Programming Without Stress

A book introducing computer programming for students in the numerate disciplines and freelancers
by

Akinola A. Adeniyi

2004

The free online version

CHAPTER ONE

CHAPTER ONE

INTRODUCTION

That Computer is changing the world is no longer news. What comes to the curious mind is “how do I take part in the change?” One can choose to be a computer user or maker or both. By computer user I mean a person that makes use of facilities provided by the computer maker such as software or hardware. The computer maker in this context is the person who makes the hardware or the software used on a computer.

This book focuses on how to develop good software. A piece of software is a set of instructions for a computer to execute. Just as human beings understand different languages so does a computer. On the larger scale a computer understands “Low Level Language” and “High Level Language”, some books include “Intermediate level”.

A computer understanding of a “language” means that it can take instructions based on the “language” used. The “low level language” is the machine code (Assembly language” which is mostly machine-specific. The high level language unlike the low level is close to human language like English; Arabic or Chinese. The high level language is not strictly machine-specific. Some high level languages interact directly with the microprocessor so they are justifiably termed Intermediate level language as earlier mentioned. JAVA, COBOL FORTRAN and BASIC are some examples of the high level languages where C and C++ are examples of intermediate level. Assembly language is a low level language.

At one time or the other the reader must have seen a computer or at least a digital wrist watch. The activities of a computer are guided by programs. The aim of this text is to give the reader a wide practical scope in programming with QBasic. However, it should be noted that QBasic is not all in all. As the name tries to portray (BASIC: An acronym standing for Beginners All-purpose Symbolic Instruction Code). QBasic is the Microsoft Corporation Version of BASIC, where QBASIC stands for Quick BASIC.

This book is written for learners from any field (Science, Engineering, Accounting and so on). The projects and the working examples take a form that applies to these fields. The tips (Side Talk) are given based on the author's experience, over six years in teaching and programming BASIC.

§1.1 *STARTING UP*

A great step you have taken is "you're having this book". You can always start to learn computer from any point so far you have interest. Starting programming could be taken from any language depending on your level of understanding but a very good starting point is BASIC. After understanding a programming language it is very easy to learn other languages. Start it up now and there shall be no regrets.

SideTalk
You may skip to Chapter Two if you have been programming before.

§1.1.1 *What you can do with programming*

With programming you can achieve various mundane tasks but compulsory tasks with ease. In the banking industry, say, hundreds of thousands of customers have to be attended to all having different problems. With well-developed software, it would only take a couple of minutes to attend to a meaningful number of customers relative to the good old days before computers. Not only here we find programming applicable in Census, Biometrics, Statistical Analysis, Forecasting et cetera. Basically, everything done with computer is about programming.

§1.1.2 *Program Users and Program Developers*

These two categories of people are computer literates. The program users depend on the intrinsic expertise of the program developers. As a program developer you are indeed a program user for you cannot develop software without using it.

A program developer is otherwise known as Programmer. There are various categories of programmers; the high level language programmers write High level languages while Low level language programmers write low level language although a person can do the two. It is more “difficult” writing low level than high level.

Program developing in most cases involve a group of program users and at times many programmers handling different subroutines depending on the complexity of the software to be developed.

After religiously following the book you can be sure of joining the league of excellent programmers. You are being prepared for a sound ground in programming with the text.

§1.1.3 *Application Packages*

Program users depend on software called application packages. Application packages are computer programs carefully developed to suit a specific task. A good example of an application package is AutoCAD 2004^{TM}, although not developed with QBasic. You can develop application packages for yourself or an establishment using BASIC. I have done it before.

In the package form, the code is no longer ordinarily editable. The code would have been converted to executable forms (.exe) format. Trying to edit the .exe format will corrupt the file usually in ASCII format.

§1.1.4 *Should I Proceed?*

At this point you should know if you want to be a developer or a user. Programming is my hobby, it is fun to program. It may be challenging in the very beginning but as you move on and *practice* with a computer you will appreciate it better.

Remember, no knowledge is waste. It may seem not very relevant now but you never can tell. Do not think programming is about stories and lots of talks as you have been reading from the introduction, this is just to whet your appetite.

§1.1.5 *Versions of BASIC*

BASIC, Beginners All-purpose Symbolic Instruction Code has developed over the years. It has various versions but all the versions are closely related. Some differ by numbering while others by some syntax differences. This book is not going to deal with the differences or metamorphosis of BASIC. As it is worth mentioning for academic purposes, we have the following versions to mention some: GWBASIC, PC-BASIC, STRUCTURED BASIC and QBASIC.

§1.2 *QUIZ*

Where do you type the BASIC code you develop for computer to execute?

(Attempt: IDE: Integrated Development Environment)

§1.3 *PROJECT*

Install QBASIC on your system on the Directory C:\QB45\ or D:\QB45\ or F:\QB45\ or your most convenient drive (not on a floppy disk). Ensure the help files are also installed.

Run the file from the command prompt, say, C:\QB45\QB.exe ***ENTER*** study the default environment.

Create a folder named MISC on your hard disk at the locations C:\MISC\ (or the appropriately chosen drive above-this folder (Directory) shall be used in the book.

CHAPTER TWO

CHAPTER TWO

§2.1 DEVELOPING A PROGRAM

This chapter will take you through the basic of what it takes to develop a (BASIC) program. The steps are similar to what it takes in developing most other .programs.

§2.1.1 Algorithm

This is broken down and summarized steps to execute a plan. It “details” the “hows” of execution. A computer does not however understand Algorithm, but it is one of the first steps in developing a program that will not “grow old” before death. Some programmers have the habit of just starting the code without the preliminaries and find out later that the code is not doing what it is expected to do.

The algorithm below shows the steps to make “Eba”

i. Boil water
ii. Quickly add “gari”
iii. Stir to blend
iv. Ready for eating
v. Stop/Finish

Fig 2.1.1 Algorithm for “eba” making

The chronological order above is quite easy to follow but may be misleading. The algorithm does not specify the quantity of water, how to recognize the boiling water, amount of “gari” what to use to stir, what level of blend and the temperature to be considered as cool. The algorithm may be further broken down.

Depending on the style you wish to be adopting, your algorithm may be simple or complex. For academic purpose it should be easy to follow by others for you cannot tell where your algorithm may be put to special use.

§2.1.1 Flowchart

A flowchart is a schematic representation of algorithm or the program flow using some set of standard symbols. The various basic symbols used are shown in Fig 2.1.2a.

SideTalk
Computer does not understand flowchart. It is not a language!

SideTalk

A flowchart without direction is as useless as writing with black pen on a black book!

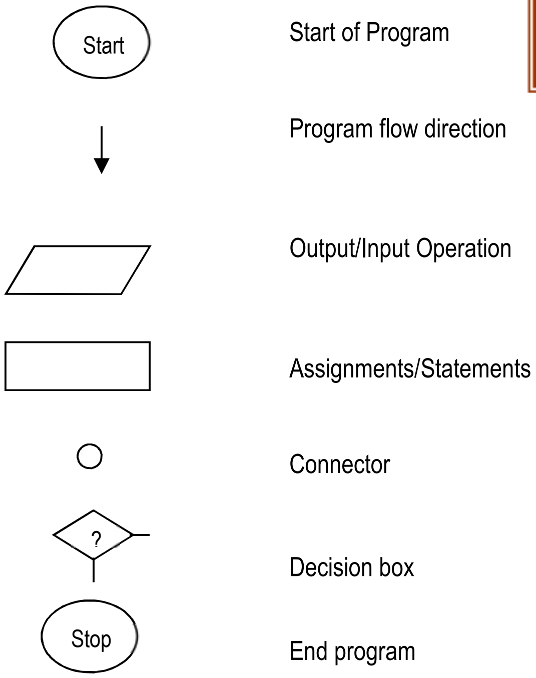


Fig 2.1.2a Basic Symbols used in Flowcharting

The flowchart of fig 2.1.2b shows a simple flowchart for implementing the algorithm of Fig 2.1.1

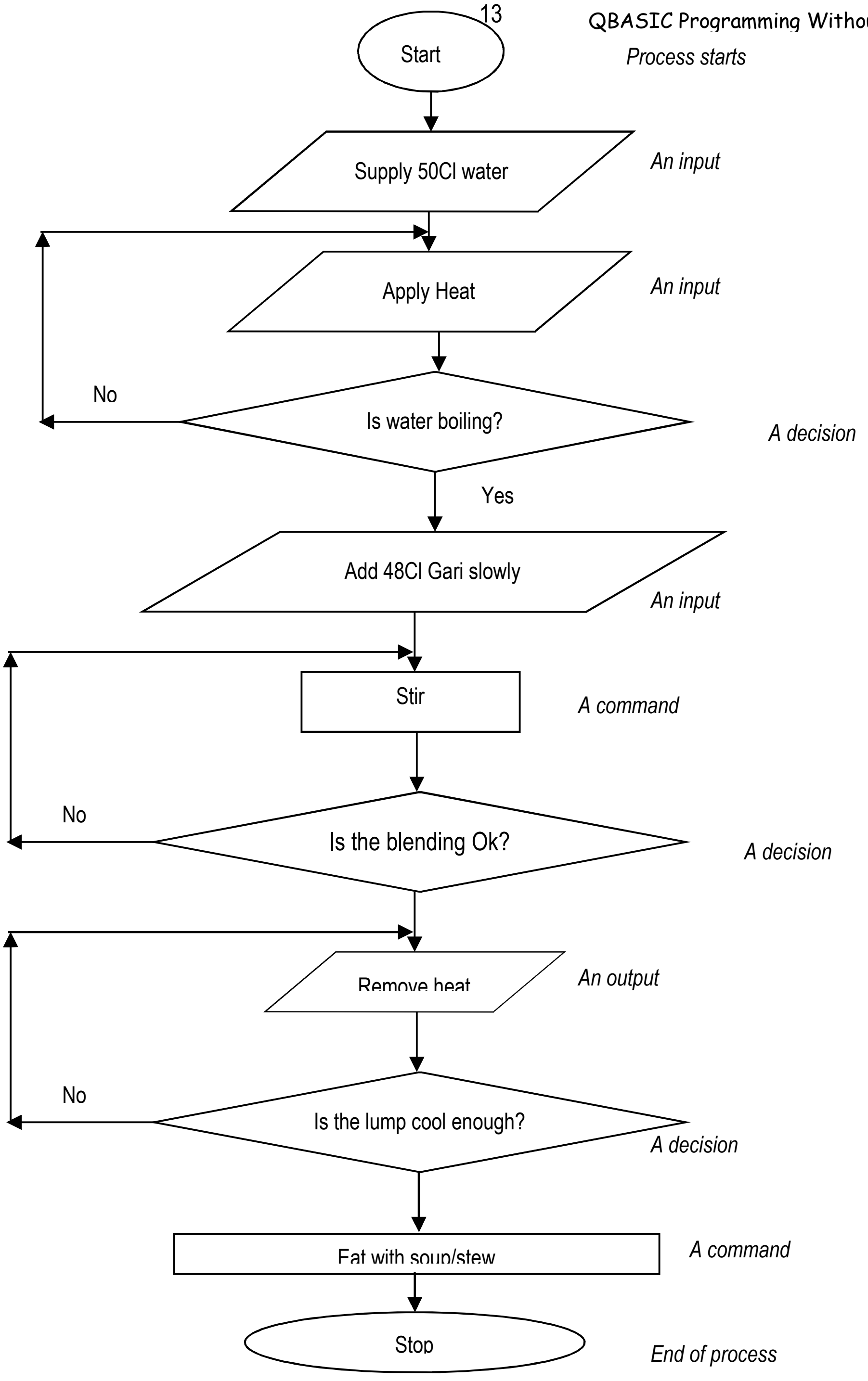


Fig 2.1.2 Flowchart for making 50Cl “eba”

§2.1.3 Pseudocode

A pseudocode is a tool used by programmers to represent a flowchart or it could be seen as an Algorithm made to look like real codes. It is human-language-like. If you understand English, you write your pseudocode with English Language. A pseudocode is generally easy to convert to any programming language. The listing of LST2.1.3 shows the pseudocode for the implementation of the "eba" making.

SideTalk
A computer does not understand pseudocode

LST 2.1.3 **Pseudocode for making "eba"**

```
STEP 0:     START THE PROCESS
STEP 1:     SUPPLY 50Cl water
STEP 2:  L1: INPUT HEAT
         L2: REPEAT L1 UNTIL WATER BOILS
STEP 3:     INPUT GARI TO THE BOILING WATER
STEP 4:  L3: STIR THE LUMP
         L4: IF THE LUMP IS BLENDED ENOUGH DO PROCEED TO L5 ELSE K3
STEP 5:  L5: PLACE LUMP CONTAINER IN COLD WATER
         L6: TAKE THE LUMP TEMPERATURE T
         L7: IF T<= SET TEMPERATURE GO TO L8 ELSE L6
         L8: LUMP IS READY
STEP 6:     STOP/FINISH
```

§2.1.4 Quadratic Equation Example

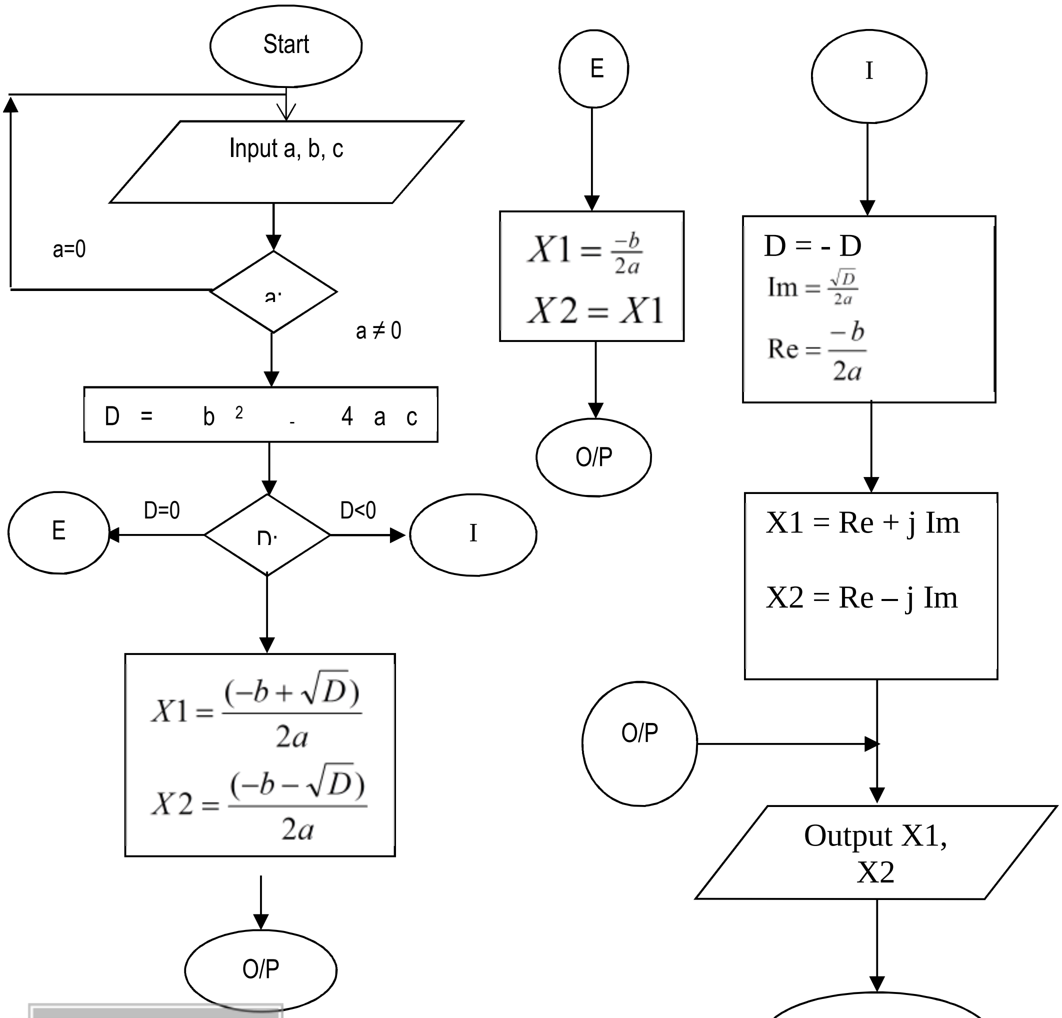
A quadratic equation is an equation of the form $\mathbf{aX^2 + bX + c = 0}$, where $a \neq 0$ and a, b, c are constants. A quadratic equation is to be solved for X. many application problems in Mathematics, Engineering, Economics lead to quadratic equation. Developing a program to implement quadratic equation is worthwhile.

Of the many methods of solving a quadratic equation, we choose the most viable to program. The quadratic formula method solves all forms of quadratic equation using the formula: $X = \dfrac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ The term $D = b^2 - 4ac$ is called the DISCRIMINANT. The discriminant determines the type of solution obtainable from an equation

I shall develop the Algorithm, flowchart and a pseudocode to implement quadratic equation solving.

```
1.      OBTAIN THE COEFFICIENT a, b, AND c
2.      COMPUTE DISCRIMINANT
3.      USE QUADRATIC FORMULA TO SOLVE FOR X
4.      OUTPUT RESULTS
5.      FINISH
```

Fig 2.1.4a Algorithm for Quadratic Equation



SideTalk
Follow the flow

Fig 2.1.4b Flowchart for Quadratic Equation

LST2.1.4 Pseudocode for Quadratic Equation

```
STEP 0: START
STEP 1:                {* ACQUISITION OF DATA*}
        L1:    INPUT a, b, c
        L2:    {*VALIDATION*}
               IF a=O DO REPEAT L1
STEP 2:                {*DISCRIMINANT*}
        L3:    D= $b^2$ -4*a*c
STEP 3:                {*EQUATION SOLUTION TYPES*}
        L4:    IF D=0 THEN GOTO L9                    {*EQUAL ROOTS*}
        L5:    IF D<0 THEN GOTO L11                   {*IMAGINARY
ROOTS*}
        L6:    IF D>0 THEN GOTO L7          {*REAL ROOTS*}
STEP 4:                {*SOLUTIONS*}
        $L7 : X1 = \frac{(-b+\sqrt{D})}{2a}$
        $L8 : X1 = \frac{(-b-\sqrt{D})}{2a}$

        L8.1:    GOTO L16                             {*OUTPUT*}
        L9: $X1 = \frac{-b}{2a}$
        L10:       X2=X1
        L10.1:    GOTO L17
         L11:       D=-D                    {*MAKE D POSITIVE*}
         $L12 : \text{Im} = \frac{\sqrt{D}}{2a}$
         $L13 : \text{Re} = \frac{-b}{2a}$
          L14: X1=Re + j Im
          L15: X1=Re - j Im
STEP 5:                     {*OUTPUT*}
          L16:        PRINT OUT X1,X2, EQUATION_TYPE
STEP 6:                     {* END PROCESS*}
          L17:        STOP
```

§2.1.5 BASIC CODE

This is the actual language this book is preparing you for. This chapter is not enough to give a full mastery of the code. A sample code is written in BASIC to perform simple addition is shown in the listing of LST2.1.5.

SideTalk

This book adopts the style using Bold and Uppercase for Keywords

A keyword is the "unit word" in the programming language and is not used as a variable!

```
10    REM Simple Addition Program
20      CLS
30    DIM X, Y, Z AS INTEGER
40    INPUT X, Y
50    LET Z = X + Y
60    PRINT " X + Y =" ; Z
70    END
```

§2.2 SUMMARY

Developing a good program involves using any or all of the tools vis-à-vis Algorithm, Flowchart and Pseudocode. Using any of these tools not only helps you to write good code but gives others reading your code better meaning. It is easier to understand your solution from these tools than going through your code!

§2.3 QUIZ

What is the difference between a connector and Start/Stop symbols of a flowchart?

(Connector is smaller)

§2.4 PROJECT

Develop a flowchart to attend and diagnose a patient for Malaria. The flowchart should prescribe drug, sleep or otherwise. Your flowchart should be standard.

CHAPTER THREE

CHAPTER THREE

§3.1 VARIABLES AND ASSIGNMENTS

In analytical sciences and other fields variables are used to define real terms, for example **I**=**PRT**/1ØØ used in simple interest computation has the variables **P**, **R**, **T** and **I**. in biological science, the rate of growth of bacteria follows the equation $\mathbf{N=N_0e^{\lambda t}}$ where **N**, $\mathbf{N_0}$, **λ** and **t** are variables. In fluid mechanics, the Laplacian equation has variables **x**, **y**, **z** and **t** as in the steady flow equation $\frac{\partial^2 x}{\partial t^2}+\frac{\partial^2 y}{\partial t^2}+\frac{\partial^2 z}{\partial t^2}=0$

Though variables appear in different fields the value they can "hold" differ. For example **I** in the first example is in %, **t** in the second example is a *time* and **x** in the third is in *metres*. In computer programming it is necessary that a *code* defines the nature of data a variable can hold. By default, QBASIC allows a variable to be used without prior declaration in *non-string* usage.

To the QBASIC programmer, variables can take any of the form *Integer, Long, Single, Double or String.* Deciding what type a variable should take depends on experience. The table below gives a guide on the data range for the variables.

TABLE 3.1 VARIABLES

VARIABLES	IDENTIFIER	VALUE RANGE
INTEGER	%	-32768 TO +32767 (16 bit Signed Integer)
LONG	&	-2147483648 TO +2147483647 (32 bit Signed Integer)
SINGLE	!	7 digits accuracy- Single precision floating point - ++
DOUBLE	#	16 digits accuracy – Double precision floating point - ++
STRING	$	Alphanumeric – Theoretically 2 billion characters. If followed by *n where n is an integer, it is a fixed length byte - see TYPEs

++(Limited accuracy at *extremes* for values are approximations)

Floating point variables are the exponential valued terms. $3X1Ø^{15}$ is a floating point value. The value is the same as 3e15 or 3E15

§3.1.2 DECLARATION OF VARIABLES

As a good programming style, good enough to develop on other programming languages, it is wish enough to make declaration of variable names as a type. In Africa, people are named by events, wishes or otherwise. Some programmers use this method when naming variables. The keywords **DIM**, **REDIM**, or **DIM SHARED** are used. The abridged program listing LST3.1.2 shows declaration of variables.

LST3.1.2 Declaration of Variables [Abridged]

```
.
.
5Ø DIM X AS INTEGER
6Ø DIM Age AS STRING
7Ø REDIM MatrixA(5Ø,5Ø)
8Ø DIM Ratio AS SINGLE
9Ø DIM SHARED myUsableMat (15, 15) AS STRING
.
.
.
```

§3.1.3 Assignment in BASIC

To make assignment in programming is to assign value to variable. The BASIC assignment is similar to the equality of a variable to a value or expression in Mathematics. Unlike in Mathematics, the expression or the value cannot be on any side of choice but on the right.

It is easy to point out that X=X+5 does not make any sense in Mathematics as it means X – X = 5 or Ø=5! In BASIC programming X=X+5 means assign to a variable X the last value of X plus 5. If in the lines of Code preceding X=X+5, the

SideTalk

It is a preferred style to name variable like MatA(); Tel; DOB; NameCandidate rather than using, A; T; D; N. Note carefully that space is not allowed within a variable and keywords are not allowed to be used as variables.

SideTalk 2

QBASIC allows either of the following forms:
1Ø **DIM** X **AS INTEGER** ' *or **DIM** X%*
2Ø **DIM** Age **AS STRING** ' *or **DIM** Age$*

SideTalk

By default a non-string variable has a value Ø while a string variable has empty value

SideTalk 2

X=2+5Y
2 + 5y=X are good expressions/equation in Mathematics. The first is permitted in QBASIC but not the second

value of X was, say, X=1Ø, the assignment statement X=X+5 means X=15. If the same line is executed again, the value of X becomes 2Ø.

The abridge program listing of LST3.1.3 shows simple assignment and output on the right.

LST3.1.3

```
.
.
.                                     OUTPUT
 5Ø LET X=6
 6Ø LET X=X+1Ø
 7Ø PRINT X                              6
 8Ø PRINT X+1Ø                          16
 9Ø LET X=X+1Ø
1ØØ PRINT X                             26
.
.
```

The statement in line 8Ø will not increase X by 1Ø but only prints out the value of X+1Ø

§3.1.4 **Operation on variables**

There are basic operations possible on variables as it is obtained in Mathematics. The basic operators used in BASIC are:

BASIC OPERATORS

*	For multiplication
+	For Addition/Increment/String concatenation
-	For subtraction/decrement
/	For division
\	For integer division

SideTalk

The misuse of / and \ will not flag error but will cause a semantic error. The code 1Ø Y=5/2

2Ø X=5\2 will give different values 2.5 and 2 respectively.

§3.1.4.1 **String Manipulation**

Strings are alphanumeric. This implies that they can contain special characters including the special ASCII characters. Variables for values like telephone numbers, social security number (as used in the US), names of people or Objects and so forth are to use string variables. The code listing of LST3.1.4a shows a single handling of string variables.

LST3.1.4a Sample String handling

```
 1Ø  DIM Age AS STRING
 2Ø  DIM  Sex   AS STRING
 3Ø  DIM Names, Tel AS STRING
 4Ø  CLS
 5Ø  PRINT "You are welcome", " Respond to the following :"
 6Ø  PRINT: PRINT
 7Ø  INPUT "Type your name:"; Names
 8Ø  INPUT "Type your sex:"; Sex
 9Ø  INPUT "Type your age:"; Age
1ØØINPUT "Type your Telephone Number:" Tel
11Ø CLS
12Ø INPUT "PRESS ENTER", REP$
13Ø PRINT: PRINT: PRINT
14Ø PRINT "THE ID JUST ENTERED >:"
15Ø PRINT Names; Sex; Age; Tel
16Ø END
```

SideTalk

Repeating the listing LST3.1.4 without the lines 1Ø, 2Ø an d 3Ø will cause a runtime error for lines 7Ø, 8Ø an d9Ø as the variable *Names, Sex* and *Age* will be assumed to be *non-string*. Adding the $ sign at the end of the variables can prevent the error as in line 12Ø.

Strings are extensively used in programming, it therefore require the knowledge of how to be manipulated. QBASIC has some built-in functions to manipulate strings.

I will maintain that practicing on a computer is the best way to put all these theory into practice. The example following gives a good understanding of string manipulation.

LST3.1.4b String Manipulation

```
52Ø A$= “What is the length of this word ?”
53Ø Lt= LEN(A$)
54Ø PRINT “The length is =”Lt
55Ø B$= “QBASIC Programming…”
56Ø C$ =UCASE$(B$)
57Ø PRINT C$
58Ø D$=LEFT$(B$, 6)
59Ø E$=RIGHT$(A$,6)
6ØØ F$=MID$(C$,8,7)
61Ø G$= MID$(C$,1,6)
62Ø H$=LCASE$(G$)
63Ø PRINT “The manipulated strings output >”
64Ø PRINT C$
65Ø PRINT D$
66Ø PRINT E$
67Ø PRINT F$: PRINT G$: PRINT H$
.
.
.
```

SideTalk

The output of the functions in listing LST3.1.4b should be carefully studied. The output screen below shows the result of the abridged code:
Hint: spaces are included in the counting of strings for they are also strings. (See 3.1.4.1)

```
OUTPUT SCREEN

The length is 33
QBASIC PROGRAMMING…
The manipulated strings output >
QBASIC PROGRAMMING…            C$
QBASIC                         D$
word?                          E$
PROGRAM                        F$
QBASIC                         G$
qbasic                         H$



Press any key to continue
```

LEN(“Adeniyi”)	7
UCASE$(“Adeniyi”)	ADENIYI
LCASE$(“Adeniyi”)	adeniyi
RIGHT$(“Adeniyi”, 3)	iyi
LEFT$(“Adeniyi”,3)	Ade
MID$(“Adeniyi”,2,5)	deniy

Use the syntax format below for the string functions:

LEN (*string or String Variable*)	Read Only
UCASE$(*string or String Variable*)	Converts to upper case
LCASE$(*string or String Variable*)	Converts to lower case
RIGHT$(*string, length of string from right)*	Read only
LEFT$(*string, length of the string from left)*	Read only
MID$(*string, start position, length of variable)*	Read only

§3.1.4.2 Concatenation of strings

This is the "merging" of two or more strings parts to form a single string with the use of the "+" operator. Strings can be concatenated after manipulation or concatenated and manipulated. For example we may want the first letter of a name to be uppercase (capital letter) and the rest be in lower case. The listing of LST3.1.4c illustrates simple concatenation of strings.

SideTalk
The only operator that works in string manipulation, of the operators described above, is the concatenation operator "+"

```
1ØØØ Part1$= "University"
1Ø1Ø Part2$ = "of"
1Ø2Ø Part3$ = "Ilorin"
1Ø3Ø Part$ =Part1$ + Part2$ + Part3$
1Ø4Ø PRINT Part$, UCASE$(Part$)
1Ø5Ø Fullname$ = "ADENIYI"
1Ø6Ø FirstLetter$ =LEFT$(Fullname$, 1)
1Ø7Ø Remain$ = MID$(Fullname$, 1, 6)
1Ø8Ø Formatted$ = UCASE$(Firstletter$) + LCASE$(Remain$)
1Ø9Ø PRINT Formatted$, Fullname$
```

```
               OUTPUT SCREEN
University of Ilorin
UNIVERSITY OF ILORIN
Adeniyi   ADENIYI

Press any key to continue
```

§3.2 OUTPUT STATEMENT

The method used to get out the result of calculations or strings and other variables either to the monitor or hard copies on paper or to memory devices are by output statements. Although you have coming across some output statements, this section is dedicated to output statements.

§3.2.1 Print and Print Using

PRINT is a keyword in QBASIC for outputting to the monitor (Screen). It is very easy to use but may require some formatting for a "beautiful" output. **PRINT** 2 will print 2 to the screen, the same is to **PRINT** "2" but the 2 here is a string. This will make more sense by comparing the output of **PRINT** "2*2" and **PRINT** 2*2. While the output of the former is 2*2 the latter gives 4.

SideTalk
To print a string, you put quotation marks but a string variable does not need a quote e.g "A $" is not it.

Printing with gaps is possible by use of comma and semi-colon. The former gives a wider gap than the latter. The codes **PRINT** 1,2,3,4 and **PRINT** 1;2;3,4 look similar but the outputs are respectively:

1   2   3   4 and 1 2 3   4 (notice the *printing zones*)

PRINT USING is also a keyword in QBASIC like **PRINT** but has special formatting characteristic which will require special skill to do with **PRINT**. The syntax is **PRINT USING** *"format string", [variables]*. The listing of LST3.1.2 gives a simple use of **PRINT** & **PRINT USING**.

LST3.2.1 Print and Print Using

```
1Ø CLS
2Ø INPUT "Enter the number of males";Nm
3Ø INPUT "Enter the number of females";Nfm
4Ø PRINT USING "The class has ###.# % of males"'1ØØ*Nm/(Nm+Nmf)
5Ø  PRINT "The percent of females =";1ØØ*Nmf/(Nm+Nmf);"%"
```

You can get output like:

```
The class has 23.75 of males
The percent of females =76.33333333333333%
```

Observe the formatting of the first one.

You should try run the following code:

```
10 CLS
20 DIM FirstName$, LastName$
30 FirstName$ = "Akinwale"
40 LastName$ = "Ajasin"
50 PRINT USING "!"; FirstName$; LastName$
60 'Line 50 prints the 1st letters of the two string expressions i.e. AA
70 PRINT USING "\        \"; FirstName$; LastName$
75 'Eight spaces between backslashes,
80 'prints Ten characters from FirstName$ i.e. Akinwale
90 PRINT USING "\   \"; FirstName$; LastName$; "!!"
95 'Three spaces, prints Akinwale and a blank. Note the overlap
110 PRINT USING "! "; FirstName$; 'First character from FirstName$ and
120 PRINT USING "&"; LastName$    'all of LastName$ on one line i.e. A Ajasin
130                               ' Note there is a space after !
140 END
```

§3.2.2 Printing to *file*/LPRINT

As your programming sills increase, you may need to handle files or "big" data as well as printing out results on paper from line printers. QBASIC is equipped with this facility.

SideTalk

The # sign is used to represent figures (numeric). If your program is that of giving values and the unit it is good to use **PRINT USING**. For temperature program we can employ the style below:

PRINT USING "Temperature=###^{Ø}F"; Temp

For Pressure:

PRINT USING "Pressure=###e##N/m^{2}"; Pres

For Law or Constitution quoting:

PRINT USING "Section ##, Sub-Section ## "; St, Sbst

Printing to file requires that you open file for the type of access you require first then use the **PRINT** *#fileNum[,…]* then **CLOSE** the **OPEN** file channel. The listings of LST3.2.2 show a simple use of printing to file. See Chapter Seven for more files.

LST3.2.2a Printing to file

```
1Ø REM Open the file for output mode for result only

2Ø OPEN “C:\MISC\test.dat” FOR OUTPUT AS #1

3Ø PRINT “The other Prints are output to file not to the screen as this.”

4Ø PRINT #1,“This is the beginning of the file”

5Ø PRINT #1“A new line on the File”

6Ø PRINT #1, 5ØØ-2ØØ*1Ø

 7Ø CLOSE #1

8Ø…..
```

SideTalk

To view the output of the Simple code listing LST3.2.2.a, you can use the explorer, notepad or follow the following steps: File | Dos Shell > On the DOS prompt enter **Cd\,** enter **Cd Misc,** next enter **Edit Test.dat** to view the result. Use File | Exit to close the window then enter **Exit** on the Dos prompt to return to QBASIC environment or as appropriate.

LPRINT: You can send output to paper using the **LPRINT** keyword. The output will be similar to what you see on your screen using the **PRINT** keyword. To run the code with **LPRINT** statement, you must ensure you have a line printer and paper. It should be noted that paper margin etc can be set from QBASIC. To see more on this press **F1** on the typed word **LPRINT.**

Run the code in LST3.2.2b if you have a printer connected and powered. It will run perhaps only with a line printer.

LST3.2.2b Using **LPRINT**

```
1Ø LPRINT "This is from QBASIC"
2Ø LPRINT "_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
3Ø LPRINT: LPRINT: LPRINT
4Ø  A$=" I am learning fast"
5Ø   Money =4ØØ
6Ø  LPRINT USING "Please pay =N= ###.## K because", Money;
7Ø  LPRINT A$
8Ø  END
```

SideTalk

REM is a keyword meaning Remark. QBASIC ignores everything following **REM** on the same line

§3.2.3 WRITE

WRITE # and **PRINT** # are similarly used but the output may vary in some cases. **WRITE** # does not put superfluous spaces as does **PRINT** #. It is better to use **PRINT** # with **LINE INPUT** # and **WRITE** # with **INPUT** #. Run the code listing LST3.2.3a and LST3.2.3.b. Open the two output files and make comparison as regards the output formats and the memory sizes.

LST3.2.3a WRITE #

```
1Ø CLS
2Ø REM A program written for comparison sake
3Ø REM
4Ø OPEN “C:\MISC\DUMMY1.TXT” FOR OUTPUT AS #1
5Ø WRITE #1, “This is the WRITE program Output”
6Ø WRITE #1, 1, 2, 3,4,5,6
7Ø WRITE #1, “A”, “B”, “C”, “D”
8Ø DIM Counter%
9Ø FOR Counter%1 =1 TO 5Ø: REM A looped Printing
1ØØ WRITE #1, Counter%;
11Ø NEXT
12Ø PRINT “Done”
13Ø CLOSE 1: REM Close the opened file
14Ø END
```

LST3.2.3b PRINT #

```
1Ø CLS
2Ø REM A program written for comparison sake
3Ø REM
4Ø OPEN “C:\MISC\DUMMY2.TXT” FOR OUTPUT AS #1
5Ø PRINT #1, “This is the PRINT # program Output”
6Ø PRINT #1, 1, 2, 3, 4, 5, 6
7Ø PRINT #1, “A”, “B”, “C”, “D”
8Ø DIM Counter%
9Ø FOR Counter%1 =1 TO 5Ø: REM A looped Printing
1ØØ PRINT #1, Counter%;
11Ø NEXT
12Ø PRINT “Done”
13Ø CLOSE 1: REM Close the opened file i.e. Dummy2.txt
14Ø END
```

§3.3 INPUT STATEMENT

There are various sources of input to a computer system. Input can be from keyboard, file, mouse, ports, joystick and so forth of concern here are the keyboards and files.

§3.3.1 Input / Input

INPUT: Used to get data from the keyboard. The data is accepted after pressing the return (Enter) key. The **INPUT$(***val***),** where *val* is a number specifying the number of keys pressed, does not require the return key to accept value. Another family of the input is the **INKEY$**. It is any key *pressed* or *not pressed***.** If a key is not pressed the value of **INKEY$** is “”.

INPUT#: This is used to “read” data from an input file or a random access file. Before **INPUT#** could be used, the data format must be known to avoid error, such trying to read a string like integer. The program listing of LST3.3.1 shows a simple use of the **INPUT** statements.

LST3.3.1a Using Input

```
1Ø CLS
2Ø INPUT X
3Ø INPUT “Enter the value of Y”; Y
4Ø INPUT A, B, C
5Ø LET D = X + A + B + C + Y
6Ø PRINT “”The sum of all values entered =”; D
8Ø END
```

The response of line 4Ø is not as friendly as line 3Ø. Line 4Ø gives a question mark but line 3Ø requests by printing the quoted string “Enter the Value of Y”. The values to be input are specified after the quotation. The punctuation after the quotation determines whether there will be a question mark or not. The statements:

INPUT “Do you like Rice”; Ans$** and **INPUT “Do you like Rice?”, Ans$ will be the same. While the first puts a question mark as result of the semicolon, the second does not include a question mark because of the comma but the question mark shown is the question mark in the quotation. Use comma or semi-colon depending on if you are asking question or requesting.

The listing of 3.3.1b shows a simple program using INPUT$ and INKEY$ functions. You will come across some codes which you will soon get to learn about.

LST3.3.1b Using INKEY$ and Input$

```
1Ø CLS
2Ø PRINT "WELCOME TO AN INTERACTIVE";
3Ø PRINT "SESSION"
4Ø PRINT: PRINT
5Ø PRINT "Press C to Continue"
7Ø DIM Reply AS STRING
8Ø Reply=INPUT$(1)
9Ø   IF Reply= "X" THEN 14Ø
1ØØ IF Reply= "C" THEN
11Ø PRINT "Press Any Key to Continue"
12Ø DO : LOOP UNTIL INKEY$ <>""
13Ø END IF
14Ø CLS: PRINT "Done"
15Ø END
```

SideTalk

Putting a Semicolon to the end of a PRINT Statement will cause the next Print Statement to Print immediately following it rather than go to the next line. See lines 20 and 30

A colon implies a new line see line 120. It is useful when you mistakenly skipped the line and you don't want to erase. It should be noted that a colon after a REM Statement will not be EXECUTED and that the first command is executed first.

Of the INPUT# family is the LINE INPUT#, it is used to “read” from a Sequential file (see Chapter Seven). It takes the whole content of a line in the file. The program listing of LST3.3.1b creates a file and uses both INPUT # and LINE INPUT # to get records from the Created file.

LST3.3.1c Using INPUT # and LINE INPUT#

```
1Ø CLS
2Ø DIM ClientID, NumCalls AS INTEGER
3Ø DIM Tel$, NameC$: DIM m%, I%
4Ø OPEN “C:\MISC\LINP.txt” FOR OUTPUT AS #1
5Ø REM Line 4Ø Opens a file to be written to
6Ø ‘This is same as REM i.e the Apostrophe
7Ø CLS
8Ø PRINT “A SIMPLE TELEPHONE BOOK”
9Ø INPUT Enter the number of Clients”; Num%
1ØØ FOR I% =1 TO Num%
11Ø   INPUT “Client Name”, NameC$
12Ø   INPUT “Client Telephone Number”; Tel$
13Ø   INPUT “Number of Calls made”; NumCalls
14Ø      ClientID=1ØØØ= I%
15Ø PRINT #1, ClientID; NameC$; Tel$; NumCalss
16Ø PRINT
17Ø NEXT
18Ø CLOSE 1: DIM Rec
19Ø CLS: OPEN C:\MISC\LINP.txt” FOR INPUT AS #2
```

```
2ØØ PRINT "THE Telephone Book Content"
21Ø PRINT "ID", "NAME", "TEL-NUM", "CALLS": PRINT
22Ø FOR I%=1 TO Num%
23Ø   LINE INPUT #1, Rec$
24Ø PRINT Rec$
25Ø NEXT
26Ø REM The code of 22Ø-25Ø is similar
27Ø  ' to the listing below:
28Ø CLOSE 2
29Ø OPEN "C:\MISC\LINP.txt" FOR INPUT AS # 1
3ØØ REM Note that once you CLOSE the file you can
31Ø 'use the same number notice that lines 29Ø and 4Ø are the
315 ' same
32Ø PRINT
33Ø FOR I%=1 TO Num%
34Ø    INPUT #1, ClientID, NameC$, Tel$, NumCalls
35Ø PRINT ClientID, NameC$, Tel$, NumCalls
36Ø NEXT
37Ø REM Line 34Ø requires that you know the format of the file
34Ø CLOSE 1
39Ø PRINT "Done"
4ØØ END
```

§3.3.2 Read and Data

If have you have small size data for analysis or manipulation, instead of always supplying the data at run-time (every time you run the program) you can use READ and DATA pair. It is to be noted that

READ and DATA are complimentary; you cannot *read* without *data* otherwise you encounter a runtime error.

The format by which you wish to read your data is the way you store it. The program listings of LST3.3.2a give a simple use of the "tool".

LST3.3.2a Using Read and Data I

```
1Ø CLS
2Ø DATA 5, 1Ø, 12, 14
3Ø DATA 6, 7, 9, Ø
4Ø READ A, B, C, D
5Ø PRINT A, B, C, D
6Ø READ A, B, C, D
7Ø PRINT A, B, C, D
8Ø END
```

SideTalk

You place DATA anywhere in the Program (Read will search for its colleague) except in Loops or subroutines depending on the neatness you desire for your code.

This program prints out the whole Data. Note that line 4Ø and 6Ø are the same but they do different things. The first READ reads four data in the DATA statement. It is not compulsory that you spilt the data as done in 2Ø and 3Ø. The entire data can be on a single line or split into as many data lines as desired. They may not necessarily be following i.e. one of the data line may be placed in line 75 while the other in 2Ø. QBASIC does see the data as data not as separate data.

LST3.3.2b Using Read & Data II

```
1Ø CLS
2Ø DATA “Nigeria”, “Ghana”, “Uganda”, “Niger”
3Ø DATA “Cameroon””, “S/Africa”, “Congo”
4Ø DATA 196Ø, 1956, 1955, 196Ø, 1963, 199Ø, 2ØØ1
5Ø DATA “COUNTRIES”
6Ø DIM I%: DIM Country$: DIM Yr
7Ø FOR I%=1 TO 7
8Ø   READ County$
9Ø PRINT UCASE$(Country$), “Number =”,I%
1ØØ NEXT
11Ø FOR I%=1 TO 7
12Ø READ Yr
13Ø PRINT USING “Country Number # was visited in ####”, I%, Yr
14Ø NEXT
15Ø END
```

If the data were stored as shown below, it would have been easier to manipulate. Care should however be taken so as not to read a string data with an integer variable. The program listing of LST3.3.2c is a better modification of LST3.3.2b. the positioning of the data is not the case but the format used.

LST3.3.2c Using Read & Data III

```
1Ø CLS
2Ø PRINT “Countries Visited and Year of Visit”
3Ø DIM I%, Country$, Yr%
4Ø FOR I%=1 TO 7
5Ø   READ Country$, Yr%
6Ø PRINT County$; “VISITED:”, Yr%
7Ø NEXT
8Ø DATA “NIGERIA”, 195Ø, “GHANA”, 1952
9Ø DATA “CAMEROON”, 1953
1ØØ DATA “TOGO”, 196Ø, “NIGERIA”, 197Ø
12Ø DATA “USA”, 2ØØ4, “CANADA”, 2ØØ4
13Ø END
```

Data can be in excess of the read encountered but the number of READ encountered (or Read Request) should not be more than available Data. An *error flag* is displayed “OUT OF DATA” if such is allowed to happen.

Running the code below will flag such error:

```
5Ø DATA 4, 8, 13, 42
6Ø FOR I%=1 TO 6
7Ø   READ Num%
8Ø PRINT Num%
9Ø NEXT
```

The error is flagged when I% =5 (5th time of entering the loop). In some situations, the same set of data may be required elsewhere after been read using READ statement, there is a keyword used to refresh DATA reading “Pointer”. RESTORE is a keyword that is used to *start* reading DATA from the first data statement. The listings of LST3.3.2d show the use of RESTORE statement.

LST3.3.2d Using RESTORE

```
1Ø CLS
2Ø DATA Ø.5, Ø.3, Ø.9
3Ø DIM A
4Ø DIM B, C, D AS SINGLE
5Ø READ A, B, C
6Ø D= 4*A – B + C
7Ø PRINT “First Time D=”; D
8Ø RESTORE
9Ø REM Line 8Ø Starts another Read from Data value Ø.5
1ØØ READ D, B, C
12Ø A=D^2+B^2+C^2
13Ø A=4*A
14Ø PRINT “Second Time A= “; A
15Ø REM Removing line 8Ø causes OUT OF DATA
16Ø REM Error flag.
17Ø END
```

§3.4 SUMMARY

Memory is allocated for the type of variable declared. This chapter discussed the following variable types INTEGER, SINGLE, DOUBLE and STRING with emphasis on QBASIC String Manipulation Functions and Concatenation of strings to suit a desired purpose. Assignment in mathematics is not as in QBASIC.

Computer takes input from *keyboard*, *files* and *other sources*. Data is taken in using the keywords INPUT, LINE INPUT and READ, others are INPUT$ and the INKEY$. Output can be on the monitor, hard paper or memory devices using PRINT, PRINT # and WRITE#.

§3.5 QUIZ

What is wrong with the following listings?

```
1Ø my Value=1Ø
2Ø Print = 6Ø
3Ø PRINT #1, "Out of Data"
4Ø DIM A$ AS STRING
5Ø A$="QUIZ"
6Ø A$=A$* "3RD QUIZ"
7Ø DATA=3, 5, 6, 7
8Ø READ A, B, C, D
9Ø END
```

Attempt:

1Ø Space not allowed,
use: myValue

2Ø Keyword should not be used as a variable

3Ø File should not be opened before bringing to file

4Ø DIM A$ OR DIM AS STRING

5Ø Correct

6Ø * is not allowed to manipulate Strings but +

7Ø = not required

8Ø Correct

9Ø Correct

-----* ***Program has no meaning***

§3.6 PROJECT

Develop a program starting as described in Chapter 2, to manage the records of a small shop. The program should be able to print available stock or record new stock and prices. You have to take into account, accounting practice, though the simple case: i.e. How to treat a cash book etc.

Bibliography

- Microsoft QuickBASIC online help
- Gary Cornell (1997) Visual Basic 5 from the Ground Up, McGraw – Hill, Berkeley, California

CHAPTER FOUR

CHAPTER FOUR

§4.1 CONTROLLING THE FLOW

This is the heartbeat of programming. A good programmer must be keen to controlling the flow of the program. The way instructions go/flow in a code determines the output of a program, therefore, call to mind the acronym *Garbage-In-Garbage-Out-GIGO*. This chapter shall discuss the basic ways to control the actions of the computer using QBASIC.

Chapter two discussed the skeletal control of program using flowcharts. It is aimed in this chapter to learn converting 'those flows' to software (QBASIC Codes).

§4.1.1 LOOPING

Looping in this context means doing a particular operation repeatedly for a 'period' controlled by a set condition such as number of passes through the loop or otherwise as is discussed below. QBASIC has some looping 'blocks' for achieving looping:

FOR-[EXIT FOR]-NEXT	loop
DO-[EXIT DO]-LOOP	loop
LOOP –UNTIL	loop
WHILE-WEND	loop

Depending on the situation and/or your 'love' for the type they can be variously manipulated.

§4.1.1.1 FOR-NEXT loop

This is the simplest and 'mostly used' looping block. The syntax is:

FOR *Counter* = *InitialValue* to *Upperlimit* [STEP *Increment*]

Line(s) of code(s)

[EXIT FOR]

NEXT [*counter*]

Expressions in the square bracket may be left out depending on the situation.

If counting is desired in the steps not 1 then the *increment* step is to be included. It should be noted that the *increment* can be negative and decimals. The listing LST4.1.1a shows a simple use of FOR-NEXT loops.

LST4.1.1a Using FOR-NEXT

```
1Ø CLS
2Ø DIM i%, j
3Ø FOR i%=Ø TO 1Ø
4Ø      PRINT i%
5Ø NEXT i%
6Ø PRINT
7Ø FOR i%=1ØØ TO 25 STEP -5
8Ø      PRINT i%
9Ø NEXT
1ØØ FOR j=Ø.2ØØ TO Ø.2Ø6 STEP Ø.ØØ1
11Ø   PRINT j
12Ø NEXT
13Ø END
```

NESTED FOR:

There can be two or more FOR-NEXT statements within one FOR-NEXT. However, the Nesting of the FORs should start from the 'innermost' loop. The structure is as shown in Fig 4.1.1

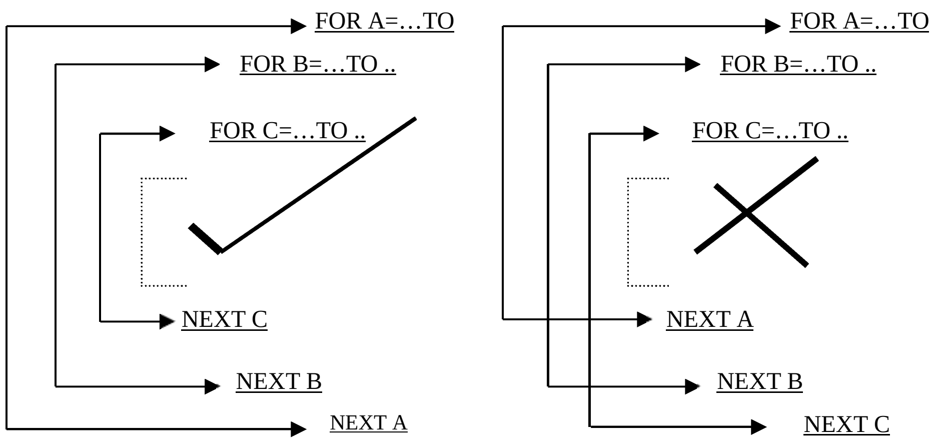


Fig 4.1.1 FOR-NEXT structure (Nested FOR)

In some situations a looping may be required to be stopped to continue with the program, the optional EXIT FOR statement comes in after checking a condition as the listing of LST4.1.1b shows.

```
LST4.1.1b Using EXIT FOR
1Ø CLS
2Ø DIM I%, J%
3Ø FOR I%=5Ø TO Ø STEP -1Ø
4Ø      FOR J%=1 TO 7
5Ø        IF 2*J%-I% <1Ø THEN EXIT FOR
6Ø      NEXT J%
7Ø          IF I%\1Ø =4 THEN EXIT FOR
8Ø NEXT I%
9Ø END
```

The listing LST4.1.1b does not have any output but it only aims at showing a sample placement of the EXIT FOR. The program will stop when I%=4Ø by the **integer division** of line 7Ø. The EXIT FOR of line 5Ø is for the J% loop while that of line 7Ø is for loop I%. it should be noted that the EXIT of a For-Loop is inside the loop.

Care should be taken when using ‘Counters’ in a loop. To understand how non-integer counters may be unpredictable, you may run the lines of code below and take note of the increment towards the end part of the output. This point should be carefully taken to avoid some bugs in your programs. It is most advisable not use integers.

LST4.1.1c Non-Integer counters

```
1Ø CLS
2Ø DIM foo AS SINGLE
3Ø FOR foo=Ø TO Ø.1 STEP Ø.ØØØØØ1
4Ø      PRINT USING “#.#########”;foo
5Ø NEXT: END
```

§ 4.1.1.2 DO-LOOP

Another looping block is the Do-Loop. The Syntax is:

DO [*Condition*]

[*Statement(s)*]

[EXIT DO]

LOOP

The optional condition is a logical comparison or a values magnitude check. If the condition is omitted and there is no condition to run the EXIT DO or go out of the Loop e.g. with GOTO *lineOutOfTheLoop*, the loop is an endless loop (The program will never stop! - If the Program is not yet compiled to an executable file Ctrl + Break is used to abruptly terminate the Program).

Note that the condition is firstly checked before control ever goes into the loop unlike the DO-LOOP-UNTIL block (see next section). The program listing of LST4.1.1d shows a simple use of the Do-Loop.

LST4.1.1d Using the Do-Loop

```
1Ø CLS
2Ø DIM I% : I%=Ø
3Ø DO WHILE I%<=1Ø
4Ø    I%=I% + 1
5Ø    PRINT  I%
6Ø LOOP
```

Compare the output of LST4.1.1d with changing the Line 3Ø to 3Ø DO WHILE I %< 1Ø

Care should be taken in positioning increments in Do-Loops. The output of the code will change if the lines 4Ø and 5Ø are interchanged.

As done in FOR-NEXT loops, Do-Loops can be nested too. This is described in LST4.1.1e

```
LST4.1.1e Nested Do-Loop
1Ø CLS
2Ø REM Nested Do-Loop
3Ø DIM i%, j%, Quit$
4Ø REM Initialising the Variables
5Ø LET i%=Ø
6Ø LET j% =Ø : LET Quit$ = "F"
7Ø DO WHILE (i% < 2Ø) AND NOT (Quit$= "T")
8Ø      DO WHILE j% < 4Ø
9Ø        PRINT USING "i=## → j=##"; i%,j%
1ØØ               j%=j%+2
12Ø     LOOP
13Ø   PRINT "QUIT T/F?"
14Ø   Quit$=INPUT$(1): Quit$=UCASE$(Quit$)
15Ø    i%=i%+5
16Ø LOOP
17Ø END
```

SideTalk
The LET keyword is optional

Try to limit the number of nesting not to run out of stack memory!

This arbitrary code listing would do similarly if line 7Ø is changed to:

7Ø DO WHILE i%<2Ø AND Quit$= "F", but if the user types a letter other than "F" the program, will stop unlike the original code. The optional [EXIT DO] can be used to abruptly 'move' out of a Do-Loop by setting a condition within the loop such as

IF Day$= "Sun" THEN EXIT DO

§4.1.1.3 LOOP UNTIL

The other form of the Do-Loop where the condition is checked after execution of the loop is as in the Syntax below:

```
DO
  Statement(s)
[EXIT DO]
LOOP UNTIL (condition(s))
```

The program listing of LST4.1.1f shows a simple use of the LOOP-UNTIL form of DO.

LST4.1.1f Using Loop Until

```
1Ø CLS
2Ø DIM i, Sum, j AS INTEGER
3Ø i=1ØØ: Sum= Ø
4Ø DO
5Ø       PRINT "Sum of all Squares from Ø to "; i;
6Ø             FOR i=Ø TO i
7Ø                   Sum=Sum + j^2
8Ø             NEXT
9Ø        PRINT " =";Sum
1ØØ      Sum=Ø
11Ø      REM This initialization is to clear last Sum
12Ø       PRINT
13Ø      i = i -1Ø
14Ø LOOP UNTIL i=Ø
15Ø PRINT "DONE"
16Ø END
```

§4.1.1.4 WHILE-WEND

While-Wend is another looping block, it has a similar structure with the first Do-Loop structure discussed. This structure is used in file processing *(just a matter of preference)*. The syntax is as shown below:

```
 WHILE condition(s)
             ...statement(s)
 WEND
```

The listing of LST4.1.1g shows a simple file processing with While-Wend structure.

LST4.1.1g Using While-Wend

```
1Ø CLS: DIM Rec$
2Ø OPEN "C:\MISC\LINP.txt" FOR INPUT AS #1
3Ø REM This program assumed you have executed the program
4Ø REM listing of LST3.3.1b once
5Ø REM Program to Output the content of the file
6Ø WHILE NOT EOF (1)
7Ø     LINE INPUT Rec$
8Ø  PRINT Rec$
9Ø WEND
1ØØ PRINT "EXECUTED"
11Ø END
```

The advantage of using the WHILE over the DO is that the number of records is not required before the entire record can be accessed. The keyword EOF means ‘End of File’ and the value 1 signifies the file number, so the *function EOF() returns* true when the

Cursor reaches the Last line in the open file. Lines 22Ø-25Ø of LST3.3.1b can be replace with 6Ø-9Ø of LST4.1.1g

§4.1.2 CONDITIONAL STATEMENTS

Although conditional statements have been used in most of the codes, it is now time to formally discuss conditional statements. There are three of such to be discussed here.

§4.1.2.1 IF-THEN-[ELSE]

This is a straight line conditional 'structuring' statement of the format:

IF *LogicalCondition* THEN *lineNumber/Command ELSE Linenumber/Command*

The code below LST4.12a shows a simple use of this format:

<u>LST4.12a Using IF-THEN I</u>

```
1Ø CLS: DIM Rep$, Days
2Ø INPUT "Enter A or B"; Rep$
3Ø IF Rep$= "A" THEN PRINT "Typed A" ELSE PRINT "Typed B"
4Ø INPUT "How many days are in January 2ØØ1"; Days
5Ø IF Days=31 THEN 8Ø ELSE 6Ø
6Ø  PRINT "You missed the answer!"
7Ø GOTO 9Ø
8Ø PRINT "Correct!"
9Ø END
```

The ELSE 6Ø of LST4.1.2a may be ignored. If Days is not 31 the Program will not go to line 8Ø but the next line which is still 6Ø.

The limitation of the 'one- line' IF-THEN statement is that only one command is executable i.e. the statement after the THEN keyword is only one. The next section caters for this 'problem' or incapability.

§4.1.2.2 BLOCK-IF

The Block-If statement syntax is:

```
IF LogicalCondition THEN
        Statement(s) executed if condition is true
[ELSE]
        Statement(s) executed if condition is not true
END IF
```

The statements could be other block-Ifs. The ELSE is optional but there must be the END IF to signify the end of the block-If. Any attempt to run the cod without the End-If causes an un-trappable error-Block IF without END IF. Notice carefully that unlike the one line IF THEN earlier discussed; the THEN after the logical condition **does not have** either any line number or statement directly following it on a line. Any attempt to put such in the front of THEN stops it from being a Block-IF structure, therefore running the code would cause an error- END IF without block IF. Also placing the Block in a FOR-NEXT loop or a similar loop should be carefully done. You must complete the block IF before ending the loop, otherwise an error of the types described above is flagged and may be confusing to the new programmer.

The program listing of LST4.1.2b shows a simple use of Block IF statement.

LST4.1.2b Using Block-IF

```
1Ø CLS: DIM Age
2Ø INPUT "Enter the Age of the Pupil"; Age
3Ø IF Age<12 THEN
4Ø      IF Age>7 THEN
5Ø         PRINT   "This pupil to watch the movie"
6Ø      ELSE
7Ø         PRINT "Movie not recommended for this pupil"
8Ø      END IF
9Ø  ELSE
1ØØ     IF Age<18 THEN
11Ø             PRINT "Grade 2 Film Recommended"
12Ø      ELSE
13Ø             PRINT "Mature, Any Grade Recommended"
14Ø      END IF
15Ø  END IF
16Ø  END
```

The End-IF of 8Ø closes the IF of 4Ø while that of 14Ø closes IF of 1ØØ. The IF of 3Ø closed at 15Ø. Note the order of the closing and my style of indenting.

§4.1.2.3 SELECT-CASE

This is another conditional statement structure which, I see as 'neater' than block-IF, can be used for as series of conditions follows the syntax below. It can also have embedded within it other select case structure(s) as the statement(s):

```
SELECT CASE variable
     CASE ConditionI
          Statements
     CASE ConditionII
```

SideTalk
For All the select Case structures there must be corresponding END SELECT

```
                    Statements
            CASE ConditionIII
                    Statements
            CASE ConditionN
                    Statements
            [CASE ELSE]
                    Statements
    END SELECT
```

The statements are executed if the conditions are true most of the times the case condition are chosen that only one of the condition would be true otherwise a bug may crawl into your code. Notice the optional [CASE ELSE] segment, it is for execution only when none of the conditions is evaluated as true. The program listing of LST4.1.2c shows a simple use of the Select Case Structure.

SideTalk

Whatever is quoted in the front of PRINT is a String. Do not confuse the *new line* delimiter "**:**" with the colon used here

LST4.1.2c Using Select Case

```
10 CLS
20 DIM Age%
30 INPUT "Enter Age of Candidate"; Age%
40 SELECT CASE   Age%
50     CASE IS <=12
60         PRINT "Candidate: Infant"
70      CASE 13 TO 19
80          PRINT "Candidate: Teenager"
90       CASE 20 TO 79
100      PRINT "Candidate: Adult"
110      CASE 80 TO 89
120        PRINT "Candidate: Octogenarian"
130      CASE ELSE
140          PRINT "Candidate: Over Aged"
150 END SELECT
160 END
```

LST4.1.2d “Embedded” Select Case

```
1Ø CLS
2Ø REM Program for Salary Scale of a Small Coy
3Ø DIM StaffID$, StaffCat$
4Ø DIM StaffOldSal, StaffNewSal AS SINGLE
5Ø PRINT “New Adjustment to Old Salaries of Staff”
6Ø INPUT “Staff Category (Senior/ Junior )”; StaffCat$
8Ø INPUT “Last Salary =N=”, StaffOldSal
9Ø REM S or J could be typed to represent the Staff
1ØØ REM category for convenience.
11Ø StaffCat$= MID$(StaffCAt$,1,1) : REM i.e. First Letter
12Ø StaffCat$=UCASE$(StaffCat$) : REM Capitalize the letter
13Ø SELECT CASE StaffCat$
14Ø         CASE “J”: REM Junior Staff
15Ø             SELECT CASE StaffOldSal
16Ø                   CASE IS < 5ØØØ.#
17Ø                       StaffNewSal=1.25*StaffOldSal
18Ø                       REM 25% Increase
19Ø                    CASE ELSE
2ØØ                        StaffNewSAl=1.15*StaffOldSal
21Ø                         REM 15% Increase
22Ø              END SELECT
23Ø           CASE “S” : REM Senior Staff
24Ø                   SELECT CASE StaffOldSal
25Ø                   CASE IS < 5ØØØØ.#
26Ø                       StaffNewSal=1.13*StaffOldSal+ 1ØØØØ.Ø
27Ø                       REM 13% Increase and 1Ø,ØØØ bonus
28Ø                    CASE ELSE
29Ø                        StaffNewSAl=1.Ø8*StaffOldSal+4ØØØØ.Ø
3ØØ                         REM 8% Increase and 4Ø, ØØØ bonus
31Ø              END SELECT
32Ø           CASE ELSE: REM Wrong Category entry for staff
33Ø                PRINT “Wrong Category entry for staff”
335             PRINT “PRESS ANY KEY”
34Ø              DO: LOOP UNTIL INKEY$ <> “”
35Ø               CLS
36Ø               GOTO 5Ø
37Ø        END SELECT
38Ø        PRINT “New Salary =N=”; StaffNewSal
39Ø        END
```

The indentation of the Case(s) helps to watch for the positioning of the End –Select. This becomes helpful especially when the Select Case is embedded within Select-Case. LST4.1.2d shows a simple “embedded” select case structure.

§4.1.3 ITERATION

The solving of some real mathematical equations can only be achieved by iteration. Iteration is a repetitive substitution of values into an equation from values gotten from the equation starting with an initial guess. There are various schemes used to iterate, these include but not limited to: Newton-Raphson iterative scheme, direct subject of the formula and so on plus the Gauss-Seidel iterative scheme (see chapter 6).

Although this is not a mathematical text, I shall take pain to explain briefly iterative schemes and write codes in QBASIC.

§4.1.3.1 NEWTON-RAPHSON

Given a function f (x) =0 to solve for x, the Newton Raphson iterative scheme requires that you find the first derivative of f(x) (i.e. $f'(x)$ or $\frac{\partial f}{\partial x}$). The solution of f(x) =0 is given by

$$x_N = x_0 - \frac{f(x_0)}{f'(x_0)} ----------------(4.1)$$

Where x_0 is the most recent value obtained or the initial guess value, and x_N is the value evaluated from equation 4.1 the value x_0 is assigned x_N and re-substituted in the equation 4.1 until the difference between x_N and x_0, *before substitution*, goes to a set condition.

If after a large number of iteration the difference is rather growing out of bounds the system of equation is said to be diverging, if it is not diverging and the value of x_N is always getting close to x_0, after subsequent iteration, the system is said to be converging. To have a more elaborate background on Newton-Raphson consult book(s) on Calculus and Analytical Geometry. Fig 4.1.3 shows the Algorithm for Newton-Raphson scheme.

```
1.  Define the function f(x)=0
2.  Take the first derivative fprime(x)=0
3.  Define convergence criteria (epsilon) eps=Value
4.  Set a counter=0 and the Max iteration number ItMax
5.  Set a guess value for the Solution(root) $x_0$ = Init
6.  Evaluate $x_N$ from the equation $XN = X0 - \frac{f(X0)}{fprime(X0)}$
7.  Compute convergence, say, $conv = \frac{(XN - X0)}{XN} * 100\%$
8.  Assign X0=XN
9.  Increment counter (Counter =Counter +1)
10. Compare the conv and eps (True or False)
11. If step 10 is false, compare counter
                and ItMax (Counter > ItMax= True/False)
12.  If step 11 evaluates true, stop there is no convergence
13. If step 10 is true, i.e. is true, i.e. system has converged
    then stop the process
14.  Output the convergence result
15. stop the process
```

Fig 4.1.3 Algorithm -NR

The program listing of LST4.1.2e attempts to find the roots of x^3 + x -2=0. (Note: the equation must be equated to zero then make f(x) =0 i.e.

$f(x) = x^3 + x - 2 = 0$

where the derivative $f'(x) = 3x^2 + 1$)

SideTalk

Dr. J. A. OMOLEYE (Mechanical Engineering Department –University of Ilorin- has a modified N-R scheme to evaluate other roots of a polynomial at a go. It should be seen that the N-R algorithm is for a single root!

<u>LST4.1.2e N-R Programming</u>

```
1Ø REM A simple NR program by:
2Ø '                    A.A. ADENIYI
3Ø '
4Ø '*******************************************
5Ø  DIM Counter, ItMAX AS INTEGER: DIM Conv
6Ø DIM Init, XN,XØ,Eps AS SINGLE
7Ø Counter=Ø : ItMAX=1ØØØ: Eps=Ø.ØØØ1
8Ø PRINT "N-R Solution of F(x)=x^3 + x – 2 =Ø"
9Ø INPUT "Enter the guess root" ; Init
1ØØ XØ=Init
1Ø5  DO
11Ø  XN=XØ-(XØ^3+ XØ-2)/(3*XØ^2 +1)
12Ø Conv=ABS((XN-XØ)/XN)*1ØØ
13Ø XØ=XN
14Ø Counter=Counter +1
15Ø LOOP UNTIL (Conv<=Eps) OR (Counter> ItMAX)
16Ø IF Counter> ITmax THEN
17Ø     PRINT "No convergence after Iterations ="; Counter
18Ø    PRINT "Retry with another guess value"
19Ø  ELSE
2ØØ     PRINT "Converged"
21Ø     PRINT "Root x="; XN
22Ø    PRINT "Iterations="; Counter
23Ø END IF
24Ø END
```

§4.1.3.2 "Subject of the formula"

By subject of the formula I mean attempting to make the variable split into parts where it will exist on both sides of the equation but only the single variable is on one side i.e. given a function g(x) such that we can express x=f(x). for example given $g(x)=x^2 +4x-10 =0$ we can write $x=(10-x^2)/4$ and $x = \sqrt{10 - 4x}$

With these two functions we can iterate for the two roots (if the equations will converge). Standard mathematical text books will give more light on conditions for convergence in iterations.

Unlike the Newton-Raphson scheme there is no need for differentiation as X_N=f(x_0 for example to solve x^2 + 4x - 10 we use:

$$x_N = \frac{(10 - x^2{}_0)}{4} - - or - - x_N = \sqrt{10 - 4x_0}$$ in place of

$$x_N = x_0 - \frac{(x_0 + 4x_0 - 10)}{(2x_0 + 4)}$$

You may want to try Newton-Raphson and the "Subject of the formula" method for this system of equation and compare the number of iterations. There is one advantage here; the two equations (or the number of equations) will converge to different roots whereas Newton-Raphson convergence depends on the initial choice. If an equation has solutions x=2.5 and x= -1. Initial guess values of x=3 and x=-1.5 will give different convergence to evaluate the two roots.

§4.2 SUMMARY

Controlling the flow of program depends on the ability of the programmer to use looping statements and conditional statement plus his intuitive reasoning as regards pitfalls in codes.

Looping can be achieved in about four ways or more ranging from the simple FOR-NEXT structure to the WHILE-WEND. Loops can be simple or complex as the situation may necessitate.

This chapter discussed three conditional statement structures from the simple IF-THEN to the versatile SELECT-CASE structure. To close the chapter a very important tool in computation was discussed, iterative scheme with the development of algorithm and sample program on iteration.

§4.3 QUIZ

(1) To wait for user to press any key appeared in which program listing(s) in this chapter?

§4.4 PROJECT

(a) Develop a program to solve a polynomial of the form

$$f(x) = \sum_{i=0}^{n} a_i x^i$$

$f(x) = a_0 + a_1x + a_2x^2 + \ldots\ldots\ldots. a_nx^n$

Where a_0, a_1, a_2 ……….a_n are constants to be entered at run time.

(Use N-R scheme or otherwise)

(b) Develop a program to compute GPA of Students using the GPA format following:

$$GPA = \frac{\sum_{i=1}^{N} (Credit_i \times Po\,\mathrm{int})}{\sum_{i=1}^{N} Credit_i}$$

$Credit_i$ =Credit Unit of each registered course i

$Point_i$ = Point in course i

Where point is obtained from:

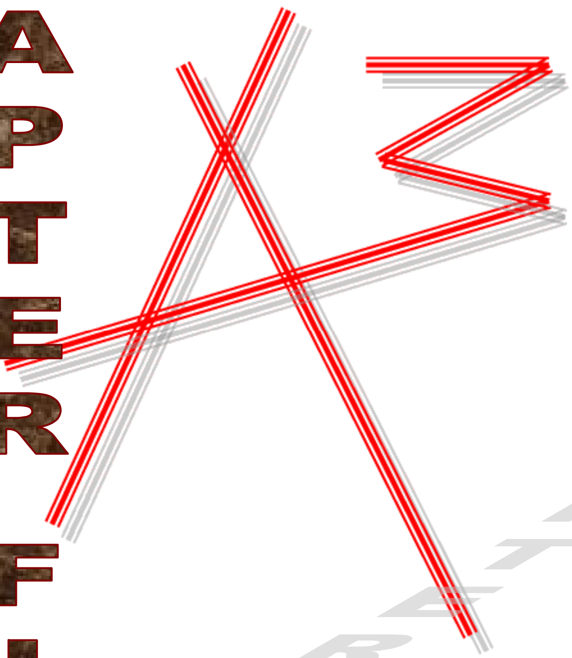
SCORE	**POINT**
70-100	5
60-69	4
55-59	3
50-54	2
0-40	0

(Hint: Use 10 courses to make project easy. The results should be entered: -Use Select Case and Do-Loops)

Bibliography

- A. A. ADENIYI (2003) – Computer Analysis for Presentation of Students Results Using Visual Basic – Mechanical Engineering Departmental Project, University of Ilorin.
- Dr. J. A OMOLEYE – Unpublished lecture note on QBASIC for MEE5Ø5 (University of Ilorin)
- Erwin Kreyzig (1999) – Advanced Engineering Mathematics, John Wiley & Sons, INC, New York

CHAPTER FIVE



CHAPTER FIVE

§5.1 ORGANIZING A PROJECT

A completely design program to perform a specific set of tasks is termed project in this book. A project should be designed to be robust i.e. should be able to stand test of time as regards error input and *life span*. A well-organized project like every other projects starts from good planning.

To write a very good project, pencil work is very essential. It is highly tempting to sit by a computer and start programming without prior planning. Although one may write code that will "perform" what is expected of it but organized planning cannot be overemphasized (I used to be victim).

The steps below would be a good guide but it is by no means the standard set by Microsoft Corporation (See http://www.msn.com for more). The steps have proven good in my various projects:

1. Defining the task to be performed
2. Developing Algorithm on the task
3. Testing Algorithm with sample(expected) input
4. Repeating (2) if (3) does not work well
5. Developing flowchart (if convenient)
6. Setting out variables
7. Setting out variables
8. Programming in modules (parts)
9. Testing (with different users and repeating necessary steps when errors are found) and Debugging. You use functions keys F4(View output), F8 (Step run), F3 (Search) etc .

After reading this chapter you should be able to write good and "user friendly" codes as well as understandable codes. The chapter takes you through using functions and subroutines.

§5.1.1 Remarking, Dimensioning and Programmer-friendly – Declarations

Remarking

This is putting comments at strategic positions in your code. This does not in any way affect the output or the program flow. The aim is to assist the programmer or any person going through the program to get a feel of what is going on at various stages in the program. The keyword used is REM or the apostrophe (‘) sign. Any other statement or "stories" written in front is ignored by the interpreter.

<u>SideTalk</u>
If you don't use REM in your code you may soon get confused with your own code!

Do not see remarking as a source of high memory for your code it is not! However it is boring to boring to over-remark.

Dimensioning

To dimension a variable is to allocate or reserve memory space for the variable. Variables could be static variables dimensioning is DIM or REDIM. The REDIM is used to re-dimension an earlier dimensioned variable.

<u>SideTalk</u>
DIM SHARED is used to allow a variable to be passed to a subroutine

When naming variables, it is advisable to use “names” that you will easily recognize. Using variables like x, y or t can be confusing when you are managing a big project (as experience has shown). You can use names of up to 256 characters! However, you cannot combine characters like underscore, #,*,-,&, / or other “non-regular characters” but you can combine alphabets A-Z with numerals 0-9. It should be noted that keywords should not be used as variable names. There should be no space between and numerals should not start a name.

The following names are good examples of variables: Telephone$, XAxisValue, Gradient, Interest, MatA(Row, Col) etc.

Programming-friendly Declarations

Just as variables should be “friendly-dimensioned”, functions and subroutines should be given friendly names. All these friendliness discussed is not for the machine but the programmer. The following function names are friendly enough: Grad(x1,x2,y1,y2), Cot(x),Log10(x), Parse(Text$) etc and the subroutines following are friendly named: Delay, Sort, WelcomeScreen, TExitScreen, and TestPrint.

Note that keywords are not to be used to name either functions or subroutines. Note that Screen, Exit and Print are keywords.

Unlike variables, functions and Subroutines are Declared instead of Dimensioned. The syntax is

DECLARE FUNCTION *functionName*(Variable(s) type(s)) AS *Type*

e.g. DECLARE FUNCTION Log10(X AS SINGLE) AS SINGLE

Where Log10 is the function name

X is the parameter passed to the function

"AS SINGLE" is the type returned by the function (the X AS SINGLE means a variable X of type SINGLE is passed to the Function.

To Declare a Subroutine, the syntax is

DECLARE SUB *subroutineName*(*Parameter(s)Passed)*

For example:

```
DECLARE SUB WelcomeScreen()
DECLARE SUB TestPrint(NPages%)
DECLARE SUB TExitScreen(x%, y%)
```

If you do not type the DECLARE line, after running the code, the statements are automatically included in your IDE (like a *template*) but you can edit (*the signatures*) if the variable types are not similar to what you desire. Note however that no statement, not even the traditional CLS or REM should come before DECLARE keyword in your code.

§5.2 FUNCTIONS

There are two kinds of functions: The Built-in and the User Defined functions. If you ever used calculators such as CASIO FX 991™ or Purpo™ you will find functions on the keyboard (built-in) also you can insert formulas to perform some calculations (User-Defined).

§5.2.1 Built-in functions

These are functions that come with the "machine" –QBASIC comes with some set of functions such as SIN( ), COS( ), LOG( ), SGN( ) etc. To get more of the built-in functions see the Help file that comes with your QBASIC.

§5.2.2 User-Defined Functions (UDF)

As the Built-in functions cannot satisfy all the forms of equation/functions that exist there is a need to be able to create functions to be able to create functions that are user defined. User defined functions can be a combination of Built-in functions and others functions.

<u>SideTalk</u>

The trigonometric functions built-in with QBASIC are evaluated in Radians i.e.

$SIN(30) \neq 0.5$

You have to convert to

$SIN(30 \times \frac{\Pi}{180}) = 0.5$

The log function is not to base 10 but to base **e** i.e. $LOG(10) \neq 1$ but

$Log(10) = 2.302585092994045684...$

UDF can be built instead of repeatedly writing "same code". User Defined Functions are basically two in structure as discussed below.

§5.2.2.1 DEF FN

This is the simpler of the user defined functions. It does not require the DECLARE statement earlier discussed, the syntax being:

DEF FN *functionName*(*Variable1*[,*Variable2]*[,…])=*Expression code*

The use requires you to pass the variables values to the FN *FunctionName()*. The program listing of LST5.2 shows a simple use of DEF FN form of UDF.

LST5.2 Using DEF FN

```
10 REM Program to Display Log to base 2
20 ‘   from 10 to 15 in steps of 0.2
30 DEF FNLog2(x) =LOG(x)/LOG (2)
40 CLS: DIM Values AS SINGLE
45 PRINT “Value--------------------------------Log Base 2”
50 FOR Values=10 TO 15 STEP 0.2
60        PRINT Values, FNLog2 (Values)
70 NEXT
80 END
```

Observe that the variable name used in defining the function in LST5.2 live 30 is x but another variable (Values) was used when the function was “called”. It is just to show you that the variables are passed by *reference*. You can pass any variable of similar type. If in the example just given, you type the function as Log2(Value) instead of FNLog2(Value) of line 60, you receive error or 0 is given as the result.

Such functions must be prefixed with FN to show that they are Functions (and not just a variable or a subroutine)

The variable(s) list must be similarly passed otherwise an error is generated. If a function is defined as

```
DEF FNGrad(x1, x2, y1, y2) = (y2-y1)/(x2-x1)
```

The call to the function must pass four variables (i.e. values) to the function FNGrad any attempt to run the line FNGrad(2,4) will give an error. Running FNGrad(2,3,6,8) will give a good value.(Running FNGrad(0,0,3,4) will crash your program unless you put error trapping codes-(why?)- See 5.5).

§5.2.2.2 Declare Function-Function

A more flexible user defined function type is discussed now. The function defined earlier can only be defined on a single line and various conditions cannot be easily programmed.

<u>SideTalk</u>
Type the keyword FUNCTION and the function name then press ***enter*** to go to the Module. End Function is added automatically.

The structure requires the DECLARE FUNCTION structure discussed earlier. The structure is:

```
DECLARE FUNCTION fname(x,y,z…)
.
.
.
FUNCTION fname(x,y,z,…)
    Statement(s)
    [EXIT FUNCTION]
    [Statement(s)]
END FUNCTION
```

As earlier mentioned, typing the DECLARE FUNCTION line is "optional"; it is automatically included if not typed. The FUNCTION-END FUNCTION part does the computation or necessary logic. Although you can type the function anywhere in the code, it is automatically put in a module. If you are inexperienced you may feel your code is gone. You can access all the modules by pressing the **F2** function key and select the function you want to edit (you may decide to include or remove variable(s) passed to the function-it is allowed, but you have to be careful).

When I write functions I keep them at the end of my main program in this book. The program listing of LST5.2b shows a simple use of function.

LST5.2b Using Function-End Function

```
DECLARE FUNCTION grad(x1,x2,y1,y2) AS SINGLE
10 CLS
20 REM Evaluating gradient of slopes
30 PRINT "Enter coordinates A(x1,y1) and B(x2,y2)"
50 PRINT "      FOR 5 POINTS"
60 DIM Inc AS INTEGER
70 FOR Inc =1 TO 5
80       INPUT "x1, y1"; x1, y1
90       INPUT "x2, y2"; x2, y2
100       PRINT "Slope of line ="; grad(x1,x2,y1,y2)
110 NEXT
120 END
```

```
FUNCTION grad(x1,x2,y1,y2)
130 DIM Numerator , Denom AS SINGLE
140 Numerator=y2-y1
150 Denom =x2-x1
160  IF Denom=0 THEN
170      PRINT "SLOPE Vertical /Same Point A=B"
180  grad=8888888.8888888888888;: PRINT "Infinity"
190 ELSE
200      grad=Numerator/Denom
210 END IF
END FUNCTION
```

Obviously, this type of function is flexible and can be multi-purpose. Observe the use of the function name. it does not require the prefix FN unlike the DEF FN that requires it. When equating the name *(grad)*, the parenthesis was not included (see LST5.2b lines 180 and 200).

There can be as many functions as required in a project and the functions can call themselves as well as a function call itself (recursion). However, care should be taken because too deep calling may cause *out of stack memory error.*

§5.3 SUBROUTINES

Some activities need to be performed similarly in more than one place in a project. To avoid wasting space or unnecessary making the project very large, subroutines are written. Subroutines are similar to functions in writing and placing but unlike a function, a sub does not return a value. Every other thing like passing variables or parameters is similarly done.

This book identifies three *types* of subroutines: “GOTO”, GOSUB, and CALL-SUB structures. See following sections for more.

§5.3.1 GOTO

GOTO statement is not a standard subroutine statement. It is used to transfer control to a line number (or line label). Note that it is spelt GOTO and not GO TO! The structure is:

```
100 Statements                play: Statements
120 GOTO 90                   done: Statements
                              GOTO play
```

The line labels are play and done with colon in the front.

It is regarded as a bad programming style to use GOTO as a subroutine programming tool; even a good programmer limits the use of GOTO in use.

§5.3.2 GOSUB

GOSUB has a pair RETURN which must be encountered at the end of the subroutine action. There may be many GOSUBs in a project but there must be at least one RETURN to be encountered. Many GOSUB statements can refer to a single line number (where the RETURN statement is).

The structure is:

```
.
.
.
GOSUB lineNumber/lineLabel
.
.
LineNumber/lineLabel

-----------
RETURN
```

The program listing of 5.2c shows a simple use of GOSUB

<u>LST5.3a Using GOSUB/RETURN</u>

```
10 CLS
20 PRINT "GOSUB PROGRAMMING"
30 GOSUB 100
40 GOSUB 150
50 GOSUB 300
55 GOSUB 500
60 END
'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 100 PRINT "AT 100"
110 PRINT "AT 110"
150 PRINT "AT 150"
300 PRINT "AT 300"
400 RETURN
500 PRINT "TRYING SECOND RETURN STATEMENT"
510 PRINT "LINE 510"
520 RETURN
```

Note that all the GOSUB statements encountered RETURN although there are only two RETURN statements but four GOSUBs.

Another program on quadratic equation using GOSUB is shown in LST5.3b.

LST5.3b GOSUB II

```
10 CLS : DIM a, b, c, d, x1, x2
20 INPUT "a, b, c "; a, b, c
30 d = b ^ 2 - 4 * a * c
40 IF d = 0 THEN GOSUB 1000
50 IF d > 0 THEN GOSUB 2000
60 IF d < 0 THEN GOSUB 3000
70 PRINT "DONE"
80 END
1000 x1 = b / (2 * a)
1010 x2 = x1
1020 PRINT "Equal Roots x ="; x1
1030 RETURN
2000 GOSUB 4000
2010 x1 = (-b + SQR(d)) / (2 * a)
2020 x2 = (-b - SQR(d)) / (2 * a)
2030 PRINT "X1 ="; x1; " X2 ="; x2
2040 RETURN
3000 GOSUB 4000
3010 PRINT "X= ";
3020 PRINT b / (2 * a); " +/- j"; SQR(-d) / (2 * a)
3030 RETURN
4000 PRINT "THE SOLUTION OF THE QUADRATIC EQUATION”
4010 PRINT a; " X^2  +"; b; "X   +"; c; "  =0"
4020 RETURN
```

Make LST5.3b robust e.g. avoid a=0 or string entry

§5.3.3 CALL-SUB

Subroutines are written like the functions of section 5.2.2.2. To transfer control to a subroutine, the keyword CALL is used (although it is optional you use CALL- you can just type the SUB Name and it is called). The DECLARE statement for Subs is:

DECLARE SUB *SubName(Parameter(s))*
.
.
SUB *SubName(Parameter(s))*
-
-
-
END SUB

The function key **F2** is also used to access Subs. The program listing of LST5.3c shows a use of CALL-SUB.

LST5.3c Call Sub

```
DECLARE SUB Animate (Text$)
DECLARE SUB Welcome ()
DECLARE SUB REQUEST ()
10 CLS
20 DIM SHARED Text$
30 Welcome
REM Also CALL Welcome
40 DO
50    REQUEST
```

```
60 LOOP UNTIL Text$ = "Quitting the Animation"
70 END
80 REM
SUB Animate (Text$)
180 CLS : DIM Ltxt
190 COLOR 14: REM Give color to the text
200 Ltxt = LEN(Text$)
210 DIM i% 'Delay
220 FOR i% = 1 TO Ltxt
230     PRINT MID$(Text$, i%, 1);
240      SLEEP 1' FOR Delay = 1 TO 95100: NEXT
250    NEXT:  PRINT : PRINT
260     FOR i% = Ltxt TO 1 STEP -1
270     PRINT MID$(Text$, i%, 1);
280      SLEEP 1 '   FOR Delay = 1 TO 39100: NEXT
290      NEXT
291   PRINT : PRINT
292   PRINT "Press Any Key"
293   DO: LOOP UNTIL INKEY$ <> "": CLS
END SUB

SUB REQUEST
150 PRINT : PRINT : PRINT "Quit to Stop"
160 INPUT "Type your Text"; Text$
170 CLS
171 IF Text$ = "Quit" THEN Text$ = "Quitting the Animation"
172 Animate (Text$)
END SUB
```

```
SUB Welcome
100 CLS
110 Text$ = "Welcome to Simple Animation"
120 Text$ = Text$ + "Brought to you by AcubeSoft of Nigeria"
140 Animate (Text$): PRINT: PRINT
END SUB
```

§5.4 USER DEFINED TYPES

User (i.e. programmer) has the type-block handy to make special type of definitions. This *Type* definition assists in organizing projects. A project on books in the library may need user defined types to ease the many declarations that may be required. The book project example will be used to throw more light on types. The structure is:

```
TYPE VariableType
          Dimension statements
END TYPE
.
.
.
DIM Variable as VariableType
.
.
.
Variable.DimensionStyle=...
.
.
.
```

The program listing of LST5.4a shows a typical use of the Type structure.

```
LST5.4a Using Define Types
10 CLS
20        TYPE Book
30                  Author AS STRING * 15
40                  Publisher AS STRING* 25
50                  Edition AS STRING * 13
60                  YrPublished AS INTEGER
70        END TYPE
80 DIM Text AS Book
90 INPUT "Enter Author's Name"; Text.Author
100 Text.Publisher = "Université de AcubeSoft"
110 Text.Edition = "1st Edition"
120 Text.YrPublished = 2004
.
.
END
```

The advantage of the Type structure becomes apparent when you need an array of such "type". (See chapter six for more on arrays). The example below (LST5.4b) uses array with Type structure.

LST5.4b Music Type Example

```
10 CLS
20 TYPE Music
30          Composer AS STRING*25
40          Album AS INTEGER
50       Producer AS STRING*25
60          CopiesSold AS INTEGER
70 END TYPE
80 DIM NCas, i%
90 INPUT “Enter no of cassette”; NCas
100 DIM Song(NCas) AS MUSIC
120 FOR i%=1 TO NCas
130     PRINT “For song”; i%; “Please Supply the following data”
140         INPUT “COMPOSER : “;Song(i%).Composer
150         INPUT “PRODUCER : “;Song(i%).Producer
160         INPUT “ALBUM  : “;Song(i%).Album
170       INPUT “COPIES SOLD : “;Song(i%).CopiesSold
180 NEXT
190 CLS
200 PRINT “Make Request for a song”
210 PRINT “Specify Number between 1 & “;NCas
220 INPUT i%
230         SELECT CASE i%
240             CASE 1 TO NCas
241                         PRINT “For the input we found result as:”
250                         PRINT “COMPOSER : “;Song(i%).Composer
260                         PRINT  “PRODUCER : “;Song(i%).Producer
270                         PRINT “ALBUM  : ” ;Song(i%).Album
280                         PRINT “COPIES SOLD : ”;Song(i%).CopiesSold
290             CASE ELSE
300                     PRINT “Invalid Input”
310       END SELECT
320   END
```

Do you have a feel that it is pretty good to use Types? However, Type structure may not be supported by your version of QBASIC. The Visual Basic programmer will find it very useful.

§5.5 ERRORS

Errors are bound to occur during the course of running or programming. This chapter briefly discusses errors; the discussion is brief for you have to get some experience on your own from which you will soon develop your own philosophy about errors.

§5.5.1 Program-Time

When programming, two types of errors can occur, the two are: (i) Syntax Error and (ii) Semantic Error.

Syntax error is an error that occurs as a result of wrongly “syntaxing” of keywords. A very common one to beginners (programmers) is typing “T” and Zero i.e. TØ for TO in the FOR-NEXT loop. The interpreter is intelligent to respond to syntax errors by flagging suggestions e.g. “Expected TO”, “FOR WITHOUT NEXT”, “NEXT WITHOUT FOR”, “SELECT CASE WITHOUT END SELECT”, “EXPECTED IDENTIFIER” etc.

The intelligence of the interpreter, however, should not be taken as a complete wizard. Following “all” the suggestion may completely change the idea of the program you are coding! To avoid many errors, strictly follow the structures of the keywords or statements to the letters.

Semantic errors are errors that the interpreter (or the compiler or the assembler) will not catch! Semantic error occurs as a result of either or all of the following:

i. Mistyping variables
ii. Wrong Algorithm
iii. Not clearing memories or wrong initialization of variables
iv. Wrong placement of line or lines of codes
v. Counting loops with non-integers etc

(i) Mistyping variables

Some programming languages have facilities for handling this type, for example Visual Basic uses ***Option Explicit*** to track variables misspelt. If you have the following lines of code in QBASIC

```
1ØØ XO=5Ø
12Ø PRINT XØ
```

Line 1ØØ is "X" and letter "O" whole line 12Ø is letter "X" and number zero. The expected output is 5Ø in line 12Ø but the output is zero or the value X0 was before line 1ØØ. This is a very serious bug in programs. (Note that QBASIC is not case sensitive, do not use same letters of different cases to mean different things, it will rather be updated! e.g.

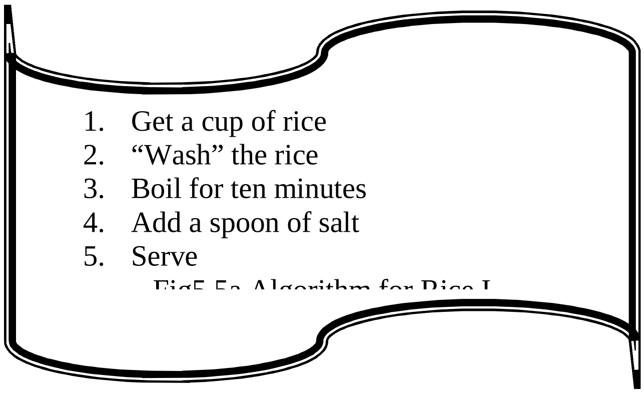
```
1Ø Ada=5ØØ
.
.
2ØØ ADA=15
..
.
5ØØ aDA=2Ø
52Ø PRINT Ada
```

The output is 2Ø not 5ØØ, this is a simple semantic error if you expected the result to be 5ØØ because of the difference in cases.

(ii) Wrong Algorithm

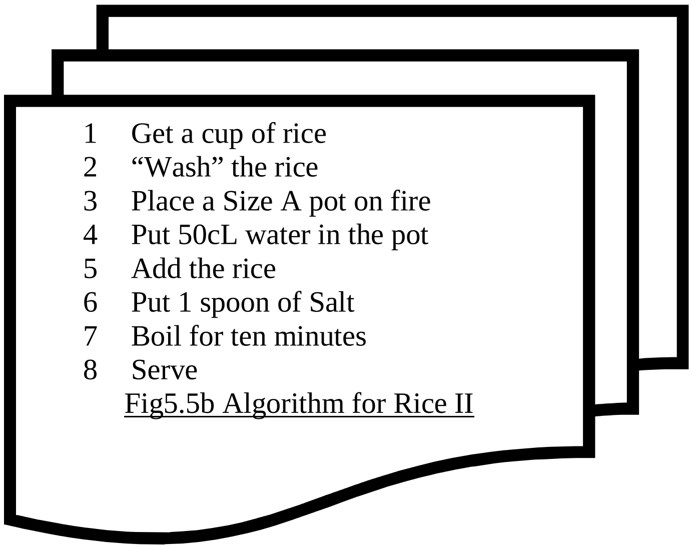
It is vital that your Algorithm is correct otherwise you may end up getting unexpected output (Garbage-In-Garbage-Out: GIGO).

Do not see computer as a genius. It only responds fast to your ingenious programs in a very fast way than you could do. Compare the two Algorithms of Figures Fig5.5a and Fig5.5b. They are intended to prepare Rice.



```
1.  Get a cup of rice
2.  "Wash" the rice
3.  Boil for ten minutes
4.  Add a spoon of salt
5.  Serve
```

Fig5.5a Algorithm for Rice I



```
1   Get a cup of rice
2   "Wash" the rice
3   Place a Size A pot on fire
4   Put 50cL water in the pot
5   Add the rice
6   Put 1 spoon of Salt
7   Boil for ten minutes
8   Serve
```

Fig5.5b Algorithm for Rice II

Did you observe any bug in Fig5.5a? You may bother to study it before I point them out.

Comparing Fig5.5a and Fig5.5b

5.5a does not specify the size of the pot to be used for the cooking (a big bug!), The quantity of water required to cook the rice was not specified, the rice of Fig5.5a may be prepare well but the salt distribution will be poor, in Fig5.5b you add salt before the boiling not after!

Although the two Algorithms are not perfect but has pointed out how wrong Algorithm could cause error.

(iii) Not Clearing Memories or Wrong Initialization

The "expected" value of i% by the programmer of the lines below is 53 but a rather different output was got!

```
110  i%=0 : REM Initialization
111  FOR i%=1 TO 50
112       PRINT Mat (i%)
113  NEXT
114  I%=I%+3
115  PRINT I%
```

The code assumes Mat is a matrix earlier defined, the Output of line 115 is 54 not 53, find out why.

§5.5.2 "Run-Time"

Pressing the function key F5 runs your code. All what happens during this period is referred as Run Time. Any error that happens here is referred as Run-Time Error. What causes error here can be either/all of:

- Wrong input
- Program crash
- Wrong Code etc

(i) *Wrong input*

DOS traps this kind of error by giving a response “Redo from Start”. If an expected input is an integer and a String is supplied or inputs to be separated with comma(s) are not rightly supplied, this error is encountered until the right input is supplied. The statement INPUT a, b, c expects input line 5, 3, 4.05 etc

(ii) *Program Crash*

A program crashes if the flow encounters lines of code that are malicious. The following could crash your code:

a. Using an array without pre-dimensioning

b. Using an array beyond its *dimensioned capacity*

c. Assigning a value beyond the dimension type e.g. if X is an integer, assigning value X=890809808080808080902344554421 will crash your code (see Table 3.1 in Chapter 3)

d. Requesting for unavailable drive(s) e.g. The use of OPEN keyword on an unconnected disk drive.

e. Encountering lines causing division by zero.

f. Running out of stack space (e.g. as a result of over looping-break down loops to avoid this)

g. etc

(iii) Wrong Code

Wrong syntax in code may not all be trappable when doing the programming depending on the size of your code or the complexity of the syntax, any attempt to run the part of the program crashes the program. The good thing is suggestions are made; however, it may not help!

(iv) Wrong placement of line or lines of codes (or Counting with non-Integers)

If you count loops with non-integer variables you may fall into error pit, so watch out! You may get errors if you position the line or lines of codes wrongly. Your Algorithm may be right but the placement matters. The following set of programs will give different outputs

```
C= 10
DO
C=2*C
PRINT C
LOOP UNTIL C>100
```

```
C= 10
DO
PRINT C
C=2*C
LOOP UNTIL C>100
```

§5.6 TRAPPING ERRORS

Most errors that might occur in your programs can be envisaged during your programming. QBASIC has a way of tracking down some trappable errors using error statements. The following are used:

ON – ERROR / RESUME Structure

ERR.*NUMBER* Statement

(i) ON ERROR/RESUME

(a)The structure is:

```
lineNumber/LineLabel: ON ERROR GOTO lineNumber/Label
   .
   .
   .
lineNumber/LineLabel: Statements
   RESUME lineNumber/label/NEXT
```

(b) *lineNumber/Label:* ON ERROR RESUME NEXT

The two types cause program flow to be transferred to a line number / or label or to the next line if an error occurs. The program listing of LST5.6a shows a simple use of error statements.

LST5.6a Error Program I

```
10 ON ERROR GOTO handler
20 OPEN "A:\text.dat" FOR INPUT AS #1
30 REM Try to open a file in a floppy
40 DIM i: i=1: DIM Ex AS INTEGER
45 REM Ex=100000000000000000000 'See Explanation
50 DO
55 i=i+1
60  GET i, NameTxt$
70  PRINT NameTxt$
80 LOOP UNTIL i=3
90 END
100 REM Error handling part
Handler:
PRINT "FILE NOT FOUND OR"
PRINT "FLOPPY NOT READY"
PRINT "CHECK FAULT"
PRINT "ENTER R TO RESTART"
PRINT "ENTER S TO STOP"
IF INPUT$(1) = "R" THEN RESUME 10 ELSE RESUME 90
```

Place the error handling "subroutine" such that it cannot be encountered unless there is an error. It is traditional to place Error handling routines after the END statement. The code of LST5.6b handles the Error of the fact that there would be a division by zero.

LST5.6b Error Programming II

```
10 CLS: DIM x AS SINGLE
20 PRINT “Y = 11/(2-x)”
30 PRINT “Values near the asymptote of Y”
40 DEF FNY(x)=11/(2-x)
50 ON ERROR RESUME NEXT
60 FOR x=1.9 TO 2.1 STEP 0.001
70    PRINT “x = “; x
80     PRINT “Y = “; FNY(x)
90 NEXT
100 PRINT “AT ASYMPTOTE Y= Infinity”
120 END
```

An error handling subroutine could have been written to get a better output,

```
Handler: PRINT “Infinity”
         RESUME NEXT
```

This is placed after line 120 but the line 50 would look like this:

```
50 ON ERROR GOTO Handler
```

(ii) ERR.NUMBER is used to return the error code for the particular type of error. The various trappable errors have numbers used to recognize them, so that wrong information is not supplied. The code LST5.6a would always flag error even if you have a floppy with file name A:\test.dat, but the flag would indicate that the file is not found, assuming line 45 were not REM. The error would have been that the value assigned to Ex (an integer) is beyond the maximum integer value. To check for such as simple "Select Case Err.Number" could be used to see if the error, the "Case Else" could message that the error is something else.

§5.7 IMPLICATION OF ERRORS

The errors you commit in your programs could go a long way to affect lives of people, it may come in financial, health or otherwise. The following story should "warn" you.

Some year before 1997, the operator of a pool maintenance company in New Jersey got the idea to use computer to monitor his customers' pool heaters. Using a microcomputer and a modem, he developed a program that would connect by phone with the heaters, check for correct operation and adjust temperature.

After a few weeks of operation, he got a frantic call from one of his client – the water in the pool was 100^0 and rising! He drove to the pool and adjusted thee the temperature by hand. Later he was able to find a bug in his program. Fortunately, nobody was injured by the scalding water *(P.38 Berman).* Should you jump to such a pool, would you wish well for "whoever is responsible- the programmer!"

This other story exists; A programming error in an **AT & T**™ Telecommunication switch shut down AT & T long-distance service for 9 hours blocking approximately 5 million calls*(P.39 Berman).*

Also a software error at Bank of New York lost an estimate of $5million in 1985 *(P.39 Berman).*

The programmer should always aim for a bug free code. According to Mary Laude, a member of the Technical staff at Sun Microsystems and a Test engineer, fixing an error in the field costs about 3 times as much as fixing the same error before release [to the market]*(P.43 Berman).*

One of my friends (a programmer) once said, jokingly, that he can put some bug in his codes so that he will be called for maintenance in the nearest future. I told him it was a bad habit to do that. You don't have to bug your code before you will be called for a maintenance routine. If you see it as your duty to perform maintenance, it is the duty of the maintenance department or section of the organization to decide whether to give you to maintain or give the contract to other developers, if they are not satisfied with your code. I'll prefer you sell your ability so that you may be called for other projects or recommendations be made about you. Care has to be taken in whatever step you take in life, for the future may not forgive!

§5.8 SUMMARY

This chapter has taken you through basics of organizing a QBASIC project. It discussed making Remarks and using friendly codes. You have gone through Functions and Subroutines as well as using error traps in programming.

The following chapter takes you through array programming and some useful Algorithms.

§5.9 QUIZ

How do you pass a value from one Subroutine to another in a module? *Attempt: Use DIM SHARED*

§5.9 PROJECT

Design and program a customized Calculator for the following functions: ln(x); Cos^{-1}x ; FahCel(x) and CelFah(x).

FahCel: Fahrenheit to Celsius

CelFah: Celsius to Fahrenheit converter.

The program should give room for user to select the kind of function she wants.

Hint 1: Use Do-Loop and Select Case for the Input$. The calculator should display the function list (with a Sub) as shown below:

```
******************************************
*       SELECT THE FUNCTION NUMBER       *
*----------------------------------------*
*       1.      In(x)                    *
*       2.      ArcCos(x)                *
*       3.      FahCel(Temp)             *
*       4.      CelFah(Temp)             *
*       0.      Quit                     *
******************************************
```

Hint 2: Use a convergent series to find In or ArcCos see a standard text for the series e.g.:

- Advanced Engineering Mathematics – Erwin Kreyzig
- Engineering Mathematics- K. A Shroud

Bibliography

- Berman A. Michael (1997), Data Structure VIA C++: Objects by Evolution, Oxford University Press, New York (Oxford).

CHAPTER SIX

CHAPTER SIX

§6.1 WORKING WITH ARRAYS

An Array is defined as an impressive display or a series. It is also defined as a type of data structure that has a multiple values. In the programming sense, an array is an "organized" storage of data with a variable. By being organized does not necessarily mean that it is a sorted or an arranged collection, it is organized in the sense that "Virtual positioning" exists in an array

In an array, a Matrix-like positioning is used to place data. The Mathematical equivalent of programming array is the Matrix. If you have not worked with Matrices before, the following examples should give you a picture. Let us define a Matrix M as a store of information on the Prices of 3 Books in 3 years.

$$Mat.M = \begin{bmatrix} 250.00 & 259.50 & 400.00 \\ 80.30 & 100.00 & 150.50 \\ 1000.00 & 1020.00 & 1500.60 \end{bmatrix}$$

The columns represent the years while the rows represent the books. Matrix M could be expressed in clearer terms as:

$$Mat.M = \begin{bmatrix} \Pr ice.of.A.in.Yr.1 & \Pr ice.of.A.in.Yr.2 & \Pr ice.of.A.in.Yr.3 \\ \Pr ice.of.B.in.Yr.1 & \Pr ice.of.B.in.Yr.2 & \Pr ice.of.B.in.Yr.3 \\ \Pr ice.of.C.in.Yr.1 & \Pr ice.of.C.in.Yr.2 & \Pr ice.of.C.in.Yr.3 \end{bmatrix}$$

In mathematics, each item of the matrix is called an element. For easy programming, it is interesting to be able to point to an element of a matrix. The matrix M is also presented in elemental terms:

$$Mat.M = \begin{bmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \\ M_{31} & M_{32} & M_{33} \end{bmatrix}$$

That is, an element of a matrix could be represented as Mij, $\forall i, j \in Z$ (i.e. where i and j are integers). The i represents the row and j represents the column. The Matrix below represents a generalized

Matrix A. $A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & . & a_{1m} \\ a_{21} & a_{22} & . & . & a_{2m} \\ . & . & . & a_{ij} & . \\ a_{n1} & . & . & . & a_{nm} \end{bmatrix}$

§6.1.1 Dimensioning Arrays (DIM & REDIM)- Matrix

The size of the matrix A above is n X m. by size, it does not mean the amount of bytes or the memory unit, but it means the allocated memory locations for the data. It is necessary to pre-allocate memory for an array before it is used anywhere in the program, although if the array size is less than 10 it may not be required, however, it is a good practice to dimension your arrays to prevent error.

DIM and REDIM are used to tell your computer (or QBASIC) to either allocate space or reallocate space respectively for your matrix. The syntax is:

DIM *VariableName(Integer[,Integer][,Integer])*

REDIM *VariableName(Integer[,Integer][,Integer])[AS type]*

The AS *type* could be included if the data type is known, it is safer to leave it out when in doubt. However, it must be included if the data type is a string. The following are typical examples:

```
DIM MatA(5Ø,1Ø), MatB(1Ø,1Ø)

DIM X(5Ø), ExtraLag(5,1Ø,3)

Dim SoxN(1Ø,1Ø) AS STRING

DIM Age$(12,1Ø) : ' This is string Array
```

The default starting position in rows or columns counting is from 1 but you may change the default Dimensioning to start from zero by placing the statement OPTION BASE Ø before any DIM or REDIM keywords. (You may want to put OPTION BASE 1; to tell your code reader that you count from 1, though not necessary)

§6.1.2 Manipulating Arrays

A database becomes useless if it could not give real life values. To make sense with a set of data it has to be manipulated. There are various ways by which an array (matrix) could be manipulated. You may even have your own format, sections 1 to 5 give some manipulations on arrays.

§6.1.2.1 Matrix Addition

Like we perform addition on non-array variables it is possible with matrices, but unlike the addition of the latter, the former requires elemental addition. Addition of matrices is the addition of corresponding elements of the matrices. This implies that two matrices A and B can only add if they have same dimension (the two must be string types if you are to add strings or each element be converted (to string) before the addition).

A man owns two book shops where he sells mainly four types of books. The sales for three days in the shops are respectively:

$$ShopA = \begin{pmatrix} 5 & 3 & 5 \\ 4 & 4 & 2 \\ 6 & 2 & 0 \\ 0 & 5 & 10 \end{pmatrix} \leftarrow ------ \rightarrow ShopB = \begin{pmatrix} 16 & 20 & 30 \\ 1 & 3 & 1 \\ 0 & 0 & 4 \\ 10 & 3 & 2 \end{pmatrix}$$

The listing below (LST6.1a) shows a simple (non-flexible) program he used to find the sale from the two shops.

LST6.1a A Simple Matrix Addition

```
1Ø CLS

2Ø PRINT "BIG JOE BOOKS, IKARE"

3Ø REM SHOP A

4Ø DATA 5,3,5,4,4,2,6,2,Ø,Ø,5,1Ø

5Ø REM SHOP B

6Ø DATA 16,2Ø,3Ø

7Ø DATA 1,3,1

8Ø DATA  Ø,Ø,4

9Ø DATA 1Ø,3,2

1ØØ REM Compare the two Arrangement of data !

11Ø REM

12Ø DIM ShopA(4,3), ShopB(4,3) , Sales(4,3)

13Ø REM  The array sizes are equal.

14Ø REM Put the data in the Arrays

15Ø DIM Rows, Cols AS INTEGER
```

```
16Ø REM Shop A
17Ø  FOR Rows=1 TO 4
18Ø     FOR Cols=1 TO 3
19Ø               READ ShopA(Rows,Cols)
195    REM Why not put Code Here?
2ØØ     NEXT
21Ø   NEXT
22Ø   FOR Rows=1 TO 4
23Ø     FOR Cols=1 TO 3
24Ø               READ ShopB(Rows,Cols)
25Ø     NEXT
26Ø   NEXT
265 'YOU COULD HAVE PLACE 24Ø IN 195 TO AVOID 22Ø-26Ø
27Ø REM Find Sum and Output Result
29Ø  REM ****************************************************
295  PRINT "SALES FROM 2 SHOPS"
296 PRINT " IN 3 DAYS ON FOUR BOOKS"
3ØØ PRINT "DAY1", "DAY2", "DAY3"
31Ø FOR Rows=1 TO 4
32Ø     PRINT "BOOK"; Rows
33Ø     FOR Cols=1 TO 3
34Ø        Sales(Rows,Cols) = ShopA(Rows,Cols)+ShopB(Rows,Cols)
35Ø         PRINT Sales(Rows,Cols),
36Ø     NEXT
37Ø        PRINT
38Ø   NEXT
39Ø END
```

SideTalk

Remove the PRINT of line 37Ø in LST6.1a and watch out for a mess up

If you have 3 matrices, one for Surname the other for Middle name and the last for First name, of 50 people. It should not be difficult to get a new matrix MatNames to contain the whole names (Surname: made Capital and a comma placed in front; other names with the first letters capital like this: ADENIYI, Akinola Abdul. The dimension is something like Dim A(5Ø) etc.)

§6.1.2.2 Matrices Multiplication

Of more interest in mathematics is the multiplication of matrices. It is highly important that you follow the order used in matrix multiplication otherwise, you end up getting figures but they will merely be garbage!

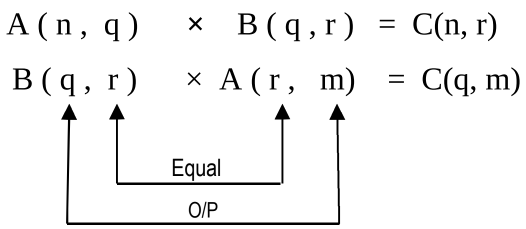
Suppose you have two matrices A(n,m) and B(q,r) i.e. the dimension of A is $n \times m$ and B is $q \times r$. For you to be able to multiply matrices A and B

(a) **m must be equal to q** (i.e. $m=q$), the *column of the first matrix* must be equal to the Column of the second

(b) **multiplication of matrices is not commutative** (i.e. $A \times B \neq B \times A$)

(c) **resulting matrix,** i.e. product matrix **has a size n × r**

e.g.


A (n , q) × B (q , r) = C(n, r)
B (q , r) × A (r , m) = C(q, m)
Equal
O/P

The example below is the multiplication of two matrices A(4,3) and B(3,2)

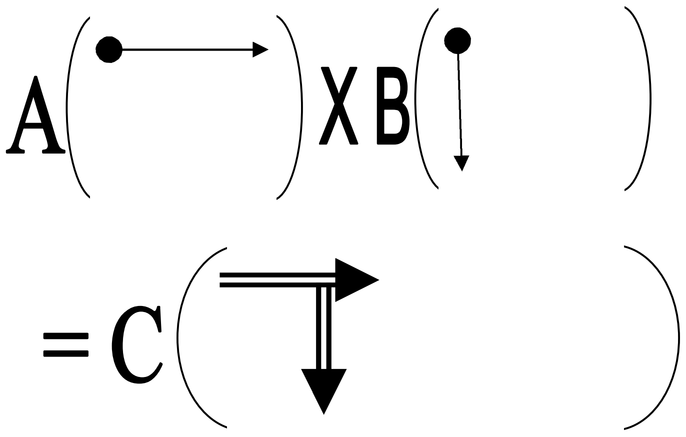
$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{pmatrix} \qquad B = \begin{pmatrix} 10 & 1 \\ 3 & 2 \\ 1 & 6 \end{pmatrix}$$

(Although you cannot add A + B but A × B is possible-why? Can you get B × A?)*"See the 7-structure and note that Dot represents multiplication i.e. 3.4=12"*

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{pmatrix} \times \begin{pmatrix} 10 & 1 \\ 3 & 2 \\ 1 & 6 \end{pmatrix} = \begin{bmatrix} 1.10+2.3++3.1 & 1.1+2.2+3.6 \\ 4.10+5.3+6.1 & 4.1+5.2+6.6 \\ 7.10+8.3+9.1 & 7.1+8.2+9.6 \\ 10.10+11.3+12.0 & 10.1+11.2+12.6 \end{bmatrix}$$

$$= \begin{pmatrix} 19 & 23 \\ 61 & 50 \\ 103 & 77 \\ 145 & 104 \end{pmatrix}$$

To see the 7 structure look at this:



Does this still make sense? C= Row of A X Col of B summed- How far?

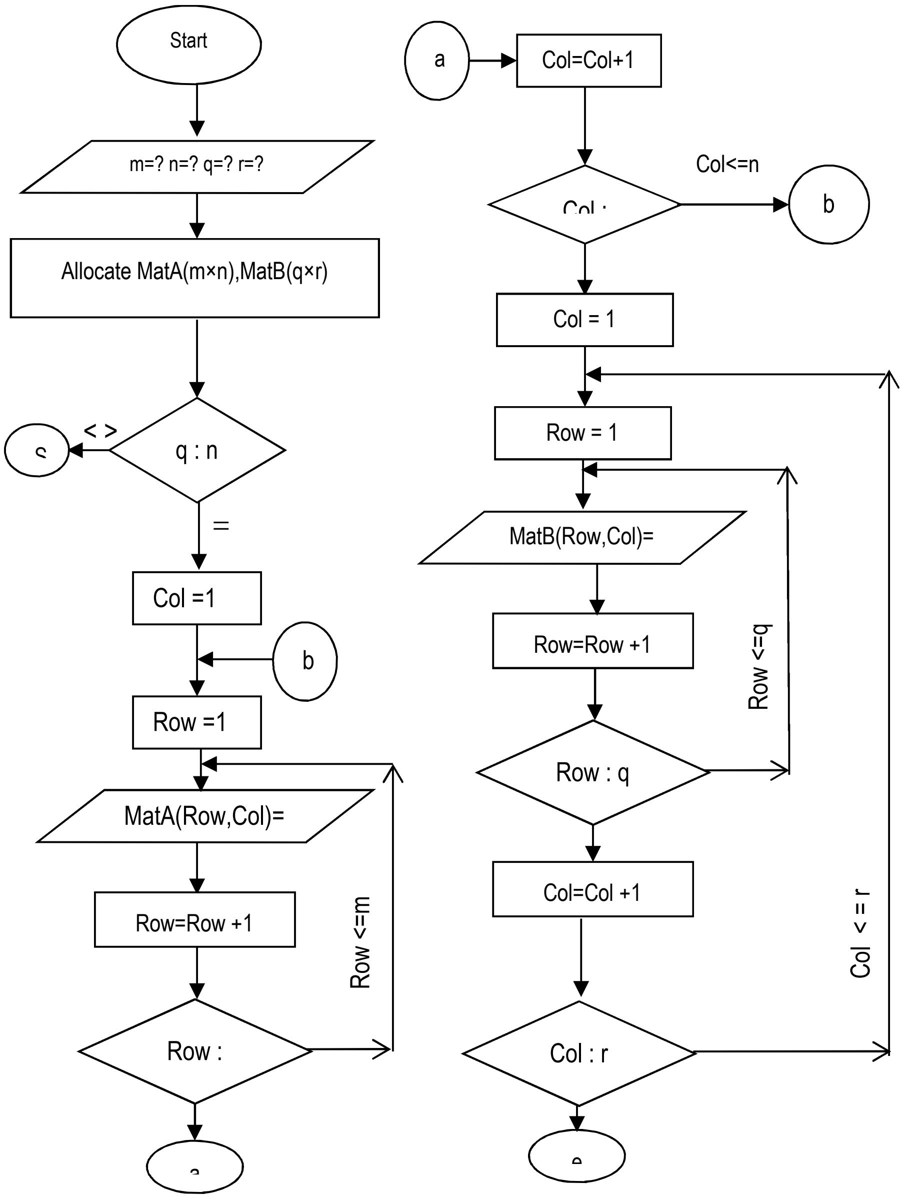
The question now is how do you write a QBASIC code to effect matrices multiplication? Just as in development of other programs, let us do it by Algorithm – coding method.

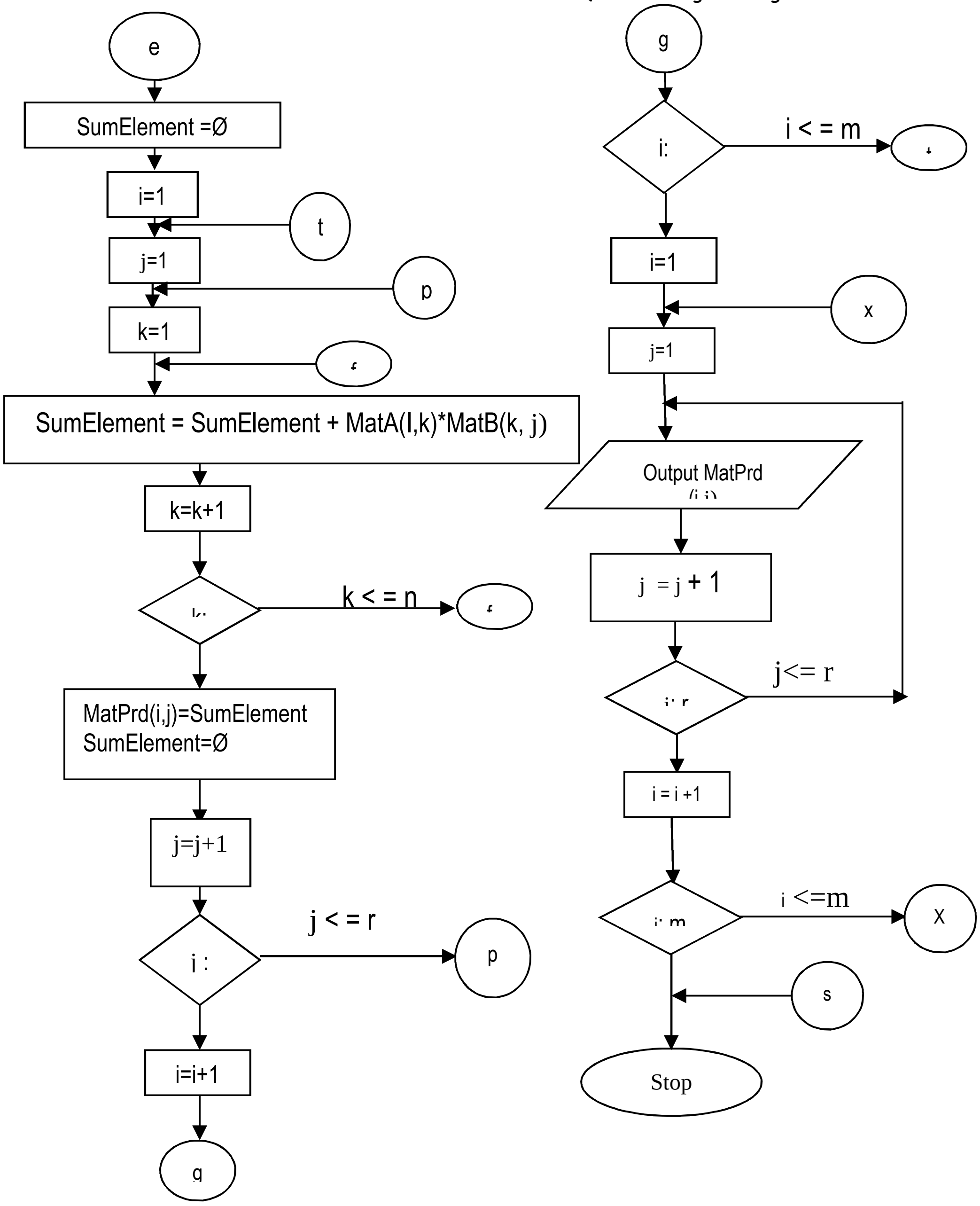
Figure Fig6.1 shows an algorithm on multiplication of two matrices MatA and MatB.

1. Request the array size of each matrix: MatA and MatB say MatA(m×n) and MatB(q×r)
2. Check the condition for multiplicity of the two matrices i.e. n=q
3. Flag error and stop if n≠q else Dimension MatPrd(m×r)
4. Request/Enter the data for the matrices
5. Multiply the rows (elements) of MatA with the columns (elements) of MatB individually and sum. This is done for each row and corresponding columns to get the new matrix MatPrd(m×r)
6. Output the result MatPrd(m×r)
7. Stop

Fig 6.1 Matrices Multiplication Algorithm

The Algorithm would be made clearer by using the flowchart of Fig6.2 (Note however that this flowchart is not the best, but to assist a beginner – think about improving it)





The flowchart and Algorithm above is hereby converted to a QBASIC code (see LST6.1b)

LST6.1b Matrix Multiplication Code

```
1Ø CLS: DIM m, n, q, r AS INTEGER
2Ø PRINT “MULTIPLICATION OF MATRICES”
3Ø PRINT “*********************************”
4Ø PRINT : PRINT “To multiply A(m×n) and B(q×r)”
5Ø INPUT “Enter m”;m
6Ø INPUT “Enter n”;n
9Ø INPUT “Enter q”; q: INPUT “Enter r”; r
1ØØ PRINT: PRINT
11Ø IF q<>n THEN 999
12Ø DIM MatA(m,n), MatB(q,n), MatPrd(m,r)
13Ø REM “Entering matrix A”
14Ø DIM Col , Row
15Ø FOR Col =1 TO n
16Ø      FOR Row =1 TO m
17Ø         PRINT “A (”; Row; “,”; Col; “) =”;
18Ø         INPUT MatA(Row,Col)
19Ø         PRINT “    ”;
2ØØ       NEXT Row    : PRINT
21Ø NEXT Col
22Ø REM Entering Matrix-B
23Ø FOR Col =1 TO r
24Ø FOR Row = TO q
25Ø      PRINT “B(”;Row; “;”;Col;”) =”;
26Ø      INPUT MatB (Row ,Col)
27Ø      PRINT “   ”;
```

```
28Ø NEXT Row
29Ø     PRINT
3ØØ NEXT Col
31Ø DIM SumElement, i, j, k
32Ø SumElement =Ø
33Ø FOR i=1 TO r
34Ø     FOR j =1 TO r
35Ø         FOR k =1 TO n
36Ø           SumElement = SumElement + MatA(i,k)* MatB(k,j)
37Ø         NEXT k
38Ø           MatPrd(i,j)=SumElement
385           SumElement=Ø
39Ø     NEXT j
4ØØ   NEXT i
41Ø REM Output Section
42Ø     PRINT "PRODUCT OF A×B"
43Ø     FOR i =1 TO r
44Ø       FOR j =1 TO m
45Ø               PRINT MatPrd(i,j),
46Ø     NEXT j
47Ø     PRINT
48Ø     NEXT i
49Ø     PRINT "DONE"
5ØØ     PRINT "******************************"
999     PRINT "USE ANOTHER TIME…THANK YOU":
1000    END
```

§6.1.2.3 Matrix Determinant

In many applications in Physics, Mathematics, Economics and Engineering, Matrix determinants are of invaluable use. The determination of determinants is discussed below. Note however that the complexity of determinants is a function of the dimension of the matrix.

A square matrix is a matrix with number of rows equal the number of columns. You find determinants of square matrices. I shall explain determinants of matrices starting with the 2×2 square matrix shown below:

$A(2\times 2) = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$, the determinant is represented as

$\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}$ and the determinant Det A or $\Delta A = a_{11} \cdot a_{22} - a_{21} \cdot a_{12}$

(Note the cross multiplication)

Another matrix, but a 3×3 square matrix A(3×3)

$A(3\times 3) = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$ and the determinant is

$$DetA = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11}\begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} - a_{12}\begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} + a_{13}\begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix}$$

$$= a_{11}(a_{22}a_{33} - a_{32}a_{23}) - a_{12}(a_{21}a_{33} - a_{31}a_{23}) + a_{13}(a_{21}a_{32} - a_{31}a_{22})$$

Note: The sign coefficients of the elements are $(-1)^{i+j}$ where i, j are the row and column numbers respectively.

Let us now consider the 4×4 matrix A (4×4)

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

$$DetA = a_{11}\begin{vmatrix} a_{22} & a_{23} & a_{24} \\ a_{32} & a_{33} & a_{34} \\ a_{42} & a_{43} & a_{44} \end{vmatrix} - a_{12}\begin{vmatrix} a_{21} & a_{23} & a_{24} \\ a_{31} & a_{33} & a_{34} \\ a_{41} & a_{43} & a_{44} \end{vmatrix} + a_{13}\begin{vmatrix} a_{21} & a_{22} & a_{24} \\ a_{31} & a_{32} & a_{34} \\ a_{41} & a_{42} & a_{44} \end{vmatrix} - a_{14}\begin{vmatrix} a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \\ a_{41} & a_{42} & a_{43} \end{vmatrix}$$

The breakdown continues as with the 3×3 matrix determinant above.

The program listing of LSt6.1c shows a simple program to find the determinant of a 3×3 matrix.

SideTalk

LST6.1c is a kind of On The Fly Programming (OTFP)-No good for a serious programmer, though the job is done!

LST6.1c Determinant for a 3×3 matrix

```
1Ø REM Program Determinant
2Ø CLS
3Ø PRINT "Enter the Matrix A(3×3)"
4Ø PRINT "******************************"
5Ø DIM N  AS INTEGER, I AS INTEGER, J AS INTEGER
7Ø DIM SHARED A(N,N)
```

```
8Ø FOR I=1 TO N
9Ø    FOR J=1 TO N
1ØØ        INPUT "A"; A(I,J)
11Ø       REM Take next
12Ø    NEXT
13Ø  NEXT
14Ø SUM=Ø
15Ø REM Pass the crossed matrix to a function
16Ø 'to determine a 2×2 determinant
17Ø DIM COL,ROW
18Ø FOR COL=1 TO N
19Ø       SUM=SUM+(-1)^(1+COL)*CROSSEDMAT(COL)*A(1,COL)
2ØØ  NEXT
21Ø       PRINT "DETERMINANT =" ; SUM
22Ø END
FUNCTION CROSSEDMAT(COL)
23Ø REDIM MAT(2,2)
24Ø DIM CLM
25Ø FOR I=1 TO 2
26Ø     FOR J=1 TO 2
27Ø                   SELECT CASE COL
28Ø                        CASE IS =1
29Ø                             CLM=J+1
3ØØ                   CASE IS =2
31Ø                                  IF J=COL THEN CLM=J+1 ELSE CLM=J
32Ø                   CASE IS =3
33Ø                              CLM=J
34Ø                   END SELECT
35Ø       MAT(I,J)=A(I+1,CLM)
36Ø       NEXT
365   NEXT
37Ø       CROSSEDMAT=MAT(1,1)*MAT(2,2)-MAT(2,1)*MAT(1,2)
END FUNCTION
```

§6.1.2.4 Cramer's Rule

To solve a problem involving n-unknowns, it requires n-equations (which are not linearly dependent). The assumption is that the variables are linear (i.e. not to power greater than 1). There are various methods by which a system of simultaneous linear equations can be solved.

If you ever did elementary mathematics, you should be familiar with the substitution and elimination methods. Some of these methods do not prove viable always especially when the order of the "matrix" system (i.e. the simultaneous equations) is big. It is expected that there should be other more powerful systems (especially those that can be coded for a computer for use in industrial or research application).

Cramer's rule and Gauss-Seidel iterative methods are methods treated in this book but note that there are other methods: Gaussian Elimination, Gauss-Jordan to mention just two.

The Cramer rule would by explained for the 33 matrix system below:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1$$
$$a_{21}x_1 + a_{21}x_2 + a_{23}x_3 = b_2 ---------S1$$
$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3$$

This system is expressed in matrix form as

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} --------S2$$

Note that multiplying back gives system S1

Experience is not taught but gained- Erwin Kreyzig

The solution of the system of equations is:

$$x_1 = \frac{|\Delta_1|}{|\Delta|}$$

$$x_2 = \frac{|\Delta_2|}{|\Delta|}$$

$$x_3 = \frac{|\Delta_3|}{|\Delta|}$$

Where

$$|\Delta| = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix}$$

$$|\Delta_1| = \begin{vmatrix} b_1 & a_{12} & a_{13} \\ b_2 & a_{22} & a_{23} \\ b_3 & a_{32} & a_{33} \end{vmatrix}$$

(Note the replacements)

$$|\Delta_2| = \begin{vmatrix} a_{11} & b_1 & a_{13} \\ a_{21} & b_2 & a_{23} \\ a_{31} & b_3 & a_{33} \end{vmatrix}$$

$$|\Delta_3| = \begin{vmatrix} a_{11} & a_{12} & b_1 \\ a_{21} & a_{22} & b_2 \\ a_{31} & a_{32} & b_3 \end{vmatrix}$$

It is obvious that the solutions exist only if $|\Delta| \neq 0$ - equations not linearly dependent. Obviously, your ability to code the Cramer's rule depends on if you can code determinants. The Algorithm below (Fig6.3) is used for the Cramer's rule.

```
1. Enter matrix A(n×n)
2. Enter matrix  B(n)
3. Determine the determinant $|\Delta|$
4. For n-times, determine determinants $|\Delta_i| \ \forall i = 1,2,...n$
5. For the n-solutions ( $|\Delta| \neq 0$ )
        $x_i = \frac{|\Delta_i|}{|\Delta|}, \ \forall i = 1,2,...n$
6. Output $x_i, \ \forall i = 1,2,...n$
7. Stop
```

Fig6.3 Cramer’s rule Algorithm

Note, when programming, to find determinants, all that is required is to pass the array into the subroutines that handle determinants, say. Note also that you can reserve a dummy matrix whose columns are changed. You may follow the Algorithm below (Fig6.4) to achieve step 4 of Fig6.3

```
1. Define a dummy matrix DumMat(n×n)
2. Define i=1
3. Read  Array A(n×n) into DumMat(n×n)
4. Read into Column I of DumMat(n×n), Array B(n)
5. Pass DumMat(n×n) into the Determinant sub and obtain $\Delta_i$
6. Increment I, until i=n, repeat step 3
7. Stop
```

Fig6.4 The Determinants of many Matrices

At the end of the day you should come up with a very lengthy code (a project indeed). The good thing is that you will have a system for solving an n×n matrix – using your personal software!

The program listing of LST6.1d shows a *crude code* to solve a 3×3 system of equations by the Cramer's rule.

LST6.1d Program – Cramer

```
DECLARE SUB WELCOME()
 REM ----------------------------------------------
 REM PROGRAMMER  AcubeSoft of Nigeria
 REM------------------------------------------------
 REM-------------------------------- 1ØTH JUNE 2ØØ4
DIM A(3,3),B(3)
DIM TempMat(3,3)
CLS
PRINT "*******************************************"
PRINT "***********CRAMER'S RULE*************"
PRINT "*******************************************"
```

```
FOR I=1 TO 3
  FOR J=1 TO 3
        INPUT "A=";A(I,J)
NEXT
 INPUT "B="; B(I)
NEXT
WELCOME
DIM Z, CURRENTCOL, DT, R, C, CMAT
DIM MAIN AS INTEGER
DIM DET(4)
REM FIND THE MAIN DETERMINANT
MAIN=1
REM LOAD THE TEMPORARY ARRAY
FOR Z=Ø TO 3
  CURRENTCOL =Z
  GOSUB LoadTempMat
  GOSUB DETERMINANT
  DET(Z+1)=DT
NEXT
IF DET(MAIN)=Ø THEN
   PRINT "Sorry this system cannot be solved by crammers rule"
   PRINT "….Exiting"
   GOTO LastLine
ELSE
   PRINT "The solutions are:"
     FOR J=1 TO 3
        PRINT USING "X# = #######.###",J, DET(J+1)/DET(MAIN)
      NEXT
END IF
LastLine:  END
```

```
REM ________________________________________
LoadTempmat:
FOR R=1 TO 3
 FOR C=1 TO 3
  IF C=CURRENTCOL THEN
   TempMat(R,C)=B(R)
  ELSE
    TempMat(R,C)=A(R,C)
  END IF
 NEXT
NEXT
RETURN
CROSSEDMAT:
FOR I=1 TO 2
      FOR J=1 TO 2
        SELECT CASE COL
             CASE IS=1
                CLM=J+1
             CASE IS = 2
                 IF J=COL THEN CLM=J+1 ELSE CLM=1
             CASE 3
                  CLM=J
          END SELECT
             MAT(I,J)=TempMat((I+1,CLM)
      NEXT
NEXT
CMAT=MAT*1,1)*MAT(2,2)-MAT(2,1)*MAT(1,2)
RETURN
REM________________________________
```

```
DETERMINANT:
SUM=Ø
REM Pass the Crossed Matrix to find 2×2 Det.
FOR COL=1 TO 3
     GOSUB CROSSEDMAT
      SUM=SUM+(-1)^(1+COL)*TempMat(1,col)*CMAT
NEXT
DT=SUM
RETURN
SUB Welcome()
REM _______________________________
CLS
PRINT "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
PRINT "xxxxx      CRAMMER'S RULE   xxxxxxxxxx"
PRINT "xxxxx FOR SYSTEM OF MATRICES (3×3) xxxxxxx"
PRINT "xxxxx BY: ACUBESOFT OF NIGERIA xxxxxxx"
PRINT "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
END SUB
```

The code you have here is a handy tool for solving, any solvable, 3×3 matrix. Note however that Cramer’s rule is not generally regarded as good for solving matrix systems (the reason is probably for the complexity in finding determinants).

§6.1.1.1 Gauss-Seidel Iteration

This is another method for solving a system of equations of the form AX=B. unlike the previously described methods, it starts by taking a guess set of values for the solutions and then iterate using the "arranged" equations until there is convergence (if it converges).

The Gauss-Seidel iteration is better explained using the matrix system below:

$$\begin{array}{l}
a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + ...+ a_{1n}x_n = b_1 \\
a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + ...+ a_{2n}x_n = b_2 \\
a_{31}x_1 + a_{32}x_3 + a_{33}x_3 + ...+ a_{3n}x_n = b_3 \\
. \quad . \quad . \quad +.....+ . \quad = . \\
. \quad . \quad . \quad +.....+ . \quad = . \\
a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + ...+ a_{nn}x_n = b_n
\end{array}$$

It is not difficult to see that this system can be expressed as

$$x_1 = \frac{1}{a_{11}}(b_1 - a_{12}x_2 - a_{13}x_3 - .... - a_{1n}x_n)$$

$$x_2 = \frac{1}{a_{22}}(b_2 - a_{21}x_1 - a_{23}x_3 - .... - a_{2n}x_n)$$

$$x_3 = \frac{1}{a_{33}}(b_3 - a_{31}x_1 - a_{32}x_2 - .... - a_{3n}x_n)$$

$$. = ...$$

$$. = ...$$

$$x_n = \frac{1}{a_{nn}}(b_n - a_{n1}x_1 - a_{n2}x_2 - .... - a_{nn-1}x_{n-1})$$

On the condition that $a_{ii} \neq 0 \quad \forall i = 1,2,3....n.$ For the iteration, initial guess values are assigned for $x_1, x_2, ...x_n$. These are substituted in the first equation to get a "new" x_1. In the next equation the updated value of x_1 is used to get x_2, this new value is used to get x_3 and the process continues until x_n. The iteration is repeated until the convergence criterion say ξ_{max} is reached using the equation:

$$\xi_{i_{(max)}} = \left| x_i^{old} - x_i^{new} \right| \leq \xi_{max} \quad , \forall i = 1,2,3....n .$$

The equation should firstly be checked to ensure that On the condition that $a_{ii} \neq 0 \quad \forall i = 1,2,3....n.$ if anyone equals zero, the matrix should be rearranged.

To get an easy programming, the equation could be expressed as:

$Lx + Ix + Ux = b$

where L = Lower triangular matrix

I = Identity matrix

U = Upper triangular matrix.

Let me use a 3×3 matrix as an example.

$$5x_1 + 25x_2 + 15x_3 = 10$$
$$24x_1 + 6x_2 + 12x_3 = 12$$
$$x_1 + 25x_2 + 15x_3 = 15$$

This matrix can be expressed as :

$$\begin{pmatrix} 5 & 25 & 15 \\ 24 & 6 & 12 \\ 1 & 2 & 5 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 10 \\ 12 \\ 15 \end{pmatrix}$$

$$or \qquad \begin{pmatrix} 1 & 5 & 3 \\ 4 & 1 & 2 \\ 0.2 & 0.4 & 1 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 2 \\ 3 \end{pmatrix}$$

and

$$\left[\begin{pmatrix} 0 & 0 & 0 \\ 4 & 0 & 0 \\ 0.2 & 0.4 & 0 \end{pmatrix} + \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} + \begin{pmatrix} 0 & 5 & 3 \\ 0 & 0 & 2 \\ 0 & 0 & 0 \end{pmatrix}\right] X = \begin{bmatrix} 2 \\ 2 \\ 3 \end{bmatrix}$$

this is of the form :

$$[\mathrm{L} + \mathrm{I} + \mathrm{U}]X = b$$

Following is an Algorithm for the Gauss-Seidel iteration

```
1.  Enter the Matrices A, B
2.  Check $A_{ii}$
3.  If $A_{ii}=Ø$ quit else proceed to 4 (A more robust version should
    try to rearrange)
4.  Define matrix L, U and b
5.  " $x_n=b-Lx_Ø-Ux_Ø$"
6.  For i=1 … n, Read guess $x_{Ø(i)}$.
7.  Compute $x_{n(i)}$ with $x_n=b-Lx_Ø-Ux_Ø$
8.  Update $x_{Ø(i)}$ for $x_{n(i)}$
9.  Compute error term $\xi_i$
10. Repeat 7 to 9 for i=1 … n
11. Check for convergence
12. Stop at Convergence or repeat 7 if iteration has not exceeded
    maximum number of iterations.
13. Stop {Gauss-Seidel}
```

Fig6.4 Algorithm-Gauss Seidel

The program listing of LST6.1e is to execute Gauss-Seidel iterative scheme.

LST6.1e Program-Gauss Seidel

```
1Ø ON ERROR GOTO LastLine
11 CLS
2Ø PRINT " GAUSS-SEIDEL ITERATION"
3Ø PRINT "INPUT MATRIX AX=B"
4Ø PRINT "                              Where aii <> Ø "
5Ø PRINT : PRINT : PRINT : DIM N AS INTEGER
6Ø INPUT " ENTER THE ORDER OF THE MATRIX N="; N
7Ø REDIM A(N, N), b(N), XNew(N), XOld(N)
8Ø REDIM Eps(N)
9Ø DIM EpsMax: INPUT "ENTER CONVERGENCE Eps"; EpsMax
1ØØ DIM i, j, k AS INTEGER
1Ø1 DIM SumUp, SumLow, Eps(N), True, Cnt
11Ø PRINT "*******************************************"
115 PRINT " BUSY...."
12Ø FOR i = 1 TO N
13Ø     FOR j = 1 TO N
14Ø             PRINT "A("; i; ","; j; ")=";
15Ø                 INPUT A(i, j)
16Ø      NEXT j
17Ø                 PRINT "B("; i; ")=";
18Ø                 INPUT b(i)
19Ø  NEXT i
2ØØ REM TEST FOR aii
21Ø FOR i = 1 TO N
22Ø      IF A(i, i) = Ø THEN ‘ You can modify this line to cater for this type!
23Ø                  PRINT "MATRIX NOT SUITABLE"
24Ø                   PRINT USING " At least A(#;#)=Ø"; i; i
```

```
25Ø                   GOTO LastLine
26Ø       END IF
27Ø NEXT
271    DIM DMat(N)
272 FOR i = 1 TO N
273  DMat(i) = A(i, i): REM Temporarily Store the Column Elements
274 NEXT
28Ø    REM Convert to L and U and b
29Ø    FOR i = 1 TO N
33Ø              b(i) = b(i) / DMat(i)
3ØØ         FOR j = 1 TO N
31Ø              A(i, j) = A(i, j) / DMat(i)
32Ø         NEXT
34Ø      NEXT
341 ERASE DMat
35Ø   REM  Read in Values for L and U
36Ø   FOR i = 1 TO N
37Ø        FOR j = 1 TO N
38Ø            IF j <= i THEN k = Ø ELSE k = 1
39Ø                 U(i, j) = A(i, j) * k
391            IF j >= i THEN k = Ø ELSE k = 1
392                      L(i, j) = A(i, j) * k
393          NEXT
4ØØ       NEXT
41Ø   PRINT "Enter initial guess values"
42Ø   PRINT "Press Z to use Ø throughout"
43Ø    PRINT "Press N to enter values"
```

```
44Ø    IF UCASE$(INPUT$(1)) = "Z" THEN
45Ø         FOR i = 1 TO N
46Ø              XOld(i) = Ø
47Ø          NEXT
48Ø      ELSE
49Ø          FOR i = 1 TO N
5ØØ            PRINT "X "; i; "   = ";
51Ø           INPUT XOld(i)
52Ø          NEXT
53Ø      END IF
54Ø      PRINT "Busy....."
55Ø  SumUp = Ø: SumLow = Ø: Cnt = Ø
56Ø   DO
57Ø  : 58Ø           FOR i = 1 TO N
59Ø              FOR j = 1 TO N
6ØØ                        SumUp = SumUp + U(i, j) * XOld(j)
61Ø                      SumLow = SumLow + L(i, j) * XNew(j)
62Ø               NEXT
63Ø               XNew(i) = b(i) - SumUp - SumLow
64Ø              Eps(i) = ABS(XOld(i) - XNew(i))
65Ø               XOld(i) = XNew(i)
66Ø           SumUp = Ø: SumLow = Ø
67Ø            IF Eps(i) <= EpsMax THEN True = 1 ELSE True = Ø
68Ø       NEXT i
69Ø          Cnt = Cnt + 1
7ØØ      IF Cnt > 4ØØ THEN
71Ø          PRINT "THE SYSTEM DID NOT CONVERGE"
```

```
72Ø           GOTO LastLine: REM You can use EXIT DO here
73Ø        END IF

74Ø   LOOP UNTIL True = 1
75Ø   PRINT "Converged after "; Cnt; "iterations"
76Ø   PRINT : PRINT SPACE$(25); "SOLUTIONS"
77Ø   PRINT
78Ø    FOR j = 1 TO N
79Ø          PRINT USING "X##  = "; j;
8ØØ          PRINT XNew(j)
81Ø          PRINT
82Ø     NEXT
83Ø      PRINT "Done !": PRINT "Yet another Y/N ?"
84Ø     IF UCASE$(INPUT$(1)) = "Y" THEN 1Ø
85Ø     REM CAN YOU SEE WHY REDIM was used at Line 7Ø?
999   END
LastLine: PRINT "ERROR!   The Matrix caused Error "
             PRINT "  Try another [Use other guess]"
             PRINT "Avoid too large Array size"
             PRINT " Thank you for using this program"
1ØØØ GOTO 999
```

§6.2 SORTING ALGORITHMS

There are various Algorithms developed to sort an array. More often than not you will want to sort an array. It may be paramount to find the best way out. While some are easy to code some execute fast. The following are viable ways of sorting arrays: Bubble sort, Shell Sort, Quick Sort, Insertion sort, Exchange sort to mention a few.

§6.2.1 Bubble Sort

It loops through an array and makes comparison with adjacent items of the array to check whether they are out of order. If they are not out of order, keep looping through the array until no items are swapped, this implies a sorted array. Unfortunately this is the slowest of all. The program listing of LST6.2a tries to execute Bubble sort.

LST6.2a Program Bubble Sort

```
CLS
REM GENERATE RANDOM NUMBER ARRAY
RANDOMIZE TIMER
DIM A(5Ø), I, J
FOR I=1 TO 5Ø
  A(I)=INT(1ØØ*RND(Ø))
  PRINT A(I),
NEXT
  PRINT “UNSORTED LIST”
REM DO SORT
1ØØ PRINT “SORTING STARTED =” ; TIME$
FOR I=2 TO 5Ø
   FOR J=5Ø TO I STEP -1
      IF  A(J-1)>A(I) THEN
        ‘SWAP A(J-1), A(J)  OR
         Temp=A(J-1)
```

```
    A(J-1)= A(J)
         A(J)= Temp
       END IF
     NEXT
NEXT
FOR I=1 TO 5Ø
    PRINT A(I);
NEXT
  PRINT "SORTED"
2ØØ PRINT "SORTING ENDED AT ="; TIME$
END
```

§6.2.2 Shell Sort

This algorithm was discovered by Donald Shell about 37 years ago. The method works similarly to the Bubble sort but instead of comparing adjacent items in an array, it takes items that are half-way between the first and the last items in the array, in its initial comparison. If the array is not yet in tune, it swaps them. It then determines the halfway point again and repeats the process. This halving of the items and the logical comparison is not stopped until adjacent items are being compared. This would have ensured that most item in the array are been sorted. A loop through the array, finally, goes same way as the Bubble sort and the array is therefore said to be sorted.

If you have *1GHz Giga pro VIA Samuel processor*, in less than 4 seconds you could sort an array of 5ØØØ integers with Shell sort but that take over 5 seconds for Bubble sort! You may want to compare the speed of sorting of similar arrays by sorting a large arrays on your system, a simple trick would help you get the sorting time: print time (using the statement PRINT TIME$) at the beginning (before sorting) and after sorting, see lines 1ØØ and 2ØØ of LST6.2a.You can compare the data generation time and the sorting time, you will be surprised to see that it take more time to generate than sorting using an efficient sorting Algorithm.

The LST6.2b is for the Shell sort Algorithm.

```
LST6.2b SHELL Sort
CLS
REM Generate the Array
GOSUB GenARRAY
PRINT "SHELL SORT"
PRINT "SORTING Initiated "; TIME$
PRINT "Busy......."
DIM Asize, Inc, Hold
Asize = UBOUND(B)
Inc = Asize \ 2
DO UNTIL Inc < 1
 FOR i = Inc + 1 TO Asize
         Hold = B(I)
     FOR j = i - Inc TO 1 STEP -Inc
       IF Hold >= B(j) THEN
```

```
         EXIT FOR
          ELSE
           B(j + Inc) = B(j)
       END IF
     NEXT j
        B(j + Inc) = hold
   NEXT i
    Inc = Inc \ 2
LOOP
PRINT "SORTED ARRAY…"
FOR i = 1 TO m
  PRINT B(i)
NEXT
PRINT "Array Sort completed "; TIME$
END
GenARRAY:
DIM B(5ØØØ)
PRINT "UNSORTED ARRAY"
FOR i=1 TO 5ØØØ
  B(i)=CINT(RND*1ØØØ)
 PRINT  B(i);
NEXT
PRINT
RETURN
```

§6.2.3 Merge Sort

The Algorithm follows the form:

```
To sort an Array
     If an array has no one entry stop
     SORT (the first Half)
     SORT (the second half)
     COMBINE (the two); i.e. MERGE
STOP
```

The listing of LST6.2d shows a code for the Merging Sort. Assuming an Array A() exists.

```
LST6.2d PROGRAM MERGE SORT

DECLARE SUB MergeSort(A(),Start%, Finish%)
DECLARE SUB  ArrayMerge (A(), Start%, Middle%, Finish%)
REM MERGE SORT
REM Put Code to generate A()
' DIM Mdl,St, Fin : REM Middle, Start, Finish
' DIM B1,E1,B2,E2, TempLoc
REM B=>Beginning   E=> End and TemLoc=> Temporary Location
PRINT "*****************************************************************"
REM The procedure used is a RECURSIVE method where a Sub is REM
CALLED in itself.
MergeSort(A(),Start,Finish)
FOR i=1 TO UBOUND(A)
  PRINT A(i);
NEXT: PRINT "SORTED"
END
```

```
 SUB MergerSort(A(),Start,Finish)
   DIM Mdl  AS INTEGER
       IF Start< Finish THEN
            Mdl=(Start+Finish)\2
             MergeSort A(), Start, Mdl
             MergeSort A(), Mdl+1, Finish
             ArrayMerge A(), Start, Mdl, Finish
         END IF
END SUB
SUB ArrayMerge (A(), Start, Middle, Finish)
     REDIM ATemp( Start TO Finish) :' AS TYPE
                      'Note that you can dim this way using to keyword
DIM B1,E1,B2,E2 AS INTEGER
DIM TempLoc, i AS INTEGER
B1=Start
E1=Middle
B2=E1+1: E2=Finish
TempLoc=Finish
DO WHILE ((B1<=E1) AND (B2<=E2))
        IF A(B1) <=A(B2) THEN
            ATemp(TempLoc)=A(B1)
              TempLoc=TempLoc + 1
              B1=B1+1
        ELSE
            ATemp(TempLoc)=A(B2)
            TempLoc=TempLoc + 1
            B2=B2+1
         END IF
LOOP
IF B1<= E1 THEN
   FOR i=B1 TO E1
       ATemp(TempLoc)=A(i)
        TempLoc = TempLoc +1
    NEXT
END IF
FOR i= Start TO Finish
     A(i)=ATemp(i)
NEXT
END SUB
```

§6.3 SUMMARY

You have been taken through some ways of handling arrays which included allocating memory to a variable as an array, storing data in array variables, making an array as user defined type.

The chapter also takes you through some mathematical applications with matrix by explaining and programming some of the Algorithms.

Finally, this chapter explains sorting arrays. You find codes for Bubble, Shell and Merge sorts, all of which you can modify to suite your future requirements.

§6.4 QUIZ

How many sub 2×2 matrices are obtained by finding the determinant of an 8×8 matrix?

6.5 PROJECT

(a) Design and code a QBASIC code that visually exhibits the performance of the three sorting scheme explained in this chapter.

Hint: The array could be used to determine the heights of bars of different colours (or pie chart of different colours and angles) drawn side by side, the sorting should then be shown. You may read up graphics to know how to use lines (See chapter eight)

(b) Find out the speed of your system and fill the chart below (right in this book (with pencil)

COMPUTER SPEED:	
TYPE: (E.G. PENTIUM IV)	

S/N	ARRAY SIZE	ALGORITHM	TIME START	TIME END	TOTAL TIME TAKEN
1	500	Bubble Shell Merge			
2	1,500	Bubble Shell Merge			
3	4,000	Bubble Shell Merge			
4	10,000	Bubble Shell Merge			
5	500	Bubble Shell Merge			

Comment on your result and possibly forward to engradeniyi@gmail.com.

Bibliography

- Erwin Kreyzig (2001),**Advanced Engineering Mathematics**, 8 Ed., John Willey & Sons, Inc, New York
- Rob Thayer (1998), **Visual Basic 6 Unleashed**, Sams Publishing, United States of America

CHAPTER SEVEN

CHAPTER SEVEN

§7.1 WORKING WITH FILES & DATA

DOS commands can be executed within your QBASIC code. The following is a list of some of the file handling DOS commands:

MKDIR: To make a directory (called folder in Windows OS)

RMDIR: To remove a directory from a disk

KILL: To delete a file from a disk

SHELL: This is a function used to run any .com, .exe, .bat etc file.

```
10 CLS
20 KILL "A:\*.*"
30 END
```

WARNING!

DO NOT RUN THIS MALICIOUS CODE!

No warning, just delete all!

You have these handy, however, care should be taken not to corrupt a file or delete file(s) (You may not be able to retrieve or restore deleted or corrupted files!). Suppose you have a floppy disk on the floppy drive and you run the code:

You can run the following:

LST7.1a Simple File Handling

```
10 CLS: ON ERROR GOTO 9999
20 MKDIR "A:\Folder"
25 PRINT "Folder Created"
30 OPEN "A:\Folder\Trial.dat" FOR OUTPUT AS #1
40 CLOSE 1
50 KILL "A:\Folder\Trial.dat"
55 PRINT "Folder Deleted!"
58 PRINT "LOADING MARIO please wait…"
60 SHELL "C:\MISC\XMARIO.EXE"
70 REM Assuming the game XMario.exe is installed on C:\Misc
80 PRINT "Loaded and done!"
90 END
999 RESUME NEXT
```

§7.1.1.1 Sequential files

At this stage you should have at least a vague idea of what a file means in computer programming. A sequential file is similar to audio tape. To get a data from it you start from the beginning and play till you get to where you want and push stop when you are through. There is no automatic way to get to any location within an audio cassette.

It is advisable to use sequential file only if:

(i) The information rarely changes in the file

(ii) The processing is from “start to end”

(iii) Adding to the file is to the end.

(But try to avoid it if you can).

To create “a sequential file” from your code, use the statement:

OPEN *FileName* FOR OUTPUT AS #*FileNum*

If the file name already exists, the content is deleted! The file name may be a path e.g. C:\MISC|Acube.dat. if the path is not specified, QBASIC places the file in the current directory of the executing program. Not that you cannot use the following characters in naming in DOS \, ?, :,*, <, >, ;, or |. However, you should follow the rules for filenames that DOS imposes.

- 8 Characters or less plus 3 letters extension (optional)
- Characters: A-**Z**, **0-9**, **( )**, **{**, **}**, **@**, **#**, **$**, **%**, **&**, **!**, **-**, **_**, **~**, **/**
- Not case sensitive

<u>Reading from a Sequential file</u>

Use the statement:

OPEN *FileName* FOR INPUT AS #*FileNum*

(where *FileNume*=1,2,3,…)

To read from the open sequential file, use:

INPUT #*FileNum*, *VariableName(s)* for example:

```
OPEN "C:\MISC\Dummy.dat" FOR INPUT AS #1
INPUT #1, Age$
PRINT Age$
CLOSE 1
```

The keyword CLOSE is like pressing STOP after playing a cassette.

Adding to a Sequential File

To add data or record to a sequential file you use the statement:

OPEN *FileName* FOR APPEND AS *#FileNum*

Then you use the WRITE # or PRINT # to do the actual appending as in the example below:

```
INPUT "C:\MISC\Dummy.dat" FOR APPEND AS #2
WRITE #2, "This is an appended text: WRITE #"
PRINT #2, "This is another appended text: PRINT #"
CLOSE 2
END
```

Editing the content of a Sequential file

It is possible that you make changes to a sequential file although it is pretty involved. You can use the Algorithm below to effect editing on a sequential file:

```
OPEN OriginalFile (For INPUT)
  OPEN TempFile (For OUTPUT)
      INPUT Field from OriginalFile
       DO UNTIL EOF (OriginalFile Number)
           Check Condition
             True{WRITE Changes to the TempFile }
           False{WRITE Field to the TempFile}
             GET NEXT field value from OriginalFile
        Continue Loop
       CLOSE OriginalFile
        KILL OriginalFile
        ReName TempFile AS OriginalFile
        CLOSE {TempFile Number}
STOP {Algorithm}
```

Fig7.1a Algorithm – Editing Sequential file

SideTalk
To rename a file
NAME "C:\MISC\Test.bas" AS "C:\MISC\New.bas"
(Note the path must be on the same drive but not necessarily on the same directory)

§7.1.1.2 Random Access file

Unlike in a sequential file, a Random access file allows you to locate a record say 20th position without necessarily going through the 19th record. This saves a considerable amount of time.

The command for setting up a Random-Access file is analogous to that of a sequential file e.g.

OPEN “C:\MISC\Musicals.RND” AS #3 LEN=100

The LEN=100 means that each record can hold 100 characters. There is no need to include FOR OUTPUT or APPEND as you did in sequential file. In Random-Access file you can read and write simultaneous. The underlying files in your OS *config.sys* file set the only restriction.

As in sequential files you use the CLOSE command. Here you use the GET and PUT commands to read and write to an *.rnd* file.

§7.1.1.3 Binary files

Binary files are like random-access files in that the open statement allows you to read and write simultaneously to the file, unlike the other types, binary files techniques allow you to manipulate any kind of file (i.e. not just text files). The technique allows you to edit any byte of a file.

The command for setting up binary file is:

OPEN *FileName* FOR BINARY AS #*FileNumber*

You use INPUT$() function to read from such a file. I believe you might have tried to open a file and saw horrible looking characters, if not try the following (BE CAREFUL NOT TO MAKE ANY CHANGES OTHERWISE YOU WILL CORRUPT THE FILE!).

(i) Open a new window in QBASIC

(ii) Write a simple BASIC code to print the sum of all integers from 0 to 100, then the square root of all numbers from 201 to 215 in steps of 0.1

(iii) Save the file (As QBASIC *fast load i.e. not as file readable by other program)* as “c:\test.bas”

(iv) Close the file

(v) On the DOS prompt enter EDIT C:\Test.bas or open the file C:\Test.bas with Notepad on windows

(vi) If you do not have QBASIC on your system try to open a picture file with Notepad to see them (horrible looking “creatures”)

You will see what I mean by horrible characters. Opening a bitmap file with a text editor gives similar action.

With Binary mode, you can open and print the contents of any file, regardless of any *embedded control characters*:

Do the following, draw a circle in Bitmap (Microsoft Paint) and save as C:\MSIC\Circle.bmp close the file and run the following lines of code:

```
OPEN “C:\MSIC\Circle.bmp” FOR BINARY AS #4
DIM i%, C$
FOR i=1 TO LOF (4)
    C$=INPUT$(1, 4): REM Single Character
     PRINT C$
```

```
NEXT
CLOSE 4
END
```

Handling Binary files is beyond this kind of introductory text.

§7.1.2 Storage Devices and Access types

The following are the storage devices that are commonly used:

- Hard disk
- Floppy disk (3 ½ʹ or 5 ¼ʹ)
- Compact Disk (CD)
- Zip disk etc

While you can Read or Write to your hard disk and floppy you cannot write to a CD-Rom (ROM means Read-Only memory). If your floppy disk is *write-protected*, you can only read but cannot write; although you can uncover the write-protect notch to give you a write-access.

You can make a file have an access you want by using the following:

OPEN path [FOR MODE] [ACCESS *access*] AS [#][*Number*][LEN=*RecordLength*]

Where:

1. path = Path or file name
2. MODE = Any of APPEND, BINARY, INPUT, OUTPUT or RANDOM
3. access = READ, WRITE or READ WRITE

   e.g. OPEN “Test.txt” FOR BINARY ACCESS READ AS #2

   (Allows reading but no changing)

4. LOCK= SHARED (e.g. on a network), LOCK READ, LOCK WRITE, LOCK READ WRITE

   Lock controls what other process can do on the same file:

   e.g.

   OPEN *filename* FOR BINARY ACCESS READ LOCK READ AS #1
5. Number = any *free* number 1, 2 ,3 …511
6. RecordLength: an integer from 1 to 32,767, in Random Access it implies Record length; in Sequential file it implies number of characters buffered by the OS.

§7.1.3 Editing Data (Edit Command)

As you code big programs, you later deal with huge size of data. I once worked on a data from a device that measures atmospheric data every 3 minutes interval. The program requires finding average and other Physical values fields from 1998 to 2000! It means finding the average of 3 minutes values within February 28th 1998 to November 2000!

You cannot open such Data on your QBASIC IDE. You can view such data from the EDIT window. From your IDE select DOS Shell and on the DOS prompt ENTER EDIT *filename* to view the content of such a large file.

§7.2 FILE STATEMENTS

The generalized OPEN statement without the Access and Locking is:

OPEN *pathname* [FOR MODE] AS #*Number*

The modes are OUTPUT, INPUT or APPEND and RANDOM or BINARY just discussed.

<u>OUTPUT</u>

This creates a new file (sequential). If the path name already exists, it deletes the content and makes it ready for *output*. It should however be noted that you cannot open a read only disk for output (see 7.1.2)

<u>APPEND</u>

It allows you to add extra record to an existing file, if the file does not exist, before, it creates a new file. It is not allowed to append to a read-only file.

§7.2.1 GET & PUT

These file statements work with Random Access file to Read or Write to a particular position. The example following (LST7.2a) illustrates well their use (in file processing – note that they exist in graphics statements)

LST7.2a GET & PUT-File

```
CLS
OPEN “C:\MISC\MUSICALS.RND” AS #1 LEN = 95
REM Assuming the file Musicals.Rnd
    ‘Contains Record in the following format
    ‘ Artist, Title, Country
TYPE Music
       Artist AS STRING*25
       Title AS STRING*50
        Country AS STRING*20
END TYPE
REM LEN=25+50+20
DIM MusicDesc AS Music
GET 1,12, MusicDesc
PRINT “ The Artist in the 12th record has :”
PRINT “NAME: = ”; MusicDesc.Artist
PRINT “Title of Song = ”; MusicDesc.Title
PRINT “Country where produced =”; MusicDesc.Country
PRINT “CHANGED THE THIRD RECORD”
INPUT “ Name  ”; MusicDesc.Artist
INPUT “ Song title  ”; MusicDesc.Title
INPUT “ Country  ”; MusicDesc.Country
PUT 1,3, MusicDesc
END
```

The format is:

GET *FileNumber, RecordPosition, Record*

PUT *FileNumber, RecordPosition, Record*

§7.2.2 CLOSE

To close a file, opened using the OPEN statement, use:

CLOSE *[FileNumber][,FileNumber2][…...]*

e.g. CLOSE 1, 2, 3

It must be preceded by an open statement. After closing a file number same number could be used to open same or another file. Note that if after closing the code references an open statement calling an already close file number, it amounts to a run time error. However if the file numbers are ignored, all the open files are closed.

§7.2.3 KILL

The KILL statement is used to delete a file. You can only delete a file that is not on read only disk. You cannot delete from a CD-Rom say.

The KILL statement is: KILL *filename*

e.g. KILL “C:\MISC\Test.dat”

Running the KILL statement after deleting the file flags an error : FILE NOT FOUND – A trappable error.

§7.3 DATABASE

A database is a large store of data held in a computer and easily accessible to person using it. There are special software designed solely for database programming, such programs include but not limited to DBASE; Microsoft ACCESS, FOXPRO, ORACLE etc.

You can although design a database manager with your random file and Binary files using ingenuity but it may not be worth the task when you can easily learn any serious database software.

§7.4 SUMMARY

You have been taken through the rudiments of file processing in this chapter, we have discussed the following types of files: Sequential files, Random files, and a little introduction to Binary files.

Also, you learnt about storage devices and access to files as well as how to use editor to open data files.

Lastly you learnt the files statements and got an insight to database programming.

§7.5 QUIZ

GET and PUT are similar to what and what?

§7.6 PROJECT

(a) Develop a Database manager for a Record Studio to assist in locating shelve and audio cassette. The customer specifies either the Artist, Song or Volume etc

Hint:

Create files for the data, Use Random file and DO-WHILE

(b) Write a code to mimic the search and change of your QBASIC IDE.

Bibliography

- Rob Thayer (1998), **Visual Basic 6 Unleashed**, Sams Publishing, United States of America

CHAPTER EIGHT

CHAPTER EIGHT

§8.1 GRAPHICS WITH QBASIC

Computer graphics has gone very far. There are hosts of software application for graphics; they include CorelDraw; Microsoft Paint; Macromedia Flash MX™ etc. QBASIC graphics still needs to be discussed, despite various advances in computer graphics, for appreciation purpose at least.

§8.1.1 Screen Modes

The output screens have numbers designated to them. The modes determine the "kind" of output: line types, text size, and resolution to mention some. The syntax is:

[*LineNumber*] SCREEN [*Mode*][,[*ColorSwitch*]][,[*apage*]][,[*vpage*]]

Where the optional variables mean:

Mode

This is an integer value or a constant that represents Screen mode. The various valid modes are discussed immediately following this section.

ColorSwitch

This is a numeric value ranging from 0 to 255 that determines whether color is displayed on composite monitors.

Note: When it is nonzero, color is disabled and only black and white images are displayed but if ColorSwitch is zero, images are in color. However, the meaning of the ColorSwitch argument is inverted in screen mode 0 while the ColorSwitch is ignored in screen mode 2.

apage

apage is a numeric expression that is the number of the screen page that text output or graphics commands write to.

See documentation on your QBASIC installation disk.

vpage

This is also a numeric expression that is the number of the screen page being displayed.

More on modes

SCREEN 0: Text mode only

SCREEN 1: 320 x 200 graphics

SCREEN 2: 640 x 200 graphics

SCREEN 3: Hercules adapter required, monochrome monitor only

SCREEN 7: 320 x 200 graphics

SCREEN 8: 640 x 200 graphics

SCREEN 9: 640 x 350 graphics

SCREEN 10: 640 x 350 graphics, monochrome monitor only

Screen 11: 640 x 480 graphics

Screen 13: 320 x 200 graphics; 40 x 25 text format

I strongly recommend that you look up the documentation on your QBASIC IDE for efficient use of the screen modes.

§8.1.2 WINDOW statement

The WINDOW statement is used to give physical boundary scale to your screen (i.e. monitor) so that you can customize coordinates on your screen. The screen of Fig8a shows the implication of using the statement WINDOW (-1, -1) – (1, 1)

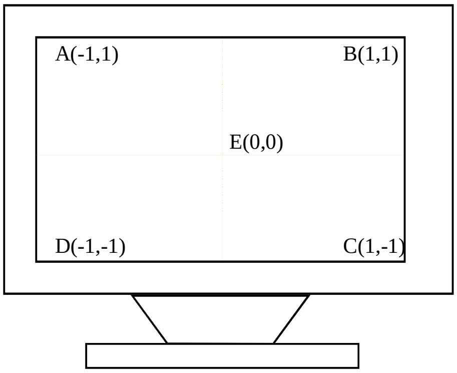


Fig8a: Screen Positions

The statement is similar to WINDOW (-1, 1)-(1,-1) where the points in brackets are the coordinates of the boundaries.

WINDOW (X1, Y1) – (X2, Y2)

Where your choice of the coordinates depends on what you intend to draw.

§8.1.3 COLOR statement & Pixel

Color statement is used to specify the text and background (fore) colours:

COLOR *TextColourNumber, ForeColourNumber*

Where *TextColourNumber* is any integer ranging from 0 to 15. The numbers are coded e.g. 4 represent Red and 15 bright white. *ForeColourNumber* is similarly coded. If the numbers are the same, you cannot see printed text. Note that not all screen modes support the statement. Try to find out. The table below shows the numbers and their colors equivalent:

0 Black	5 Magenta	10 Light green
1 Blue	6 Brown	11 Light Cyan
2 Green	7 White	12 Light red
3 Cyan	8 Gray	13 Light magenta
4 Red	9 Light Blue	14 Yellow
15 High-intensity white	XXXXXXXX	XXXXXXXX

Pixel: This is a picture element; it is the smallest unit of resolution on your monitor. To turn-on a pixel, you can use the PSET statement as used below:

PSET (Column, Row) [, *ColorCode*]

The *ColorCode*, as indicated by the square bracket, is optional.

You can plot graphs with the PSET statement. The program listing of LST8.1a uses the PSET statement to plot three different graphs.

LST8.1a Using PSET

```
10 CLS: WINDOW (-1, 1) – (1, -1): DIM Reply
15 CONST PI=3.14159
20 PRINT "SELECT A GRAPH"
30 PRINT "1 – Sine Graph"
40 PRINT "2 – Quadratic Graph"
50 PRINT "3 – Four – Leaf Clover"
60 PRINT "0 – Quit"
70 INPUT " ", Reply
80 SELECT CASE Reply
90      CASE 3
```

SideTalk

Try to change the STEPs to see the dramatic change in output try 0.0001, 0.1 and 0.5 or even leaving them out i.e. STEP 1

```
100        FOR I=0 TO 2*PI STEP 0.01
110                Radius = COS (2*I)
120                X = Radius *COS (I)
130                Y = Radius * SIN (I)
140                PSET (X,Y),  15
150          NEXT
160     CASE 2
170        FOR X = -1 TO 1   STEP 0.01
180             Y = X^2+2
190             PSET (X, Y), 2
200           NEXT
210     CASE 1
220        RADIUS = 1
230            FOR I = 0 TO 2 *PI   STEP  0.01
240                 Y= Radius * SIN (I)
250                  REM Y=RADIUS*COS (I)
260                   PSET (I, Y), 4
270              NEXT
280     CASE 0
290            GOTO LastLine
300      END SELECT
310     PRINT "PRESS ANY KEY TO CONTINUE"
320     DO: LOOP UNTIL INKEY$<> ""
330     CLS :        340    GOTO 20
LastLine:
350 END
```

§8.1.4 LINE statement

This statement is used to draw a line on the Screen. The syntax is:

LINE [[*step*][(x1,y1)]-[*step*](x2,y2)[,[*ColorCode*][B[F]][*Style*]

Where (x1,y1) and (x2,y2) specify coordinates.

(x1,y1) : Beginning coordinate

(x2,y2): End coordinate

Step: if used enables you to specify relative screen coordinate i.e. from a location to another with the step value given.

Color : See §8.1.3

BF: Option draws a filled box (**b**ox **f**ill)

B: Draws a box with diagonal of the specified coordinates (x1,y1)-(x2,y2)

The program listing of LST8.1b shows some simple use of line statement.

LST8.1b Lines

```
10 CLS
20 SCREEN 2 : WINDOW (-10,10) – (10,10)
40 LINE (0,0) – (3,3) : REM Line A
50 LINE – (3,4)          : REM Line B
60 LINE – STEP (1,1) : REM Line C
70 LINE STEP (1,2) – STEP(1,1) : REM Line D
80 LINE – (0,0) : REM Line E
90 PRINT “Press Key” : DO : LOOP UNTIL INKEY$<> “”
100 LINE (0,0) – (10,10),,,&HFF00: REM Dashed Line
120 END
```

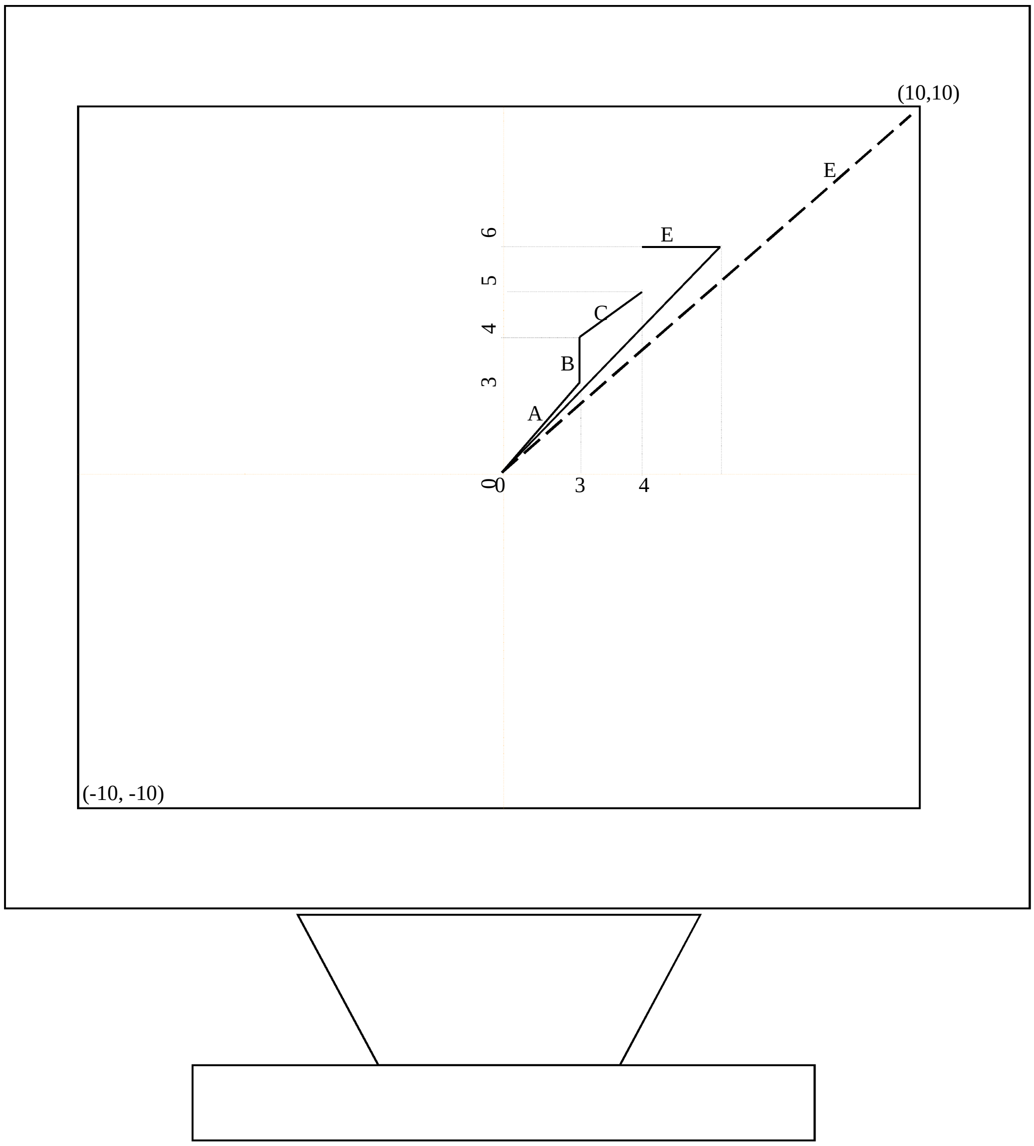


Fig8.1 Expected Output of LST8.1b

[Note that the labels are not expected-it was put to give you light]

§ 8.1.5 CIRCLE statement

This statement is used to draw Circles, Arcs, Ellipses and Sectors. Unlike in LINE statement where BF can be used you use PAINT to put (fill) colour. The Circle syntax is:

CIRCLE [STEP](X, Y), Radius [, *Color*][,[*Start*][,[*End*][,*Aspect*]]]

The optional *Start* and *End* are used to draw Arc. While STEP is used to specify a relative position if used, X, Y specifies the center of the circle.

Aspect, also known as aspect ratio, is the ratio of the Y-Radius to the X-Radius. The default value for *aspect* is the value required to draw a round circle in the Screen mode (§8.1.1). You calculate *aspect* ratio using the relation:

4*(YPixels/XPixels)/3

Screen resolution is defined by XPixel × YPixel for example the resolution of Screen 1 is 320 × 200, therefore the *aspect* ratio is = 4*(200/320) /3 = 5/6

If *aspect* ratio is less than 1, the radius is X-Radius, it is Y-Radius if *aspect* ratio is greater than 1.

You can write programs and put Circle statement in a loop etc.

§ 8.2 SIMPLE ANIMATION

Animation is the technique of making people or animals in pictures appear to move. It involves giving “observers” the impression that an image is moving or performing some actions. This technique is employed in cartoons. Some animated pictures come with windows. I once animated pictures with Coffee Animator™ software, where I used seven different images to give an impression that an electric motor is "working".

With QBASIC, like in many programming languages, you can create animations depending on your level of intuitiveness. To really appreciate the simple animations of LST8.1c you have to run the code in QBASIC environment.

LST8.1c SIMPLE ANIMATION CODE

```
DIM Tennis (90)
DIM VirtualPad (90):'These arrays hold enough memory
DIM I, AUTHUR$
AUTHUR$ = "ACUBESOFT OF NIGERIA" + SPACE$(30)
T = 10 :' 10 Seconds
DO
T = T - 1
COLOR INT(1 + RND * 14) :' Random colours
LOCATE 13, 35: PRINT T: REM Display count down
SLEEP 1
LOOP UNTIL T = 0
SCREEN 2
CLS
CIRCLE (6, 6), 5       'Draw and paint Tennis.
PAINT (6, 6), 1
GET (0, 0)-(14, 14), Tennis
CLS
```

```
LINE (1, 1)-(10, 10), , BF ‘ Box Fill
GET (0, 0)-(14, 14), VirtualPad
DIM Horiz, Vert, u
Horiz = 10: Vert = 10
H0 = .91: V0 = .91
LINE (0, 0)-(550, 160), , B
LINE (10, 10)-(500, 150), , B
LOCATE 22: PRINT "NAUGHTY CODE!"
DO
   Horiz = Horiz + H0
   Vert = Vert + V0
   IF INKEY$ <> "" THEN END  ' Test for key press.
  'Change direction if Tennis hits left or right edge.
   IF (Horiz < 10.1 OR Horiz > 480) THEN
      H0 = -H0
     PUT (Horiz, Vert), VirtualPad, PRESET
     LOCATE 20, 5: PRINT "Caught"; TIME$
```

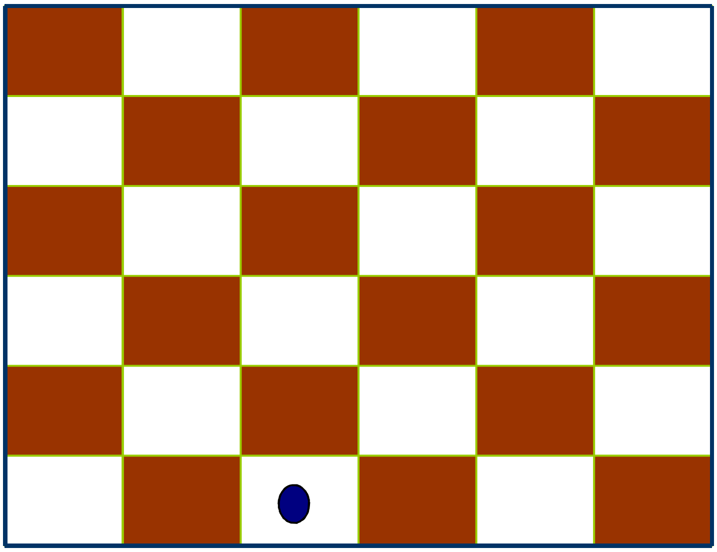
```
SLEEP 1
  END IF
  LOCATE 20, 5: PRINT "      "; TIME$
  IF I = LEN(AUTHUR$) THEN
  LOCATE 22, 20: PRINT "                            "
  I = 1
  ELSE
  LOCATE 22, 20: PRINT MID$(AUTHUR$, 1, I)
  I = I + 1
  END IF
  IF (Vert < 10 OR Vert > 140) THEN
    V0 = -V0
  END IF
  FOR u = 1 TO 3000: NEXT
  PUT (Horiz, Vert), Tennis, PSET
LOOP
END
```

§8.3 Summary

With QBASIC, you can generate shapes with the graphics statements as well as perform some simple graphical animations depending on your skills and ingenuity although there is a limit to what you can do with graphics in QBASIC.

§8.4 Project

Design the draft board shown and allow a user to move seed using the arrow keys.

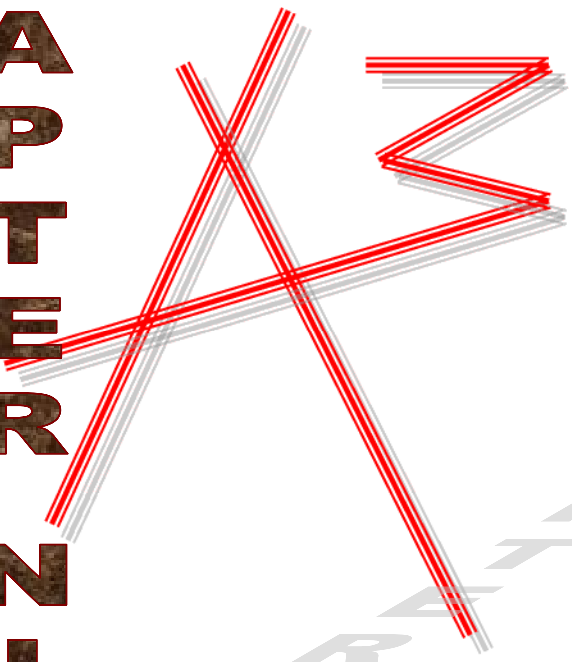


Hint: Use two FOR-NEXT loops; LINE (BF) then adapt the code of LST8.1c i.e. GET and PUT the seed

Bibliography

- Microsoft QuickBASIC online Help.

CHAPTER NINE



CHAPTER NINE

§9.1 MISCELLANEOUS TOPICS

You have been through a great deal of programming concepts with QBASIC, however, you have not arrived yet. QBASIC has more powerful things it can perform. You have to see more advanced textbooks for more. I can guarantee you that having gone through and understood this book, you can now be sure that you can proceed to *any* programming text with little or no special assistance and get more (even on other languages like C++, FORTRAN77 and so on)

This ending chapter will help you see some programs which will assist you somehow.

§9.1.1 TIMER FUNCTION

To write a code that will operate on time basis rather than user input, the Timer Function may come helpful. See LST9.1a for an example.

```
LST9.1a Timer
10 CLS: C=0:
 LOCATE 18
PRINT "STOP WATCH"
PRINT "**********************"
PRINT "*                          *"
PRINT "*                          *"
PRINT "***********************"
70 ON TIMER (1) GOSUB TPRINTER
REM Call Timer Subroutine
LOCATE 50: PRINT "R – Refresh"
LOCATE 51: PRINT "C – Continue"
LOCATE 52: PRINT "S – Stop"
LOCATE 54: PRINT "P – Pause"
DIM KEY1$
DO: KEY1$ = INKEY$
SELECT CASE UCASE$(KEY1$)
              CASE "P"
                          TIMER STOP
                CASE "C"
                          TIMER ON
                   CASE "S"
                           TIMER OFF
                     CASE "R"
                            C=0
        END SELECT
END
```

```
TPRINTER:
LOCATE 20
  SELECT CASE C
       CASE 0 TO 60
             PRINT C;
             PRINT    "SECONDS"
           CASE ELSE
                      C=0
  END SELECT
            C=C+1
RETURN
```

You should find it easy to modify LST9.1a to print minutes and hours elapsed.

§9.1.2 Wildcard Searching

If you have searched for files on Windows or DOS prompt you will be familiar with wildcards. They are also found in most database applications where records can be found.

A typical DOS command to search all files with extension .bas can be gotten as follows:

C:\DIR A*.bas /p/s ↵ will return files like the following:

ADE.BAS

ALABI.BAS

ALGOLS.BAS

APPENDIX.BAS etc

The wildcard is “*”

C:\ DIR *.D*/S /P ↵ will return all files with the extension starting with D. the following will be returned (say):

C:\WINDOWS\mypictures\file2.dat

C:\WINDOWS\My Documents\Registry.doc etc.

The program listing of LST9.1b shows a simple programming for wildcard searching, assuming there is file C:\MISC\friends.txt containing the names of your friends – say up to500 first names in a Notepad.

The program helps you search for names matching your search criteria using wildcards.

LST9.1b Wildcard Searching

```
DECLARE SUB DoAsterikSearch (Query AS STRING)
DECLARE SUB DoParticularSearch (Query AS STRING)
DECLARE SUB DoQuestionSearch (Query AS STRING)
DECLARE SUB Scroll (Text AS STRING)
DECLARE FUNCTION Parse$ (Query AS STRING, Delimiter AS STRING)
DECLARE FUNCTION DoSearch! (Query AS STRING)
DECLARE SUB Welcome ()
DECLARE SUB Help ()
ON ERROR GOTO LastLine

'++++++++++++++++++++++++++++++++++++++++++++++

CONST ProgramName = "WXp Search -04"

DIM SHARED Hit

DIM Query AS STRING

CONST Asterik = "*" : CONST Question = "?" : CONST None = "None"

'+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

TYPE Delimiter
  Position AS INTEGER
  Valid AS INTEGER
END TYPE
DIM SHARED D AS Delimiter
'++++++++++++++++++++++++++++++++++++++++
CONST True = -1
CONST False = 0: Hit = &H12
'++++++++++++++++++++++++++++++++++++++++++++++++
Welcome
DIM FunctKey$
FunctKey$ = "|"
KEY 1, FunctKey$ + CHR$(13)
'+++++++++++++++++++++++++++++++++++++++++++++++++
GetSearch:
INPUT Query
IF Query = FunctKey$ THEN
  Help
  GOTO GetSearch
ELSEIF UCASE$(Query) = "/Q" THEN GOTO 100
ELSEIF Query = "" THEN GOTO EnterPressed
ELSE
  IF DoSearch(Query) = True THEN Hit = True ELSE Hit = False
  Welcome
  GOTO GetSearch
END IF
CLOSE 1
100  PRINT : PRINT : PRINT
Scroll ProgramName + " TERMINATED...THANK YOU "
COLOR 14
Scroll "  ACUBESOFT OF NIGERIA " + CHR$(13) + CHR$(9) + DATE$
COLOR 0
END
```

```
LastLine:     PRINT : PRINT : PRINT : PRINT
COLOR 4: Scroll ProgramName + " Execution Failure!": COLOR 15
RESUME 100
EnterPressed:
Welcome
Scroll "PRESSING ENTER, WITHOUT QUERY WILL NOT RETURN MEANINGFUL RESULTS "
GOTO GetSearch
SUB DoAsterikSearch (Query AS STRING)
DIM LeftPart AS STRING, RightPart AS STRING
IF Query = Asterik THEN
  Scroll "Are you sure you want to display all Records ? Y/N"
  IF LCASE$(INPUT$(1)) <> "y" THEN 7000
 END IF
LeftPart = MID$(Query, 1, D.Position - 1)
RightPart = MID$(Query, D.Position + 1, LEN(Query))
DIM FileContent AS STRING
DIM HitsCount
HitsCount = 0
PRINT : PRINT
DIM L AS LONG
L = 21
WHILE NOT EOF(1)
   INPUT #1, FileContent
REM The following italicized lines should be a single Line
                              IF UCASE$(MID$(FileContent, 1, LEN(LeftPart))) =
             UCASE$(LeftPart) AND  UCASE$(RIGHT$(FileContent,
             LEN(RightPart))) = UCASE$(RightPart) THEN
     HitsCount = HitsCount + 1
```

```
    IF HitsCount = L THEN  :' Display Results in 20 Per Page
     L = HitsCount + 20
     COLOR 6
     Scroll "          Press Spacebar to Continue"
     COLOR 15
     DO: LOOP UNTIL INKEY$ <> ""
    END IF
    PRINT USING "####  &"; HitsCount; FileContent
    REM The & (Ampersand) format means Print Entire string
   END IF
WEND
IF HitsCount > 0 THEN
  COLOR 13
  Scroll "    We found" + STR$(HitsCount) + " matching Record(s)"
  COLOR 15
  Hit = True
ELSE
 PRINT : PRINT
 COLOR 4
 Scroll " We found no Matching record try again"
 COLOR 15
Hit = False
END IF
Scroll "Press any key"
DO: LOOP UNTIL INKEY$ <> ""
7000 CLOSE 1
END SUB
```

```
SUB DoParticularSearch (Query AS STRING)
DIM FileContent AS STRING: DIM HitsCount
HitsCount = 0
PRINT : PRINT
WHILE NOT EOF(1)
   INPUT #1, FileContent
   IF UCASE$(FileContent) = UCASE$(Query) THEN
     HitsCount = HitsCount + 1
     PRINT USING "####  &"; HitsCount; FileContent
     REM The & (Ampersand) format means Print Entire string
   END IF
WEND
IF HitsCount > 0 THEN
  COLOR 13
  Scroll "    We found" + STR$(HitsCount) + " matching Record(s)"
  COLOR 15
  Hit = True
ELSE
 PRINT : PRINT
 COLOR 4
 Scroll " We found no Matching record try again"
 COLOR 15
Hit = False
END IF
Scroll "Press any key"
DO: LOOP UNTIL INKEY$ <> ""
 CLOSE 1
END SUB
```

```
FOR i = 1 TO LEN(Query)
  SELECT CASE MID$(Query, i, 1)
    CASE Delimiter
      P = Delimiter
    CASE ELSE
      P = None
  END SELECT
    IF P = Asterik THEN
      D.Position = i
      EXIT FOR
    END IF
    IF P = Question THEN
      D.Position = i
      EXIT FOR
    END IF
        D.Position = -1
NEXT
Parse = P
END FUNCTION
SUB Scroll (Text AS STRING)
DIM L, i, Delay: L = LEN(Text)
FOR i = 1 TO L
  PRINT MID$(Text, i, 1);
   FOR Delay = 1 TO 10000: NEXT
NEXT
PRINT
END SUB
```

```
SUB Welcome
CLS
COLOR 15
PRINT SPACE$(30); ProgramName
PRINT SPACE$(20); "++++++++++++++++++++++++++++++++++"
PRINT SPACE$(20); "+++++ ENTER YOUR SEARCH ... +++++"
PRINT SPACE$(20); "    PRESS ";
COLOR 3: PRINT "F1"; : COLOR 15:
PRINT " TO SHOW " + ProgramName + " HELP "
PRINT SPACE$(30); "Enter "; : COLOR 12: PRINT "/Q";
COLOR 15: PRINT " To Quit"
COLOR 12
IF Hit = False THEN
  Scroll ProgramName + " DID NOT HIT ANY TARGET!  ENTER ANOTHER"
END IF
IF Hit = True THEN
  PRINT SPACE$(40)
END IF
COLOR 15
END SUB
```

§ 9.1.3 Miscellaneous Programs

The following sets of codes are intended by the author to give some insights.

```
LST9.1c PROGRAM FIBONACCI
DECLARE FUNCTION IsPrime! (X!)
'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
'XXXXXXXX ACUBESOFT OF NIGERIA XXXXXXXXXXXXXXX
'XXXXXXXXXXXXXXXXXXXXXXXXXXXX27 - 10 - 2004 XX
CLS
DIM F1, F2, X, I, J, Q1
CONST True = -1
CONST False = 0
DIM Status AS STRING
F1 = 1
F2 = 1
N = 20
Status = " is a Prime number"
COLOR 14
PRINT "   FIBONACCI SERIES & PRIME NUMBERS"
COLOR 3
FOR I = 1 TO 35: PRINT CHR$(21); : NEXT: PRINT
COLOR 15
PRINT USING "######## &"; F1; Status
PRINT USING "######## &"; F2; Status
```

```
FOR J = 3 TO N + 1
  X = F1 + F2
   REM A NUMBER IS PRIME
   ' IF :
   IF IsPrime(X) THEN ' See the nomenclature: Friendly ?
     Status = " is a Prime number"
   ELSE
    Status = " is not a Prime number"
    END IF
    F1 = F2
    F2 = X
   PRINT USING "######## &"; X; Status
NEXT J
END
FUNCTION IsPrime (X)
   FOR I = 2 TO INT(SQR(X))
       Q = X / I
      Q1 = INT(Q)
     IF Q = Q1 THEN 200
   NEXT
Prime = True
GOTO 1000: EXIT FUNCTION
200 Prime = False
GOTO 1000: EXIT FUNCTION
1000: IsPrime = Prime
END FUNCTION
```

LST9.1d SIMPLE RECURSIVE ROUTINE

```
DECLARE FUNCTION FACT& (Num!)

'---------------------------------------------

'------ACUBESOFT OF NIGERIA----

'----------------(C) 2004------------------

10 CLS
PRINT "FACTORIAL CALCULATOR"
200 ON ERROR GOTO 1000
INPUT N
PRINT FACT(N)
PRINT "Another Factorial Y/N ?"
  IF UCASE$(INPUT$(1)) = "Y" THEN 200
PRINT "Thank you good bye"
END
1000
PRINT "This code could note handle this value"
RESUME 10
FUNCTION FACT& (Num) STATIC
IF Num <= 1 THEN
  F = 1
ELSE
 REM Recursion – Calling itself in it self!
  F = Num * FACT(Num - 1)
END IF
FACT = F
END FUNCTION
```

LST9.1e SPECIAL NUMBERS

```
' - - - - - - - - - - - - - - --ACUBESOFT OF NIGERIA  - - - - - - - - -
' - - - - - - - - - - - - - - - - - -(C) 2ØØ4  - - - - - - - - - - - - - - - - - - -
CLS
DIM n, e, x, y, z, r
FOR n = 1 TO 5Ø
FOR e = Ø TO 9
FOR x = Ø TO 9
FOR y = Ø TO 9
FOR z = Ø TO 9
FOR r = Ø TO 9
   IF 1ØØØØ * e + 1ØØØ * x + 1ØØ * y + 1Ø * z + r > 1 THEN
   IF 1ØØØØ * e + 1ØØØ * x + 1ØØ * y + 1Ø * z + r = e ^ n ↵
             + x ^ n  + y ^ n + z ^ n + r ^ n THEN
 PRINT (1ØØØØ * e + 1ØØØ * x + 1ØØ * y + 1Ø * z + r); ↵
       "=>  "; e; "^"; n; " + "; x; "^"; n; " + "; y; "^"; n; " + "; z; "^"; n; " +
"; r; "^"; n
END IF
END IF
NEXT: NEXT: NEXT: NEXT: NEXT: NEXT
END
```

'Generated set output within 0 – 99999

$8208 = 8^4+2^4+0^4+8^4$

$4150 = 4^5+1^5+5^5+0^5$

$4151 = 4^5+1^5+5^5+1^5$

$54748 = 5^5+4^5+7^5+4^5+8^5$

$92727 = 9^5+2^5+7^5+2^5+7^5$

$93084 = 9^5+3^5+0^5+8^5+4^5$

LST9.1f Binary Numbers

```
CLS
DIM Value AS INTEGER
INPUT “Enter an integer”; Value
PRINT Code$(Value)
END
FUNCTION Code$ (Value)
REM Convert Integer Value to Binary Number
DIM  Token AS STRING, c$
DIM R
11 R = Value MOD 2
 Value = Value \ 2
‘Integer Division e.g. 3\2 = 1:: 3/2 = 1.5::CINT (3/2) =2::INT (3/2) =1
 c$ = c$ + STR$(R)
IF Value <= 1 THEN 22 ELSE 11
22 c$ = c$ + STR$(Value)
FOR j = LEN(c$) TO 1 STEP -1
 'The Ltrim$ and Rtrim$ are to ensure close packing by trimming
 Token = Token + LTRIM$(RTRIM$(MID$(c$, j, 1)))
NEXT
IF VAL(Token) = 1 THEN Token = "1"
Code$ = Token
END FUNCTION
```

LST9.1g Time Interval

```
CLS
REM This function returns the number of
'Seconds Difference between two times
PRINT "Time 1 = 13:23:10"
PRINT "Time 2 = 21:13:05"
PRINT "Time difference ="; Seconds ("21:13:05") - Seconds ("13:23:10")
END
FUNCTION Seconds (T$)
REM This Function assumes Time of format 24:00:00
DIM Sec, Min, Hr
Sec = VAL(MID$(T$, 7, 8))
Min = VAL(MID$(T$, 4, 5))
Hr = VAL(MID$(T$, 1, 2))
Seconds = Sec + Min * 60 + Hr * 60 * 60
END FUNCTION
```

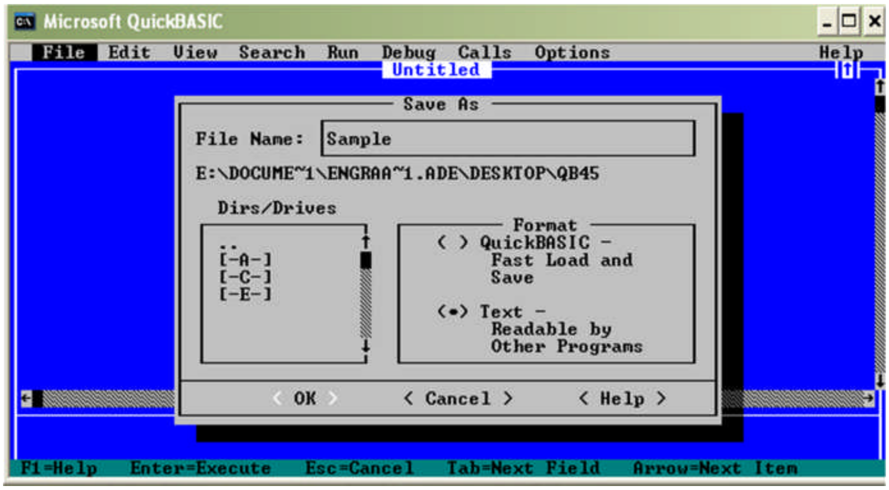
LST9.1h Normal Distribution Curves

```
DECLARE FUNCTION NormalDist! (x!, Sigma!, Miu!)
DECLARE SUB PutAxes ()
DIM Clr, Mean
10 CLS:   Clr = 3
  CONST PI = 3.141592653#
SCREEN 7
WINDOW (-5, -.5)-(5, 1.5)
  PRINT "Enter Parameters: Mean=0 is Symmetric"
  INPUT "Mean ="; Mean
PutAxes
 FOR Sigma = .25 TO 1 STEP .25
   LOCATE 12: PRINT USING "Sigma =#.##"; Sigma
   Clr = Clr + 1
    FOR x = -2 TO 2 STEP .001
      PSET (x, NormalDist(x, Sigma, Mean)), Clr
    NEXT
    SLEEP 1
 NEXT
LOCATE 21: PRINT "Normal Distribution “;
     PRINT “Curves"
  PRINT "Press a to do another"
  LOCATE 12: PRINT "                    "
   IF LCASE$(INPUT$(1)) = "a" THEN 10
    REM Modify the Axes to put Labels
END
```

```
FUNCTION NormalDist (x, Sigma, Miu)
DIM Den, Part1
IF Sigma>0 THEN
Den=(Sigma * SQR(2 * PI))
Part1=((x - Miu) / Sigma) ^ 2
NormalDist = EXP(-.5 *Part1) / Den
END IF
END FUNCTION
SUB PutAxes
LINE (-5, 0)-(5, 0)   'X- Axis
LINE (0, 5)-(0, 0), 3  'Y- Axis
END SUB
```

§ 9.1.4 Printing Your Code

You may want to print your code on a printer, which may not necessarily be a line printer; it is my way to print my codes using Windows text editors like MS Word, WordPad and the like. To accomplish this you save the QBASIC program as <*Text- Readable by other programs*> in the save as dialog box of your QBASIC IDE as shown in the figure below (Fig 9.1)



```
File Name: Sample
E:\DOCUME~1\ENGRAA~1.ADE\DESKTOP\QB45
Dirs/Drives
..
[-A-]
[-C-]
[-E-]
Format
( ) QuickBASIC - Fast Load and Save
(•) Text - Readable by Other Programs
F1=Help   Enter=Execute   Esc=Cancel   Tab=Next Field   Arrow=Next Item
```

Fig 9.1 Save As Dialog Box

Saving the program (Source Code) file in this format allows you to be able to *read* the content of the file with text editors. If you save your code as E:\GSeidel.bas you will have to use the *open dialog* of the editor to locate or just type the path. If open is chosen from Word, the default *extension* is *.doc not *.bas as you used to have it in QBASIC IDE, so use (*All Files*) – *.* or specifically use *.bas in the *file Name* request of the *Open Dialog*. You can then edit, print or mail your code to your lecturer or do otherwise. See Fig9.1b – Open Dialog

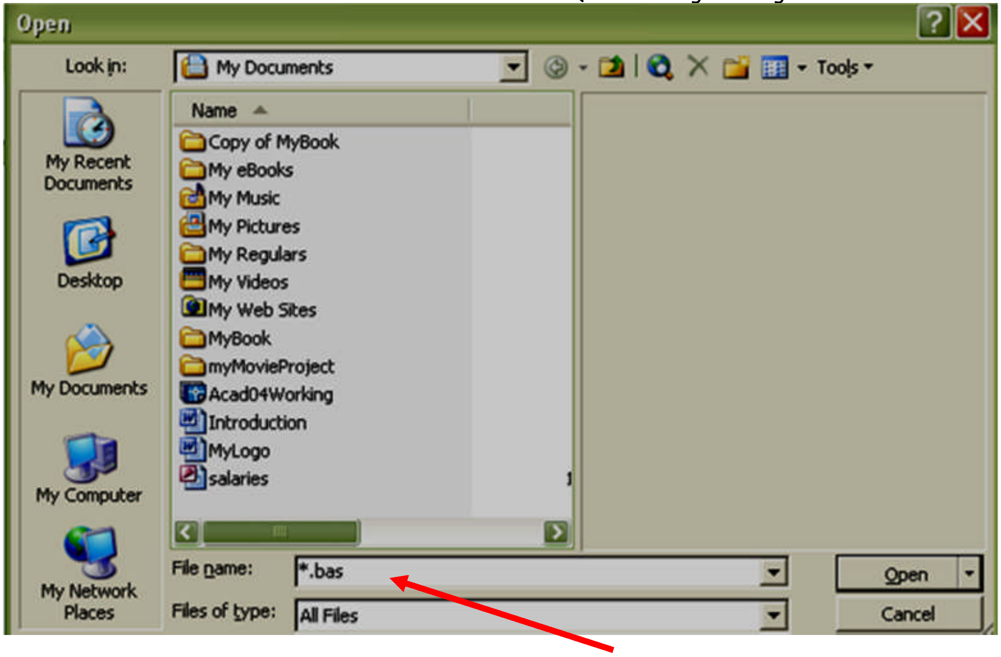


Fig9.1b Open Dialog Box

§ 9.1.5 Printing Output

Printing output looks like I am repeating a section, No! You, at times want to have a paper copy of your code and may not have a line printer to use the LPRINT command – see index. I will suggest that you use either of the following to get your output to a text Editor:

(i) Use PRINT # or WRITE # discussed in Chapter Seven, then read the content using the discussions of § 9.1.4 or

(ii) (ii) You Run your code then press the button ***Print Screen SysRq*** on your keyboard. So doing copies the content of the Window to the Clipboard. You then use paste methods of Windows: Ctrl + V or Shift + Insert depending on your windows setting. See your operating system manual for more on Printing or Copying your *Window*.

§9.2 Miscellaneous Test

Carry out a research on how file compression is done. Try to code a compression program to compress any of the following file formats (*.bmp, *.jpg, and *.txt). You can get information on the web by searching for the following words (Lemple-Ziv-Welch-LZW, Huffman Compression, Run-Length Encoding – RLE, Code Table Optimization) using any good search site, but I recommend http\\www.google.com , your school library or a state Library might be of invaluable help.

Even if you *cannot* code them, in a case where you are not very optimistic, going through the Algorithms would let you appreciate WINZIP™, Stacker and the like used to compress files.

Bibliography

- Erwin Kreyzig (2001), **Advanced Engineering Mathematics**, John Wiley & Sons (Asia) – p1085
- Microsoft QuickBASIC™ **online help**
- Rob Thayer(1998),**Visual Basic 6 Unleashed**, Sams Publishing

QBASIC Programming Without Stress is an introductory book for people interested in programming or students taking a course in computer programming. The book introduces basic concept of programming using the Microsoft QBASIC programming language. Understanding the basics of any programming language can help to understand what it takes to be a programmer.

This book was written in 2004 by Akinola A Adeniyi and is available for free download on *Scribd*.com.

AAA0001QPWS001