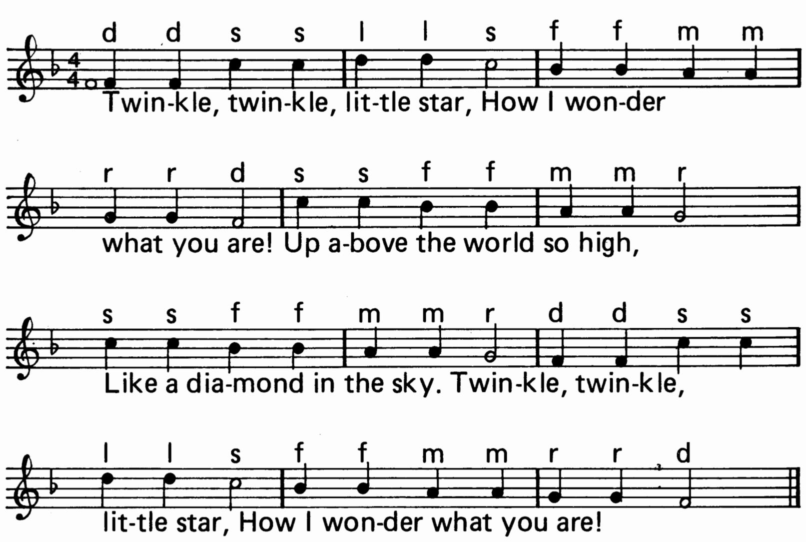
Code	Pitch
Ø	rest
1	A2
2	A#2
3	B2
4	C3
5	C#3
6	D3
7	D#3
8	E3
9	F
1Ø	F#3
11	G3
12	G#3
13	A3
14	A#3
15	B3
16	C4
17	C#4
18	D4
19	D#4
2Ø	E4
21	F4
22	F#4
23	G4
24	G#4
25	A4
26	A#4
27	B4
28	C5
29	C#5
3Ø	D5
31	D#5

SONGS

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

TWINKLE, TWINKLE, LITTLE STAR
Nursery Rhyme

Key F



TWINKLE, TWINKLE, LITTLE STAR

```
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
```

CHAPTER 19

COLOR

— COLOR

COLOR

Your computer, as you already know, is capable of producing different colors. With **COLOR I, J,** I is the foreground color – from 1 to 8 and J is the background color – from Ø to 1.

In MODE (Ø);

Code	Color
1	Green
2	Yellow
3	Blue
4	Red
5	Buff
6	Cyan
7	Magenta
8	Orange

The background color can be either green (Ø) or orange (1). To change the background color just type color, 1. To get back to the original just repeat using Ø instead of 1.

In MODE (1)

With background color is green (Ø)

Color Code	Color
1	green
2	yellow
3	blue
4	red

and with background color buff (1).

Color code	Color
5	buff
6	cyan
7	magenta
8	orange

Your computer will also accept **COLOR I** or **COLOR J.** It will remain in the defined color mode until another **COLOR** command is executed.

```
10 FOR I = 0 TO 15
20 FOR J = 0 TO 31
30 COLOR (I/2) + 1
40 PRINT "■";
50 NEXT : NEXT
60 FOR I = 1 TO 1000 : NEXT
70 END
```

Note: By pressing [SHIFT] and [J] keys together you get the character ■.

This will display the eight colors that your computer is capable of production in this mode.

The colors in your computer are very suitable for graphic and game purposes especially in **MODE (1).**

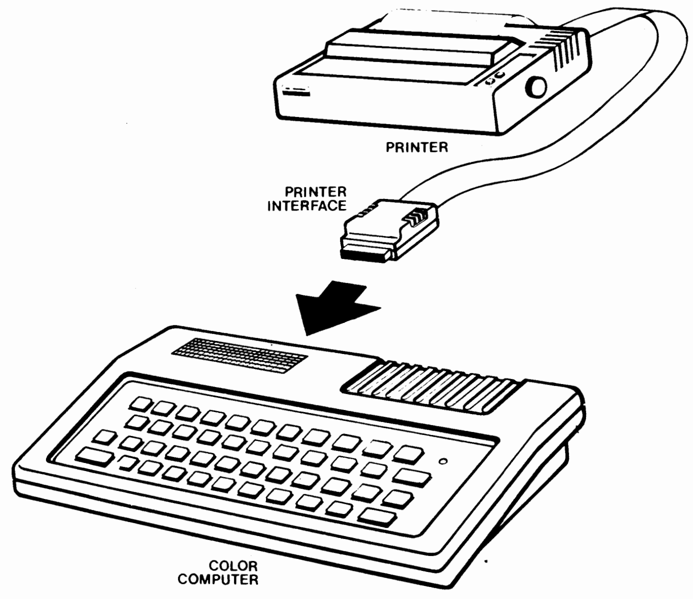
CHAPTER 20

THE PRINTER

- — L LIST
- — L PRINT
- — COPY

To further expand the capabilities of your computer you can acquire a PRINTER. This can be attached to your computer by means of a PRINTER INTERFACE. If you decide to acquire a PRINTER, you will receive a separate leaflet containing detailed operating instructions.

SETTING UP THE PRINTER



To operate the computer successfully you need to be familiar with the following: **LLIST, LPRINT, COPY.**

LLIST

This performs a similar function in relation to the PRINTER as **LIST** does in relation to the TV screen. **LLIST** outputs to the PRINTER.

LPRINT

This command (and statement) is similar to **PRINT. LPRINT** is used with the PRINTER.

COPY

The PRINTER will print out what you see on the screen to the PRINTER. You can stop the PRINTER by pressing the **CTRL** **BREAK** key.

This command can only be used for SEIKOSHA GP-1ØØ and SEIKOSHA GP-100A PRINTERS.

APPENDIX

- ERROR MESSAGE
- ASCII CODE TABLE
- SUMMARY OF BASIC COMMANDS

ERROR MESSAGE

- NEXT WITHOUT FOR
- SYNTAX ERROR
- RETURN WITHOUT GOSUB
- OUT OF DATA
- FUNCTION CODE
- OVERFLOW
- OUT OF MEMORY
- UNDEFINED STATEMENT
- BAD SUBSCRIPT
- REDIMENSIONED ARRAY
- DIVISION BY ZERO
- ILLEGAL DIRECT
- TYPE MISMATCH
- OUT OF SPACE
- STRING TOO LONG
- FORMULA TOO COMPLEX
- CAN'T CONTINUE
- MISSING OPERAND
- BAD FILE DATA
- DISK COMMAND

ASCII CODE TABLE

ASCII CODE	CHARACTER
32	(Space)
33	! (exclamation point)
34	" (quote)
35	# (number or pound sign)
36	$ (dollar)
37	% (percent)
38	& (ampersand)
39	' (apostrophe)
4Ø	((open parenthesis)
41	) (close parenthesis)
42	* (asterisk)
43	+ (plus)
44	, (comma)
45	– (minus)
46	. (period)
47	/ (slant)
48	Ø
49	1
5Ø	2
51	3
52	4
53	5
54	6
55	7
56	8
57	9
58	: (colon)
59	; (semicolon)
6Ø	< (less than)
61	= (equals)
62	> (greater than)
63	? (question mark)
64	@ (at sign)
65	A
66	B
67	C
68	D
69	E
7Ø	F
71	G
72	H
73	I
74	J
75	K
76	L
77	M
78	N
79	O
8Ø	P
81	Q
82	R
83	S
84	T
85	U
86	V
87	W
88	X
89	Y
9Ø	Z

SUMMARY OF BASIC COMMANDS

Functions:

1) Arithmetic operators

   +, –, *, /, ↑

2) Relational operators

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

3) Arithmetic functions:

   SQR – Square root
   INT – Integer part
   RND – Random number
   ABS – Absolute magnitude
   SGN – Sign
   COS – Cosine
   SIN – Sine
   EXP – e^x
   TAN – Tangent
   LOG – Natural logarithm
   ATN – Arc tangent

4) String functions:

   LEN – Length
   STR$ – String of numeric argument
   VAL – Numeric value of string
   ASC – ASCII value
   CHR$ – Character
   LEFT$ – Left characters
   MID$ – Middle characters
   RIGHT$ – Right characters
   INKEY$ – Check keyboard

5) Logical operators

AND
OR
NOT

Relation and logical expressions have value 1 if true, Ø if false.

6) Graphics and sound functions:

CLS – Clear screen
SET – Plot a point
RESET – Clear a point
POINT – Return the color code
COLOR – Set color
SOUND – Produce tone of different frequency and duration
MODE – Select graphic or text

7) Program statements

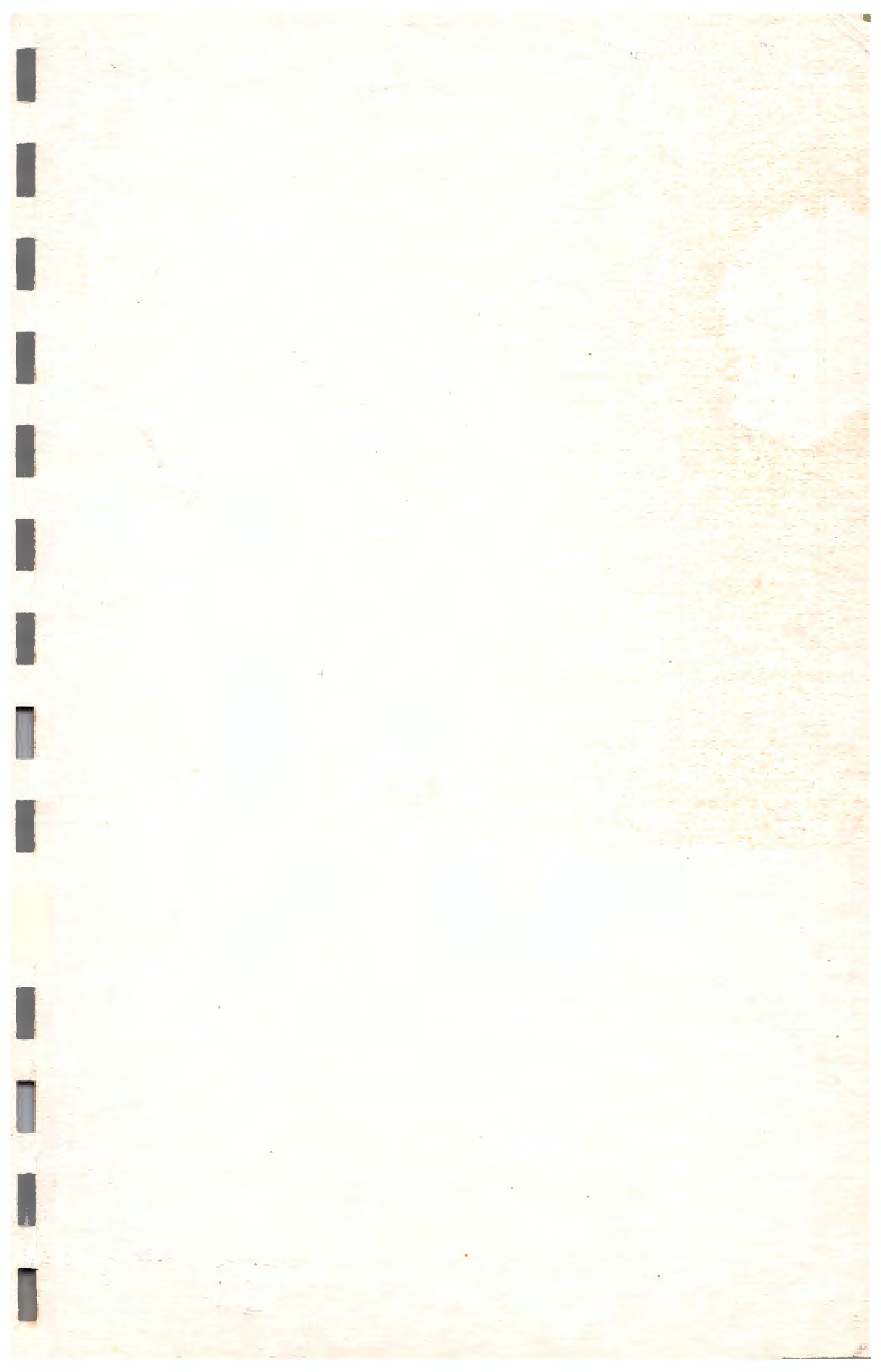
DIM – Dimensions
STOP
END
GOTO
GOSUB
RETURN
FOR . . . TO . . . STEP
NEXT
REM
IF . . . THEN . . . ELSE
INPUT
PRINT
PRINT TAB
PRINT USING
PRINT @
LET
DATA
READ
RESTORE

8) Commands:

LIST
RUN
NEW
CONT
VERIFY – Check whether program on tape and memory are equal
CLOAD – Load program on tape
CSAVE – Save program on tape
CRUN – Load program on tape and run
CTRL BREAK – To halt program

9) Other Statements

PEEK – Return the value stored at the location specified
POKE – Load a value into a specified location
LPRINT – Print on line printer
LLIST – List on line printer
INP – Return the contents read from ports
OUT – Send values to ports
COPY – Copy the content on screen to printer
USR – Call the user's assembly language subroutine



COLOR COMPUTER

BASIC
REFERENCE MANUAL

© 1983 VTL
MADE IN HONG KONG
91-0144-02