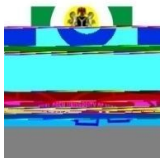
NATIONAL OPEN UNIVERSITY OF NIGERIA

COURSE CODE: CIT 215

COURSE TITLE: NUCLEAR AND RADIOCHEMISTRY

Course Code	CIT 215
Course Title	Introduction to Programming Languages
Course Developer/Writer	Dr. Sunday Reju National Open University of Nigeria 14/16 Ahmadu Bello Way Victoria Island, Lagos
Course Coordinator	Dr. (Mrs.) C.A. Okonkwo National Open University of Nigeria 14/16 Ahmadu Bello Way Victoria Island, Lagos
Course Team	Mr. A. M. Balogun Mr. Bankole Abiola Dr. Sunday A. Reju



NATIONAL OPEN UNIVERSITY OF NIGERIA

National Open University of Nigeria
Headquarters
14/16 Ahmadu Bello Way
Victoria Island
Lagos

Abuja Office
No. 5 Dar es Salaam Street
Off Aminu Kano Crescent
Wuse II, Abuja
Nigeria

e-mail: centralinfo@nou.edu.ng
URL: www.nou.edu.ng

First Printed 2006
Second Print 2007

ISBN: 978-058-266-5

Printed By **ADESOLA PRINTS & CO. LTD.**
24 Yusuff Street, Papa-Ajao
Mushin, Lagos

For
National Open University of Nigeria

TABLE OF CONTENTS **PAGE**

MODULE 1

UNIT 1 THE ART OF COMPUTER PROGRAMMING

CONTENTS

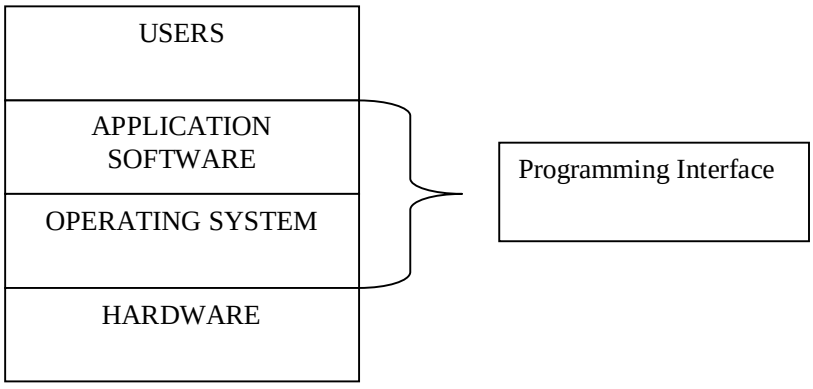
1.0 INTRODUCTION

In this unit you are going to be introduced to the principles of computer programming. As you know, a Computer is not a useful device as an entity without a programming force driving its operations.

Generally, a complete a computer system is made up of the following:

- Hardware
- Operating System Software
- Application Software

In fact, a better way of illustrating the above is as follows:



As you can see above, programming plays a very essential role in the usefulness of a computer system, forming the interface link between human users and the computer machinery. This unit will therefore take you through the fundamentals of computer programming

2.0 OBJECTIVES

By the end of this unit, you should be able to:

- Explain the major types of programming
- Discuss the fundamental principles of programming
- Identify computer programming with problem-solving process.

3.0 MAIN CONTENT

3.1 Types Of Computer Programming

As you are now aware, a Computer is simply a device manipulate I by a person.
Moreover, a complete functional Computer system consists of the hardware and software, that is the programs. Now what is a Program itself?, you may ask. The following definition gives the answer.

Definition of a Computer Program

A PROGRAM is a series of step-by-step instructions that provides a solution to a particular problem and directs the computer on what to do exactly. Now, though a program is a set of instructions, but the statements must be submitted to a computer as a unit to direct the computer behaviour. There are generally two major types of programming:

- System Programming
- Application Programming

You will now study the features of the above two types of programming System Programming.

In short, system programs constitute the driving force behind 'tile operations of the Computer System. They are specially designed to facilitate the use of the hardware and to make the Computer System function efficiently and run quickly.

During the early days of Computer Systems, human operators monitored

computer operations, decided on the order in which submitted programs should be run and made ready the input and output (I/O) devices. Even though the speeds of Central.

Processing Units (CPUs) increased as a result of early electronic revolution, the speed of human operators behind the operational procedures of the computer did not increase. There were therefore time delays and errors by the human operators which constituted most of the problems that led to the development of a Super-Controller program)* handle the problems caused by human-operators. This special program is what we call an OPERATING SYSTEM (OS). See its definition below:

Operating System

An Operating System is a collection of system programs that jointly controls the operations of a computer system and its resources. An **Q** simply helps you to efficiently and reliably run other programs At manage your files. Now, with an OS installed on your computer system, most)f the responsibilities of human operators that characterized Abe early generation computer operations, such as preparing the I/O device to be used for each program or loading the programs into memory, are now all done by the operating system.

There are two types Of programs that make up the Operating System:-

- Control Programs
- Processing Programs

Control Programs

The OS control programs generally oversee the system operations **skid** carry out tasks such as Input/Output (I/O), scheduling, communicating with the Computer user or programmer and handling interrupts. An interrupt is just a signal sent to the CPU indicating that an event 'hat occurred.

Processing Programs

The OS processing programs are those that facilitate efficient processing operations by simplifying program preparation and execution for you as a user. The major processing programs existing in the OS are as follows:

- Language Translators
- Linkage Editor
- Library Programs

- Utility Programs

Full discussions on each of the above programs are beyond the scope of **this course. You** will learn more of them when you take full course on **OS. All that have** been said so far are on system programming. As a **reminder, take a break** and attempt the following exercise.

SELF ASSESSMENT EXERCISE 1

i. **What is a program?**
Define system programs

But before concluding on the brief review of OS, the following are some **common examples of** operating systems:

- **Disk Operating** System (DOS) — Microsoft original OS.
- **Microsoft** Windows Operating Systems (Window s 95, 98. **2000)**
- **Microsoft** Windows NT
- Microsoft Windows XP
- Unix
- Linux
- Novel Netware

The above are found on most Personal Computers (PCs). However, we equally have the:

- Macintosh Operating System - which runs only on the Apple Macintosh machines.
- Palm OS — which runs on Handheld computers, PC Phones or Palmtops.
- Windows CE — a slimmed version of Windows that runs on handheld PCs.
- Symbian OS that runs on PDAs or Mobile phones like Nokia 9300/9500

Application Programming

Application Programs are those that perform specific computational **tasks or** data processing to solve user's problems. From this definition, you can see that application programs concentrate on the particular problems to be solved.

Broad categories of application programs are stated below:

- Word Processing Software
- Database Applications
- Electronic Spreadsheet applications
- Desktop Publishing applications
- **Web** Publishing Software
- Communication Software
- Accounting Software

- Graphic Tools
- Design Software (e.g. CAD Packages)
- Modeling Software
  Video Editing
- ... and others

Full details of some of the above types of software are in another course: "Software Application Skills" in the School of Science and Technology. You may wish to refer to it. Now you will go to the next section of this unit.

3.2 Basic Principles Of Programming

As you have already seen in the last section, a computer program is designed to solve a specific problem following the execution of the program instructions. However, you should know that a Computer does not solve problems the way we do. Human beings make use of reasoning, intelligence and intuition to solve problems while computers solve problems according to instructions supplied by programmers.

In view of the above, every program has to be properly designed so as to solve **the** problem behind its development effectively and accurately. Before you are introduced to the basic steps involved in program development, you should know the aims guiding the design of a good computer program. These are as follows:

- Reliability
- Maintainability
- Portability
- Readability
- Performance
- Memory Saving

Let us see these one by one.

Reliability
By reliability, we mean that you should be able to depend on the program to always do what it has been designed to do.

Maintainability
By this, we mean that you should be able to modify the program when the need arises.

Portability
The concept of portability in programming is that a program should be capable of being transferable to a different computer platforms with a

minimum modification, if any at all.

Readability

A program should be easy for other programmers to read and understand. For example, a readable program is easier to maintain.

Performance

A program that doesn't carry out the expected tasks quickly and efficiently has lost the performance aim. Therefore, a major aim in program design is that the program should execute quickly and efficiently too.

Memory Saving

What is meant here is simply that a program should not be unnecessarily too long and requiring excessive large memory to execute.

Having gone through the aims underlying the designs of programs, you will now see below the different stages involved in program development:

- Problem Definition
- Solution Design
- Program Coding or Writing
- Program Testing
- Program Documentation and Maintenance The above will be explained further as follows:

Problem Definition

Problem formulation or definition is very essential in programming and it begins with recognition of a need for information by a user or an organisation. The programmer is expected to analyse the problem thoroughly in order to understand what is required of its solution. Generally, if you describe a problem carefully at the beginning of the programming process, your program will be better and might cost less to develop.

One way of defining a problem is to do that in terms of the following:

- Input
- Output
- Processing

Starting with **OUTPUT,** this represents the information requirements of users of the program. This is the reason most of the times the programmer can simply use a report generated by a program to design the corresponding input form or interface.

Having determined the output requirements, then the **INPUT** required to

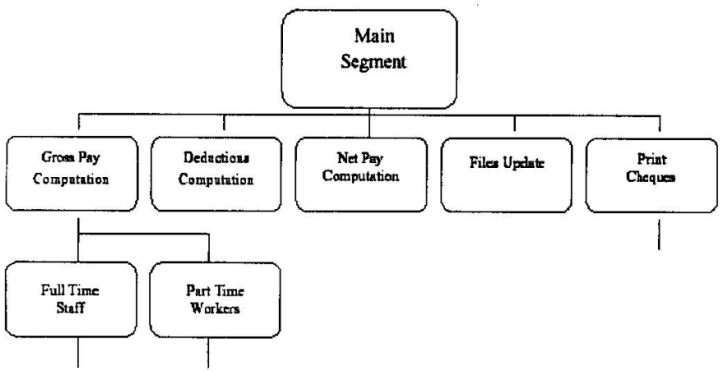
provide the output should be determined too. Finally, based on the input and output requirements, the **PROCESSING** can then be determined.

Solution Design
After the definition of the problem is completed, the design of the solution is the next step and this may take the form of one or more programs.

What is best to do here is for the programmer to take each step or segment of the problem definition and then work out a tentative program flow. When you approach the solution by handling each segment separately, you can concentrate on developing an efficient and logical flow for that segment. This is the approach employed in what is called **"Modular Programming",** where a program is divided into parts or modules for easy development and maintenance.

To develop a program logical flow, there are two major aspects:
- General Logic
- Detailed Logic

The *"general logic"* flow design can be done by using a *"Structure Chart"* which shows the major elements of the program and their relationships to each other. After the general logic description, you must deal with the *"detailed logic"*. This is simply the step-by-step operation of each block in your structure chart. Below for example is a structure chart for a simple Payroll program:



By *detailed logic*, we mean each of the boxes in the above chart should be described in clear terms. In *detailed logic* description, you will definitely need to use such tools as Flowcharts and Pseudocode which will be treated later in this course.

In general, there is nothing frightening about writing a C Imputer Program after describing the problem and its solution; you simply need to understand the basic logic patterns involved in programming which

will be fully described under "Structure Programming" unit in this course, that is, in unit 6.

Program Coding or Writing

Next to the two steps of program development described above is the coding or writing of the program itself in a specific programming language, especially High Level Languages (HLL), such as Basic, Pascal or C++, which you will be introduced to in the next unit.

Generally, the definition and solution of a problem do not depend on a particular programming language. But most of the times, the proposed solution may limit the choices of languages that can be employed. It is also necessary to know that some language' are better suited for some types of problems as you will see in the next unit.

Program Testing
After the coding or writing of a program, it is submitted to the computer for testing. Generally, testing involves the following:

- Debugging
- Compiling
- Testing (in stages)

Debugging
Errors in programs are usually called *bugs* and the processing of removing errors in your programs is called *debugging.*

Compiling
Though you will be taught in detail the process of program compilation in unit 8, but you should know that your program has to be *translated* before the computer can execute it. Compiling is one way of translating your program. The other is by using *Interpreters,* which will be treated ahead in unit 8.

Testing
Usually, for a large program that has been developed using modular metho I, there are various stages of testing as follows

- Unit Testing
- Integration Testing
- System Testing
- User Testing

Unit Testing involves testing the separate components or modules as they are being developed. **Integration Testing** involves testing the program as separate modules are put together. Finally, on the part of the programmer, **System Testing** occurs when the whole program is being

tested in its final form to be ready for use. However, there is also the usual need for the user's testing. This is when the user of the program tests the final program to see whether it meets his or her needs.

Program Documentation and Maintenance

There is no good programming without documentation. This is the documentation of all the work involved in the program development. The documentation should consist of all written descriptions and explanations of the program and other materials associated with the development.

Generally, proper documentation serves as a reference guide for programmers and system analysts who are to modify the programs and their procedures when the needs arise. When an organisation grows for example, program modifications must keep pace with its changing needs. Hence the process of documentation is an ongoing one. Maintenance includes any activity aimed at keeping programs in working condition, error-free, and up to date. You will now round up this unit.

4.0 CONCLUSION

In this unit, you have been introduced to the two main types of programming, namely, System programming and Application Programming. As you have learned in the unit, system programming facilitates the use of the Computer hardware and the running of the application programs. The application programs are simply those that solve the users' problems. The unit has equally taken you through the basic stages involved in programming.

5.0 SUMMARY

This unit has shown you that the computer machine is simply an electronic device which only becomes a useful tool through its programming. According to what you have learned in the unit, every computer program is designed to solve a specific problem or perform a particular task. Hence the art of programming starts by defining your problem and then followed by the designing of the solution. A very vital aspect of programming from what you have learned in this unit is the documentation of your program and this should be an ongoing process for a program that is regularly modified to keep pace with the changing needs of its user.

6.0 TUTOR-MARKED ASSIGNMENTS

I. Define a program and hence distinguish between system

programming and application programming.

2. What are processing programs?. Give examples of these in an operating system.
3. (a) Discuss portability and maintainability in program **design.**
   (b) What is integration testing?

7.0 REFERENCES/FURTHER READINGS

Brightman, R. W. and Dimsdale, J. M., Using Computers in **an** Information Age, Delmar Publishers Inc., 1986

Mandell, S. L., Computers and Data Processing, West Publishing Company, 1985.

Williams, B.K. and Sawyer, S. C., Using Information Technology, **4, Edition,** McGraw-Hill, 2001

UNIT 2 COMPUTER PROGRAMMING LANGUAGES

CONTENTS

1.0 INTRODUCTION

Historically, instructions for Computer processing in the early days of the Computer had to be either wired on control panels and plugged into the machine at the start of a job processing or had to be read from punched cards in distinct steps of the job processing. Any of these methods was very slow since the computer had to wait for these instructions to be fed in by a human operator.

To speed up processing, the computer memory became useful to store instructions as well as data. This development introduced what was known as *Stored-Program* concept. The need to represent instructions in one form of code or the other gave birth to the notion of Programming Language and the first form of these codes became what is called the *Machine Language (ML).*

In this unit, the above short history will be built upon to introduce you to computer programming languages and their classifications. Your study objectives for this unit will therefore be as follows:

2.0 OBJECTIVES

By the end of this unit, you should be able to:

- Categorise Computer programming languages.
- State the various types of languages.
- Identify different levels of Computer programming languages.

3.0 MAIN CONTENT

3.1 Classification of Languages

Before you see the general classification of programming languages,

you will recall the origin of programming languages mentioned under the Introduction. You have been introduced to what is known as the Machine Language which is generally known as the language that the computer understands. With the birth of the Machine Language, two broad categories of languages are as follows:

- Low-Level Language
- High-Level Language

Low-Level Language

Machine Language is generally called the lowest- level language and it was the first language available to computer programmers. It is very fast since the language needs no translation because its instructions are made up of zeros and ones (O's and I 's). Thus Machine Language is merely data to the computer and the program can modify itself during execution.

The *advantages* of Machine Language (ML) can be summarised as follows:

- Fast execution speed
- St irage Saving
- Programmer's full control of the Computer and its capabilities

However, ML has some *disadvantages* as follows:

- Difficult to learn
- Highly prone to errors
- Totally machine-dependent — this means that programs written in ML will only execute on the specific machine they were written.

You have been informed that ML is the lowest-level language. Hence it is the most efficient language and it is good for you to know that every computer responds only to its machine language.

The next in the hierarchy of languages that is closer to the Machine Language is the *Assembly Language (AL).* Assembly Language was developed in the early 1950s to alleviate some of the difficulties associated with the Machine Language. Symbolic names or **mnemonics** were used to replace the binary code of the Machine Language. Remember that a mnemonic means a memory aid. Hence AL instructions are easier to remember than the O's and I 's of the ML instructions.

Below are the *advantages* of the AL:

- It is efficient in processing time and in the use of memory space.
- It encourages Modular Programming, where programs are broken into modules.
- It provides an error listing which is useful in debugging.

- Since it is very close to machine language, it is also fast.

Just as you have seen the *disadvantages* of Machine Language, the Assembly Language also has its disadvantages. Some of these are stated below:

- It is cumbersome in usage.
- Assembly Language has one-to-one-relationship with machine language, meaning that one Assembly Language instruction is translated into one Machine Language instruction. This process leads to long program preparation time.
- Assembly Language is machine-dependent like the Machine Language.

SELF ASSESSMENT EXERCISE 1

State the problem associated with machine dependency of ML, and AL.

-

Now, remember that under introduction to this unit, it was stated that languages can be broadly categorised into two as Low-Level and High-Level languages. However, there are various ways of categorising computer programming languages. One way of doing this is to classify them by the following:

- Level
- Purpose
- Orientation
- Structure
- Translation Method

Level Classification

You have already seen this above as low- level and high- level languages. High- Level Languages will be treated in the next section of this unit.

Purpose Classification

Under this classification, you have:

- General-Purpose Languages
- Special-Purpose Languages

A *general-purpose* language is one that can be used to solve a variety of problem types. From what you have learnt already about low- level language, it means that the lower the level, the more general purpose the language. A *special-purpose* language is one that can be used for specific types of problems, such as a language called WPL (Word Processing Language) developed by Apple for word processing.

Orientation Classification

In this classification, you have the following:

- Procedure — Oriented Languages
- Problem — Oriented Languages

In using A *procedure-oriented* language, you have to specify how to solve a problem by indicating the procedures the computer will follow step by step. However, in a *problem-oriented* language, you simply specify what to obtain as your results while the development of the procedures is left to the language.

Translation Method Classification

Among all the Computer programming languages, only the machine language is in machine-executable form. All other languages must be translated into O's and l 's, the only things understood by the computer. Now, translators take the forms of the following:

- Interpreter
- Compiler
- Assembler

Interpreter

Some languages are interpreted by converting the *"source program"* into machine language as the program is being executed. Interpreters translate code line-by- line which therefore makes them run slowly than other translators. For example, sonic BASIC language versions only interpret programs instead of compiling them. However in Turbo Basic, KBASIC or BASIC 4GL you can compile the BASIC source code into an executable code.

Compiler

Unlike an Interpreter, a Compiler translates an entire program into machine language before the execution of the program. A Compiler usually translates the SOURCE program into another program called the OBJECT program which is the machine language version of the source code. With the object program created by your compiler, you will never use the source program again except when you want to modify it. Generally, a compiled program runs faster than an interpreted program.

Assembler

You have already been introduced to the Assembly Language which is neither compiled nor interpreted. For the language, it is simply assembled. Since the assembly language is already close to the machine language, assembling a program is therefore less time-consuming than compilation.

Before you proceed to the next section on High-Level Lang ·ages, it is

worth noting that advancement in computer programming languages has added a new classification of languages, called Very High Level Languages (VHLL's) or what are usually referred to as the 4th Generation Languages (4GL's).

Below are the basic characteristic layers of a simple 4GL:

- Database
- Data Communication
- Data Processing
- End User Facilities (EUF)

From the above, database languages such as FoxPro, Dbase and Foxbase are 4GL's. A language such as Visual Basic which equally has database capabilities can also be classified as a 4GL from the above characteristics.

Now, you will be introduced to the High- Level Languages in the following section.

3.2 High Level Languages

In the previous section, you have learnt about the Low-Level language which is fundamentally machine-dependent. In contrast to machine-dependence of low- level language, High-Level Languages (HLL's) are machine- independent.

High-Level Languages are either

- Procedure-Oriented, or
- Problem-oriented

You will recall that the above two classes of languages were mentioned under *orientation classification* of languages in the last section. High-Level Languages can further be classified again as follows, remembering that there are many ways of classifying computer programming languages:

- Scientific-Oriented Languages
- Business-Oriented Languages
- Multi-Purpose Languages
- Education-Oriented languages
- Natural Languages

-

It is good for you to know that a language can have any of the above characteristics with one of the two earlier features, that is, *problem-oriented* or *procedure-oriented* feature. This should not confuse you. Now, it is time to introduce you to some common high- level languages. Detailed features of some of these languages will be treated in other units in this course.

FORTRAN

You are starting with Fortran because it is generally referred to as the first high- level language. The name is the short for "FORmula TRANslator. Fortran is a scientific-oriented and problem-oriented language. Since early computer users were scientists and engineers, it was not therefore surprising that the first high- level language was designed to solve scientific problems. The language was developed by an IBM (International Business Machines) group led by John Backus in 1957, though its first appearance was in 1956 according to some authors. There have been various versions of the language as follows:

- Fortran II in 1959
- Fortran IV in 1966
- Fortran 77 in 1977
- Fortran 90/95 — the latest version.

You will learn more about the language in unit 10 of this course. BASIC
BASIC is the acronym for *"Beginner's All-purpose Symbolic Instruction Code* and it is the most popular programming language. It is an Education-oriented language developed in 1965 for use by Colleges and Universities for instructional purposes. It is also a general-purpose and procedure-oriented language. BASIC can be interpreted and also compiled. You will learn more about BASIC in this course. Today, we have a number of BASIC language versions which you can use to develop beautifully designed software packages.

PASCAL

Pascal language was designed in 1971 like BASIC as a teaching language. The language was named after Blaise Pascal who was a French Mathematician and Philosopher and the inventor of the first mechanical adding machine. You are going to learn about the fundamentals of the language in unit 12 of this course.

C or C++

The C language was developed as an improvement of the earlier versions, A and B which were developed by Bell Laboratories. C language was developed in conjunction with the UNIX operating system; in fact it is the language of the Unix operating system. The language is lower in level than a language like Pascal but higher than assembly language. Presently, we have the C++ and Visual C++ languages which are the later versions of the original C language.

A good number of big organisations today prefer C++ programmers for their in-house software development because of the essential features of C++ which are absent in other languages. You will also be introduced to

the basic features of C/C++ language in this course. A good knowledge of C++ will help you to understand Java language easily which is an essential language in Web publishing.

COBOL

COBOL stands for *"COmmon Business Oriented Language"* and was developed in 1960 as a language suitable for business applications. Except for some big organisations which are unwilling to rewrite their programs using one of the modern languages, COBOL is almost totally abandoned by programmers today.

There are a number of other high-level languages that you may ha ye no cause to study today because of the new trend in programming. Some of these are listed below:

- ADA - named after Augusta Ada Byron, the 1st Computer programmer. It was developed in 1983 by U.S Defence Department.
- APL - A Programming Language, developed between 1962 and 1968.
- PL/1 - Programming Language 1, developed in 1964.
- LOGO - developed in 1966.
- RPG - Report Program Generator, developed in the 1970's.
- LISP - Short for LIST Processor, developed in 1960 as a Special — purpose language designed to manipulate nonnumeric data.

Though you will not be introduced to the essential programming languages in this course until later in your academic programme, however, there are some vital programming languages you will need to study as a programmer. Examples of these are:

- HTML - Hypertext Markup Language
- Java - an object-oriented language
- SQL - Structured Query Language

You will be introduced to the HTML in this course.

SELF ASSESSMENT EXERCISE 2

Define a procedure-oriented language and give two examples from the list of languages mentioned in this unit.

4.0 CONCLUSION

In this unit, you have been introduced to the general classification of computer programming languages, namely, low- level and high-level languages. The two low-level languages are the machine language and the assembly language, the former being the lowest.

The machine language, as you have learnt in this unit is the only language understood by the Computer and all other languages have to be translated into the language for the Computer to execute their instructions.

5.0 SUMMARY

This unit has presented to you an overview of computer programming languages. Apart from the low- level languages, namely, the machine and assembly languages, you were introduced to some common examples of high-level languages The basic characteristics of some of these languages such as BASIC, FORTRAN, PASCAL, C++ and HTML will be treated later in this course.

6.0 TUTOR-MARKED ASSIGNMENTS

1. Identify the first, second and third generation languages.
2. (a) Give two common advantages of the machine and assembly languages.
   (b) How are languages translated into machine code?
3. (a) Classify the languages below:
   (i) Fortran (ii) BASIC (iii) C++
   (iv) Pascal
   (b) State the features of 4GLs.

7.0 REFERENCES/FURTHER READINGS

Brightman, R. W. and Dimsdale, J. M., Using Computers in an Information Age, Delmar Publishers Inc., 1986

Mandell, S. L., Computers and Data Processing, West Publishing Company, 1985.

UNIT 3 INTRODUCTORY THEORY OF ALGORITHMS

CONTENTS

1.0 INTRODUCTION

As you already learnt in unit 1, a Computer program is usually designed to solve a specific problem. However, a PROGRAM belongs to a larger class of problem-solving techniques known as ALGORITHMS. This unit is therefore aimed at introducing you to the basic theory of problem-solving and its relationship with algorithms. You will also be taken through the fundamental features of an algorithm in the unit.
You will now see your study objectives in the next section. 2.0

OBJECTIVES

By the end of this unit, you should be able to:

- Define the basic concepts of problem theory
- Explain the principle of solving complex problems.
- Discuss the characteristic features of an algorithm
- State various forms of representing algorithms.

3.0 MAIN CONTENT

3.1 Problem Theory and Algorithms

The design and development of algorithms rest solely on the principles of problem theory. You may ask, what is an algorithm? Precisely, an algorithm is a procedure for solving a particular problem. This definition will be expanded more in the next section. It is good to also quickly state here that a PROGRAM is an ALGORITHM but the converse is not true. An algorithm has to be expressed in a high- level language to qualify as a program.

Now, every problem is usually characterized by the following states:

- Initial State
- Goal State.

Every algorithm is designed to transform a problem from its Initial (or unsolved) state to its Goal (or solved) state by means of permissible operators or steps specified by the algorithm. You will need to see an important definition below.

Definition (Stepwise Refinement)

Complex problems that cannot be solved directly are usually simplified into sub-problems and sub-goals until you obtain a sub-problem state that you can solve directly. Such a problem solving method is called Stepwise Refinement.

The principle of stepwise refinement is usually employed in Computer program development in which an initial highly abstract representation of a required program is gradually refined through a sequence of intermediate representation to obtain a final program in a chosen programming language.

Before you go to the next section, below is an important set of concepts in algorithm theory closely related to problem-solving principle.
Problem State Assertions

The description of problem states is often given by what you call ASSERTIONS in problem theory. The assertions are classified as follows in association with algorithm development:

- Initial Assertion
- Final Assertion
- Invariant Assertion
- Pre-condition
- Post-condition.

The above types of assertions are discussed briefly as follows:

Initial Assertion
This is the declaration or statement that precedes the first step of an algorithm and it simply describes the initial problem state before the execution of the associated algorithm.

Final Assertion
This is the assertion following the last step of an algorithm which describes that goal state of the associated problem, that is, after the associated algorithm has been executed.

Invariant Assertion
This is an assertion that is true under the execution of a step in an algorithm irrespective of the number of times the step is repeated (both before and after the execution of the step).

Pre-condition
A precondition is the assertion preceding a step of an algorithm. By this definition, you will see that Initial Assertion is a precondition.

Post condition
A post condition is the assertion following an algorithm step and by this definition, it also means that Final Assertion is a post-condition. Generally, precondition and post-condition are called Intermediate Assertions and they generalize Initial and Final Assertions respectively.

3.2 Basic Features of Algorithms

In the last section, you have been introduced to the fundamental concepts of problem theory and the fact that an algorithm is a procedure for solving a particular problem. Now, you will be going through the features of an algorithm. As stated in the last section, below now is an expanded definition of an algorithm.

Definition (Algorithm)
An Algorithm is a prescribed set of well-defined rules or instructions for the solution of a problem in a finite number of steps. On the other hand, you can define an algorithm to be a sequence of well-defined, finite steps, ordered sequentially to accomplish a task.

In fact, most recipes for dishes are simply algorithms, however, with the avoidance of a statement such as "Add Salt to taste". Why? The reason is that an algorithm step should not contain such an ambiguous statement.

Now, you will see below the features or characteristics of an algorithm. You may as well call them Algorithm Criteria as stated below.

Algorithm Criteria
Every algorithm is expected to satisfy the following criteria:

- Input Criterion
- Output Criterion
- Precision Criterion
- Finiteness Criterion
- Efficiency or Effectiveness Criterion.

The above five criteria will now be discussed below. Input Criterion
There should be zero or more values which are externally supplied to the algorithm.

Output Criterion
An algorithm should provide an output of implementation or experimentation. This is very essential to check the correctness of the algorithm.
Precision Criterion
Each step of an algorithm must be clear without any ambiguity or any inherent assumptions.

Finiteness Criterion
An algorithm must terminate after a finite number of steps. There should also be a stopping criterion to terminate an algorithm in a case of a step with repeated execution.

Efficiency Criterion
Each step of an algorithm must be sufficiently basic that it can be carried out in principle. That is, each step must be feasible (in addition to being definite or precise). There must never be an impossible task included in algorithm steps. In actual programming, an example of impossible tasks is when a quantity is being divided by zero.

You will now be introduced to an essential method of measuring the efficiency of an algorithm, presented under what is usually termed algorithm complexity.

Algorithm Complexity
What is meant by algorithm complexity here is simply the difficulty of solving computational problems, measured in terms of some resources employed during the computational process. This is described below.

Suppose P is an algorithm with n as the size of the input data. Usually, Time and Memory (or space) employed by P are two major measures of the efficiency of P. While time is measured by counting the operations that consume the maximum time among other operations, the space is measured by counting the memory locations needed by the algorithm. You can see from the above description that algorithm complexity is a function *f(n)* which gives the running time and/or memory requirements of the algorithm in terms of the size *n* of the input data.

Generally, the memory requirements of an algorithm are some multiples of input data size and the program size. Now, in brief, an algorithm is said to be efficient if it requires the following:

- Minimum Memory
- Minimum Time.

But generally, unless otherwise stated, algorithm efficiency mostly refers to the running time of the algorithm and it is not usually possible to of sin an algorithm with the minimum time and memory features. Hence you have to make a choice between alternate algorithms; that is, one with minimum time or one with minimum memory.

Algorithm Representation

Below are the common forms of representing algorithms:

- Pseudocode
- Flowcharts
- Formulae.

There are other forms of representing algorithms which you may not bother about for now at this level. Some of these forms are:

- Decision Trees
- Nassi-Shneidermann Structured
- Flow Diagrams (NSSF)
- Warnier-Or Diagrams.

Apart from the first three forms listed above, the last three are seldom used today despite some of their merits, especially the NSSF diagrams. You will be introduced to flowcharts in the next unit. However you will see the definition of Pseudocode below.

Definition (PSEUDOCODE)

A Pseudocode is a logical representation of an algorithm using third generation language (3 GL) style such as DO, WHILE, IF-THEN-ELSE, etc. Pseudocode is also called structured English since it is English- like in structure. It is essential to know that a Pseudocode is not directly executable on a computer except it is transformed into a high- level language code.

Interestingly, Pascal language has a general form for its Pseudocode which cannot be said of other languages. The general form is as follows:

ALGORITHM name
DECLARE
    definitions and declarations
EXECUTE

```
        statements to be executed
END name.
```

As you can sec above, a Pascal Pseudocode is simply made up of four blocks:

- Algorithm Name
- Declarations
- Executable Statements
- End of algorithm.

See an example below.

Example 1 (Pascal Pseudocode)

```
ALGORITHM average
DECLARE
        a, b, c : REAL
EXECUTE
        INPUT a, b
        c ? (a+ b)/2
        OUTPUT 'c=', c

END average.
```

The above example finds the average of two numbers. You can see below how the above Pseudocode can be transformed easily into a Pascal code.

Example 2 (Pascal Code)

```
PROGRAM Average (INPUT, OUTPUT);
VAR a, b, c : REAL;
BEGIN
        READ (a, b);
        c : = (a + b)/2;
        WRITTEN (' c=', c);

END.
```

You will return to this example in the unit on Introduction to Pascal Language in this course. However, you can see how it is very easy to transform a Pascal Pseudocode into its associated high- level language Pascal code.

The symbol ? in the Pascal Pseudocode in Example 1 is called the

ASSIGNMENT OPERATOR or SYMBOL and it is used in place of equality sign. The general usage is as follows:

variable ? expression.

The above is read as *"expression is assigned to variable"* or *"variable becomes expression"*

In the Pascal pseudocode above if you have an output statement as follows:

OUTPUT 'c = `, c, MORE

the equivalent Pascal code statement will become:

WRITE ('c = `, c);

which means that the next output will be printed on the same line.

SELF ASSESSMENT EXERCISE 1

i. Why do you think the END statement in the Pascal Pseudocode in Example 1 has to bear the name again?

WRITE: *variable list*

Or

OUTPUT *variable list*

As you have already seen in the example above, an output statement could consist of a message or a string of characters to make the output more meaningful to the user.

Instruction Execution

The instructions in an algorithm are usually executed one after the other as they appear in the algorithm steps.

Algorithm Completion

Generally, an algorithm is completed with the execution of the last instruction. However, an algorithm can be terminated at any intermediate state by using the exit instruction.

4.0 CONCLUSION

In this unit, you have learnt the basic concepts of problem theory and how they are related to algorithm development. The unit has also shown you the essential criteria that characterize ever algorithm. If you remember, they are:

- Input
- Output

- Precision
- Finiteness
- Efficiency.

Finally, in this unit, you **have been introduced** to some various forms of representing algorithms.

5.0 SUMMARY

Problem theory is very fundamental to the design and development of algorithms as you have been taught in this unit. The unit has therefore taken you through some basic concepts of problem theory.

As you will remember, a major concept in problem theory is ASSERTION which describes the state of a problem as it is being transformed by algorithm steps. This unit has also specifically shown you the major characteristics of algorithms and how they are represented in various forms. The next unit will specifically take you through a very essential and common v■ ay of representing algorithms which is very basic to computer programming design.

6.0 TUTOR-MARKED ASSIGNMENTS

1. How does the method of Stepwise Refinement affect the development of an algorithm?.....
2. What is the essential factor in the measurement of an algorithm efficiency?.....
3. Construct an algorithm to compute the sum of n integers starting with the kth integer.

7.0 REFERENCES/FURTHER READINGS

Reju, S.A., Lecture Notes on Theory of Algorithms, Unpublished Manuscript.

Safaria, R.S., Computer Oriented Numerical Methods, Khanna Book Publishing Company, Delhi, 1999.

UNIT 4 FLOWCHARTING TECHNIQUES CONTENTS

1.0 INTRODUCTION

The task of computer programming rests to a large extent on the ability of the computer programmer to properly make use of the initial analysis and design tools available to him. Programming is one of the essential components of the broad task of system analysis and design. For analysis and design work to be useful, it must be expressed in one understandable form or the other. In general practice, analysts and programmers make use of diagrams and charts as important tools. In this unit, you will therefore be introduced to the use of flowcharts, specifically in programming. Below are your study objectives for this unit.

2.0 OBJECTIVES

By the end of this unit, you should be able to:

- State the types of flowcharts employed in programming.
- Identify various ANSI (American National Standards Institute) symbols used in flowcharts.
- Construct simple flowcharts for simple programs.

3.0 MAIN CONTENT

3.1 Types of Flowcharts and Symbols

Under the broad area of System Analysis and Design of which programming is a component, you can classify diagrams and charts used by analysts and programmers as follows:

- Data Flow Diagrams (DFD).
- System Flowcharts
- Forms Flowcharts

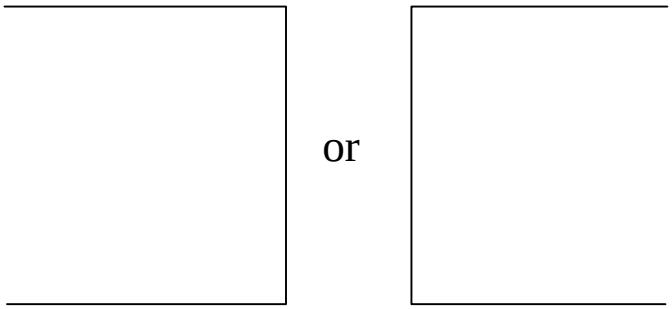
- Program Flowcharts
- Hierarchy or Structure Charts
- Hierarchy plus Input-Processing-Output (HIPO) Charts.

Though your emphasis will be or Program Flowcharts in this unit, however, see some brief descriptions of the above list as follows.

Data Flow Diagrams

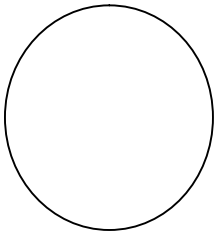
The common symbols use DFD d in are
- Open-ended rectangle or Sinks.
- Circles or Bubbles.
- Arrows.

The Sinks are as shown below



They signify files or other sources of information.

The Bubbles are as stated above circles:



A bubble is an operation or a procedure that transforms data.

The arrows ? represent the flow of data and they are usually accompanied by comments to indicate the flowing data. There is what is called SANDWICH PRINCIPLE which states that "Every bubble falls between at least two data flow arrows".

System Flowcharts

While Data Flow Diagrams focus on the flow of data and the general operations in using data to obtain results, system flowcharts perform the

same role but in addition specify the data processing techniques to be used. Since system flowcharts provide more details, they are therefore referred to as *"Device-Specific" _,* especially when automated processing methods are employed.

Forms Flowcharts
The DFDs and system flowcharts already considered give no any indication of the units of an organization that perform the data processing task and which units use the information. However, Forms Flowcharts are simply employed to supply this information of how documents and forms flow among the organizational units. They don't indicate how data are processed.

Program Flowcharts
You have already seen three types of charts above that provide the overview of the flow of data and the assignments of functions among the agents in a system. When all the agents are human, you call the system a MANUAL SYSTEM. But when computers are employed to perform some procedures, the system is called a COMPUTERISED SYSTEM.

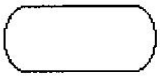
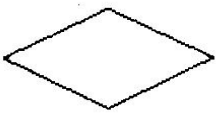
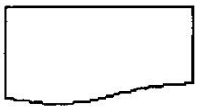
Now, program flowcharts or what are also called "Logic or Block" diagrams are the charts employed to depict the step-by-step procedures to be used by a computer system to process data. Usually, the flowchart symbols are arranged in the same logical sequence in which corresponding program statements will appear in the program.

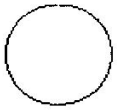
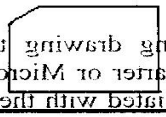
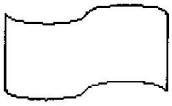
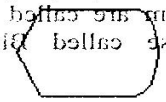
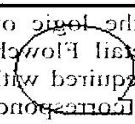
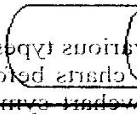
As a programmer, you are expected to be very familiar with how to construct a good program flowchart. This is because flowcharts provide excellent way of documenting your program. Moreover, flowcharts serve as good maintenance tools since they can be employed to guide the programmer to determine what statements are required to make any necessary modifications and where to locate them. Thus an updated flowchart provides good documentation for a revised program.

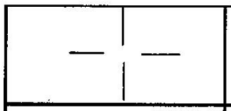
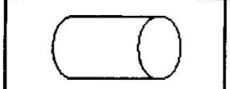
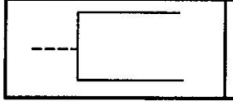
Now, at this juncture, it is in order to quickly introduce you to the flowcharts symbols as approved by ANSI (American National Standards Institute).

Flowchart Symbols
Flowcharts generally make use of symbols that are assigned specific meanings. The common symbols are as follows, in the table below:

Table 1 ANSI Flowchart Symbols

SYMBOLS	DESIGNATION OR NAME	MEANING
	Oval	Terminals
	Rectangle	Process
	Diamond	Decision
	Parallelogram	Input/Out
	Rectangle with Double Vertical Bars	Predefined Process or Modules
	Curve-sided Quad	Document or Prin\ d Output
	Multiple Curve-sided Quads	Multi-document
	Hexagon	Preparation or Initialization
	Trapezium	Manual Input

	Symmetric Trapezium	Manual Operation
	Rectangle with Vertical & Horizontal bars	Internal Storage
	Small Circle	Connector
	Pentagon	Off page Connector
		Card
		Punched Tape
	Vertical Cylinder	Magnetic Disk
		Display
		Sequential Access Storage
		Stored Data

	Arrows or Flowlines	Flow Directions
		Tape
	Horizontal Cylinder	Direct Access Storage
	Open Rectangle	Comments

There are yet some few other symbols not included in the above table that are among the symbols adopted by ANSI and also ISO (International Standards Organisation).

Now, you can draw these symbols easily by using drawing tools available in some software packages such as Flowcharter or Microsoft Word Autoshapes. You will be given examples associated with the use of the above flowchart symbols in the next section.

Program flowcharts can be divided into two:

- Macro Flowchart
- Micro Flowchart.

A macro flowchart is the flowchart that outlines the general flow and major segments of a program. It is also called the "Modular program flowchart" where the major segments of the program are called the modules. Specifically, macro flowcharts are those called Block Diagrams as earlier mentioned.

Unlike macro flowcharts, a micro flowchart shows the logic of the program in full detail. Hence it is also called a "Detail Flowchart". Micro flowcharts usually depict the processing steps required within a particular program and often consist of a one-to-one correspondence between flowchart blocks and program statements.

Now, you remember that you were being introduced to various types of flowcharts and you are yet to treat structure and HIPO charts before it was seen to be getting a bit late to introduce the flowchart symbols. Now, see the brief descriptions of the remaining two types below:

Hierarchy or Structure Charts
Structure charts are diagrams that depict the procedures of an operation in a hierarchical form. A typical example of a hierarchical chart is the one showing the organizational structure of an organization. Structure charts are usually read from top to bottom and left to right and they concentrate majorly on those procedures that should be executed to perform the job or each module. This is referred to as "Top-Down analysis".

HIPO Charts
HIPO (pronounced "hypo") as you have already learnt stands for Hierarchy plus Input-Processing-Output. HIPO charts have design approach similar to that of structure charts and they are more concerned with WHAT is done than with HOW it is done. Since the emphasis in this unit is more concerned with flowcharts that directly provide aids to programming, you will therefore presently not be bothered with detail description of HIPO charts.

SELF ASSESSMENT EXERCISE 1
Among the various types of charts introduced above, identify those that provide details essential to practical programming.....

Answer
Program flowcharts and system flowcharts, but more specifically the former (i.e. program flowcharts).

You will now go through some simple applications of flowcharts.

3.2 Applications of Flowcharts

In this section, you will be introduced to some simple examples of problems with their associated flowcharts. You will start with the example in unit 3, having the following Pseudocode.

```
Start
        read: a, b
        set c = (a+b)/2
        write: c
End.
```

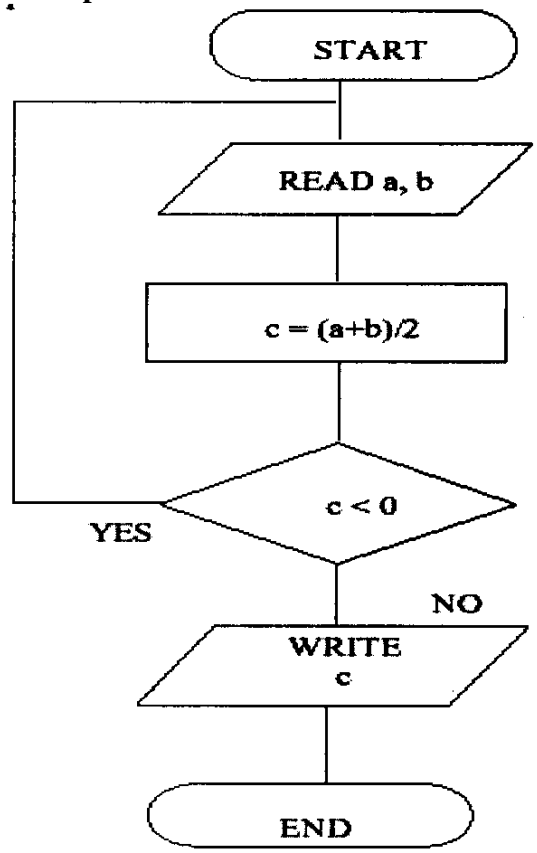
However, remember you used Pascal Pseudocode for the above example in unit 3. Before drawing the simple flowchart for the problem, modify the Pseudocode as follows:

```
Start
       read: a, b
        set c = (a + b)/2
```

if c <0 then
write: "Average is negative"
re-start
else
end if write: c

End.
The flowchart for the above problem is as follows:

Example 1



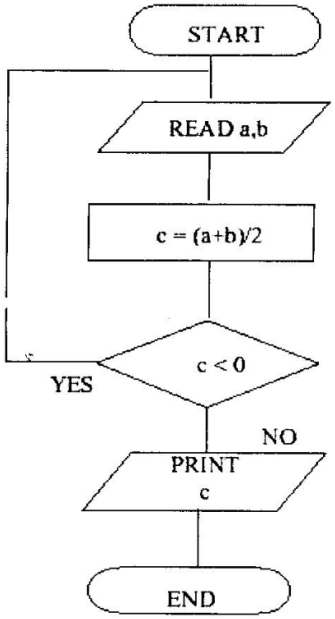
The above flowchart uses the most commonly used symbols, namely: •
Terminals

- Input/Output
- Processing
- Decision,

The original problem was deliberately modified for you to use the Decision box in the flowchart.
Now your exercise below

SELF ASSESSMENT EXERCISE 2

Reconstruct the above flowchart for the same problem with manual input and a print output.
Answer
The new flowchart is as follows:



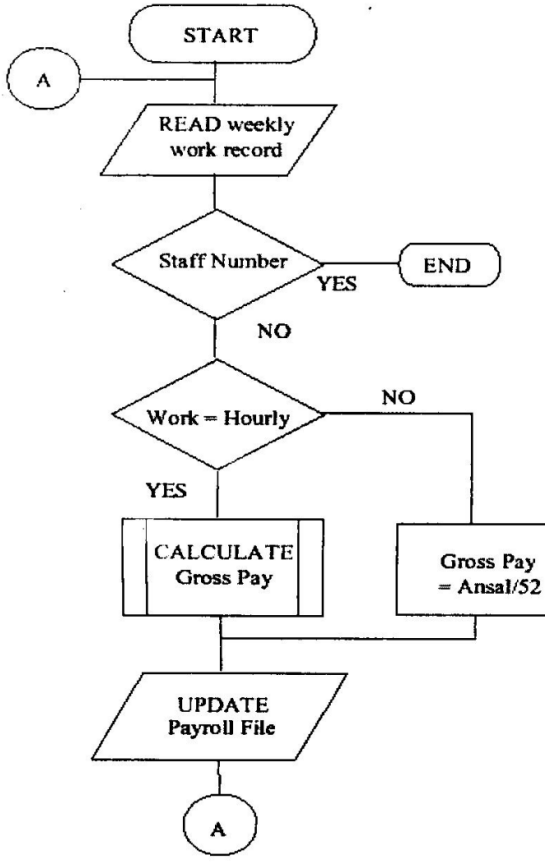
You observe that "PRINT" is used in the Document Symbol to make it more appropriate. The same word can be used in the Pseudocode since you are free to be flexible in your use of words in Pseudocode.

You will now go through another example, having the following Pseudocode:

Example 2

```
START
        DOWHILE staff number ? 0
        READ: staff weekly work record
        IF work is hourly THEN
                CALCULATE gross pay
       ELSE
                Pay = (annual salary)/52
       ENDIF
       Update payroll file
       ENDDO
END.
```

In the above Pseudocode, the calculation of the gross pay is taken to be a pre-defined process. The flowchart is as follows in an unbroken form:



To minimize the use of a long flowline, a connector has been used in the above flowchart. In the flowchart, you will observe that the program is terminated when the staff number is entered as zero. As you will observe in the flowchart, expressions have to be shortened in forms that the boxes can easily accommodate.

As you have already learnt in the previous section of this unit, the predefined process box is representing a module of the program, which in the above example is the segment of the program that computes the Gross Pay.

Later in your course of study, you will definitely came across more complex problems that will demand the use of more of the flowchart symbols you have already been introduced to in this unit. You will now round up the unit.

4.0 CONCLUSION

In this unit, you were taken through the general types of flowcharts employed in system analysis and design with special emphasis on program flowcharts.

As you learnt in the unit, program flowcharts can be divided into two: Macro flowcharts and Micro flowcharts. While the former focus on the major segments of a program, the latter show the logic of the programming in full detail. There are standard symbols approved by ANSI and ISO as presented in Table 1 in the unit. You need to get used to the symbols since flowcharting provides a very good way of documenting your program.

5.0 SUMMARY

The unit has introduced you to the fundamentals of flowcharting as an essential step to good programming. You were also taken through some examples of flowcharts, using the most common symbols. You need to practice with more complex problems that will suggest the use of more symbols of flowcharts.

In summary, below are some rules that guide flowcharting:

- Organise flowcharts in modules
- Use the standardized symbols approved by ANSI and ISC
- Vary symbol size but not shape
- Maintain good and consistent spacing between symbols for good readability.
- Arrange program flow from top to bottom and left to right.
- • Never cross flowlines
- Avoid the use of connectors unless they are necessary to avoid too many breaks in the flowchart
- There must be no path or flowline that goes nowhere. This is a rule that is consistent with the concept of a "Proper Program" that has one entry and one exit.

6.0 TUTOR-MARKED ASSIGNMENT

1. What is the sandwich principle and in what type of chart is it applicable?
2. Distinguish between system flowcharts and program flowcharts.
3. Let f(x) be a given function with xmin and xmax as the minimum and maximum end points over which f(x) is to tabulated. Let dx be the increment in x. Study the following pseudocode and then construct an appropriate flowchart for it.

```
START
        READ: xmin, xmax, dx
        Set x = xmin
        WHILE (x = xmax) DO
                Set y = f(x)
                WRITE: xly
                Set x = x+dx
        END WHILE
END.
```

7.0 REFERENCES/FURTHER READINGS

Brightman, R.W. and Dimsdate, J.M., Using Computers in an Information Age, Delmar Publishers Inc., 1986.

Mandell, S.L., Computers and Data Processing, West Publishing Company, 1985.

Salaria, R.S., Computer Oriented Numerical Methods, Khanna Book Publishing Company, Delhi, 1999.

UNIT 5 STRUCTURED PROGRAMMING CONTENTS

1.0 INTRODUCTION

In this unit, you will be taught an essential method of programming, known as "Structured Programming". The structured programming design is a programming tool developed in the 1960s as a way of defining the elements of a problem and as at today it's the best approach to all computer programming tasks.

This unit is aimed at introducing you to the basic elements of structured programming with their appropriate illustrative flowcharts. Now, look at your study objectives for this unit.

2.0 OBJECTIVES

By the end of this unit, you should be able to:

- Define the three basic structures of structured programming
- Explain the techniques employed in structured programming projects.
- Develop computer programs using any of the patterns of structured programming.

3.0 MAIN CONTENT

3.1 Basic Logical Structures

Before you go into the three types of logical patterns that characterize structured programming you need to understand the benefits of this method of programming. To start with, you need to know that without a standardized method of solving a problem, a programmer may spend more time than expected in finding an appropriate solution method and in the development of the associated program. Structured programming gives room for well-thought-out program logic and provides an attempt to keep programs as simple and straight forward as possible.

Basically, structured programming employs Modular Approach and Top-Down Principle, the concepts which you have come across in the previous units. Now, the fundamental objectives of structured programming are as follows:

- To increase programmer productivity
- To increase program clarity by reducing complexity
- To reduce program testing time
- To reduce program maintenance time and effort.

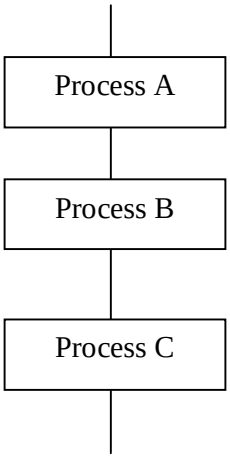
Here is an essential feature of structured programming: It is a "GO-TO-less" programming technique, meaning that the BRANCH pattern characterized by GO TO statements is highly discouraged in structured programming. This is because a GO TO statement causes an unconditional branch from one part of the program to another and an excessive use of such statements leads to continuous changes in the program execution flow. Moreover, a program with many GOTO statements is very difficult to modify by programmers.

Having gone through the above preliminaries but important aspects of structured programming you will now see the three basic logical patterns that characterize structured programming. They are as follows:

- Sequence Structure
- Selection Structure
- Iterative Structure.

Sequence Structure

In sequential structure or what you can also call sequential logic, the steps are executed one after another as serial operations. This can be illustrated by the following flowchart:



In fact, most processes, even of very complex **problems** will usually follow this elementary sequence structure or logic.

Looking at the above flowchart, there are two **arrows, one** at the top leading to the first process box and one at the bottom **leading** out from the last process box. They respectively represent the **ENTRY and** EXIT points of the program segment. It is good to state **here** that a• basic guideline of structured programming is that each module should have only one entry point and one exit point. For this reason, a program that has only one entry and one exit is called a PROPER program.

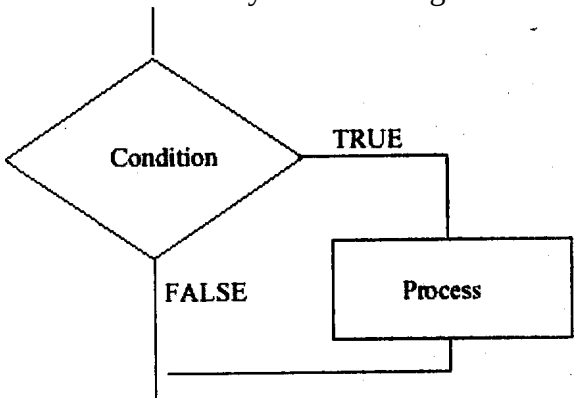
Selection Structure

The selection structure uses conditions and based **on** the decision taken by the computer after comparison **of data, one of** the available alternatives is selected. For this reason, it is **also called** Alternation structure or Conditional structure. You **can also call it the IF** structure. The selection structures can be categorized **as follows:**

- Single Alternation
- Double Alternation
- Multiple Alternation.

Single Alternation

This is described by the following flowchart:



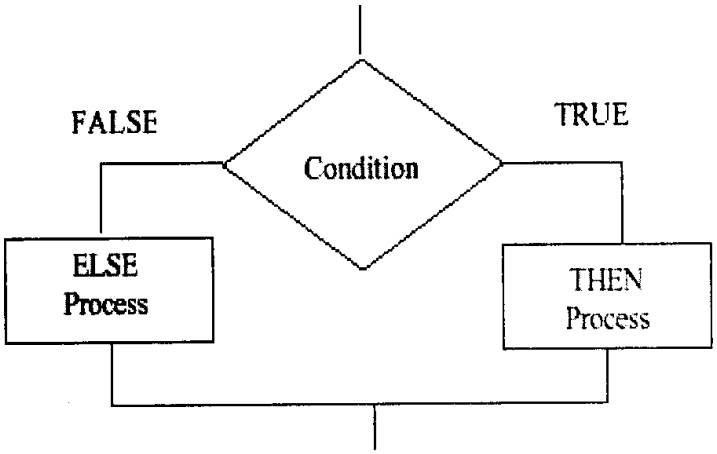
In the above flowchart, if the conditio is satisfied, then the instruction(s) in the Process box (or what is also called ACTION BLOCK) is/are executed. If the condition is not satisfied, the control transfers to the next instruction following the checking of the condition.

The pseudocode is as follows:

IF *condition* THEN
action-block statement(s) ENDIF

Double Alternation

The structure is illustrated below:



You will do the following exercise now.

SELF ASSESSMENT EXERCISE 1

i. Construct the pseudocode for the above structure.

Answer

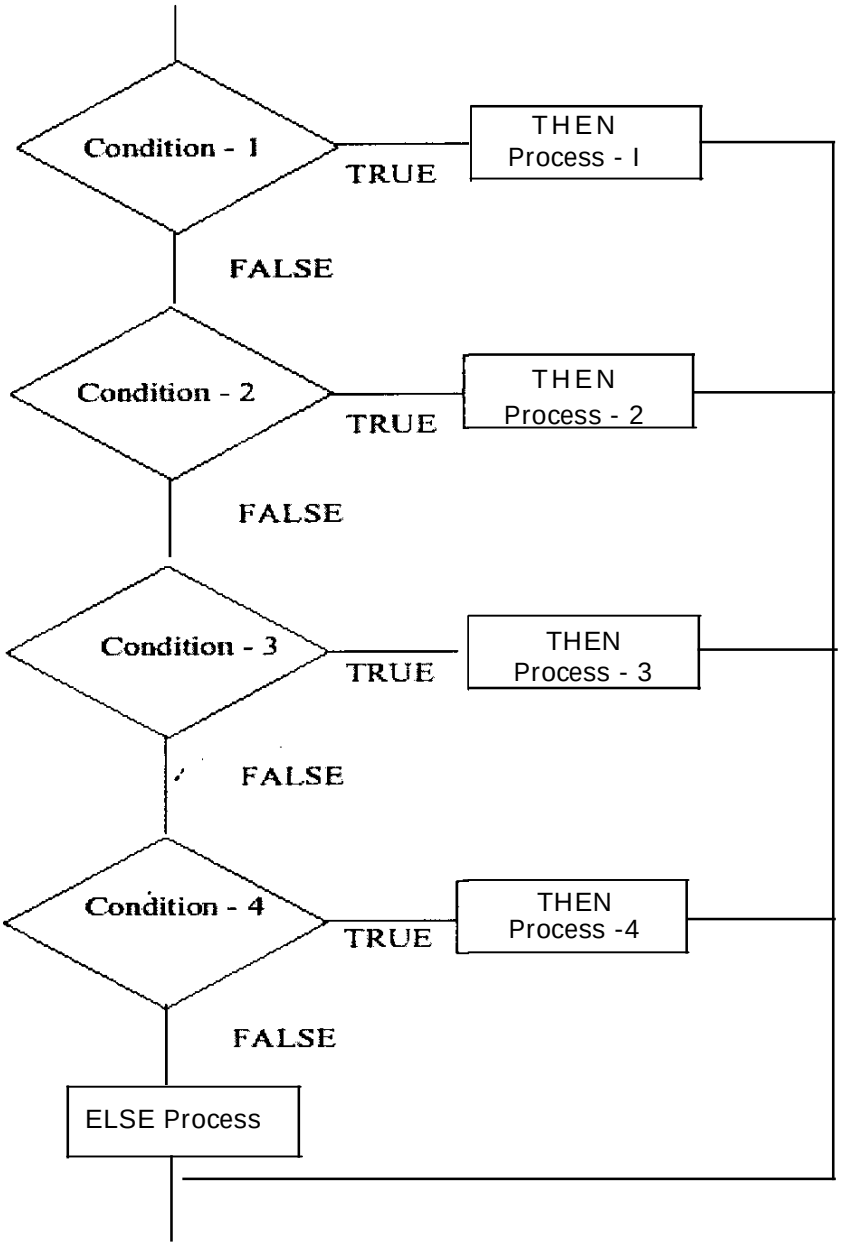
```
IF condition THEN
        Action — block - 1
ELSE
        Action block — 2
ENDIF
```

Where action-block-1 constitutes the THEN Process and action-block-2 represents the ELSE Process.

As you can see in the above flowchart, if the condition is satisfied, then the instruction(s) in the THEN process box is/are executed, otherwise the instruction(s) in the ELSE process box is/are executed.

Multiple Alternation

The structure is seen in the following flowchart using three conditions:



Looking at the above flowchart, you will see that if none of the four conditions is satisfied, then the process in the ELSE box will be executed.

You will now study the third structure below

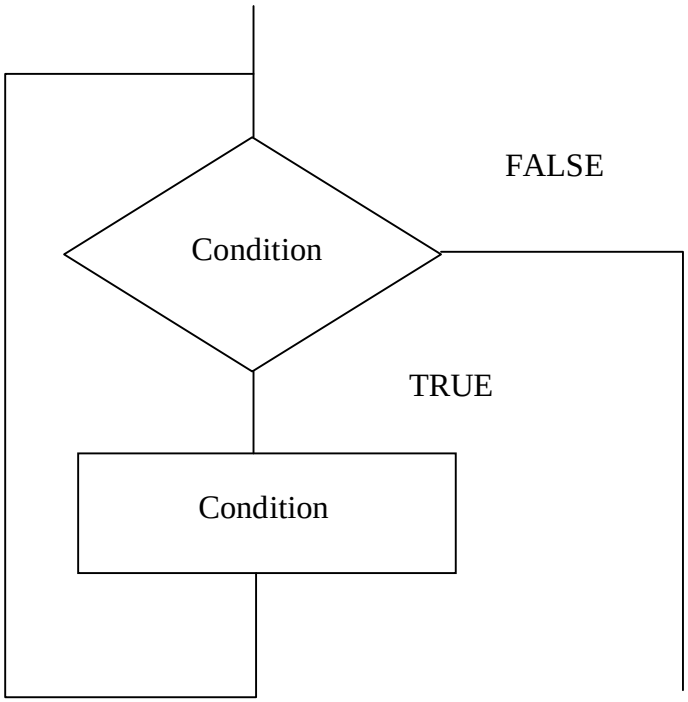
Iterative Structure

Iterative Structure is of the following two forms generally

- WHILE Loop
- DO-UNTIL Loop.

WHILE Loop

To start with, an iterative structure is also called the Loop or Repetition structure. Now the WHILE loop which you can also call the DO-WHILE structure is illustrated below:



A pseudocode for the above flowchart is as follows:

WHILE *condition* DO
statement(s)
ENDWHILE.

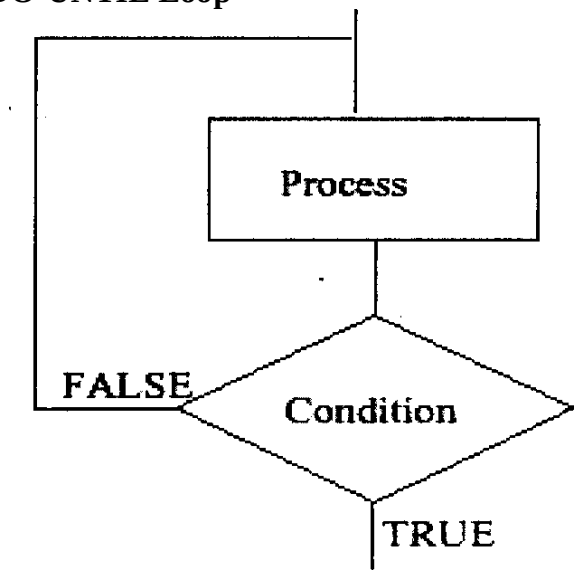
When using the WHILE loop, the following points are essential:

- Have an instruction that initializes the condition before the use of the WHILE structure.
- Let there be an instruction that modified the condition within the loop. Without such a statement, the loop will repeat itself indefinitely.

As you have seen in the above flowchart, the "condition" is used to control the loop. Thus the loop is executed until the condition no longer remains true.

Now, see the other type of loop structure in the following illustrative flowchart.

DO-UNTIL Loop



You can also call the above structure the REPEAT-UNTIL structure. A Pseudocode for this logical structure is therefore as follows:

REPEAT
Statement(s) UNTIL
Condition

SELF ASSESSMENT EXERCISE 2

Construct a flowchart for the Pseudocode below.

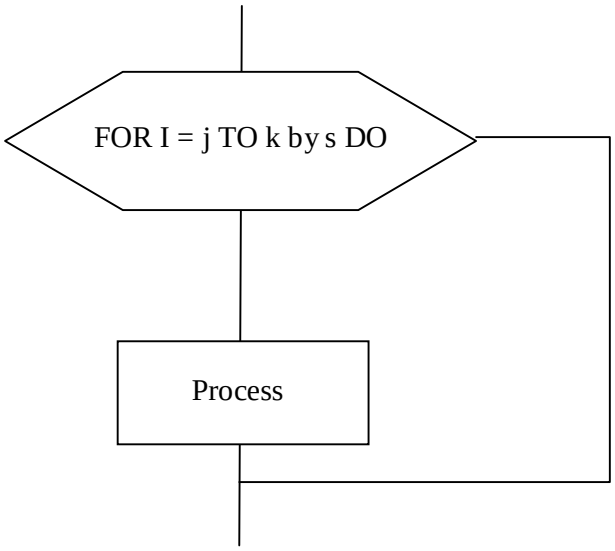
DO
Statement(s)
WHILE *condition*

Answer

The above illustrated structure is of course also a DO-WHILE structure. Do you see that? Before, you end the description of the Loop structure, there is another special type of loop which you can simply call **the FOR** Loop.

For Loop

This is illustrated below, using the following flowchart:



A Pseudocode for the above type of loop is as follows:

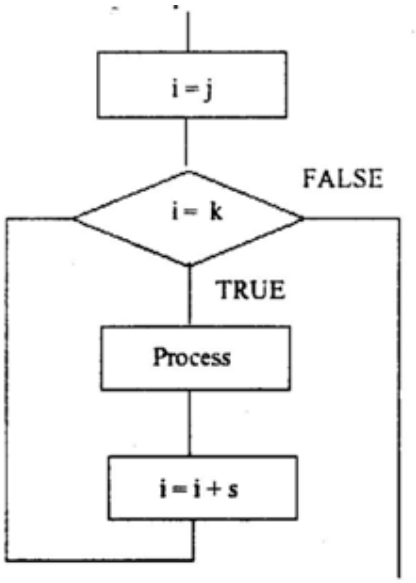
FOR I = j TO by s DO
Statement(s)
ENDFOR.

In the above pseudocode, the variables are defined as follows:

i	.	index variable
J	.	initial value
k	.	final value
s	.	step size.

Essentially, the index variable i controls the loop while **the step size can** either be negative (to implement decrement) or positive **(to implement** increment). When the step size is not stated, the default step size is **taken** to be 1.

For example, a flowchart for positive step size is as follows:



With the above you will now go to the next section before you round up this unit.

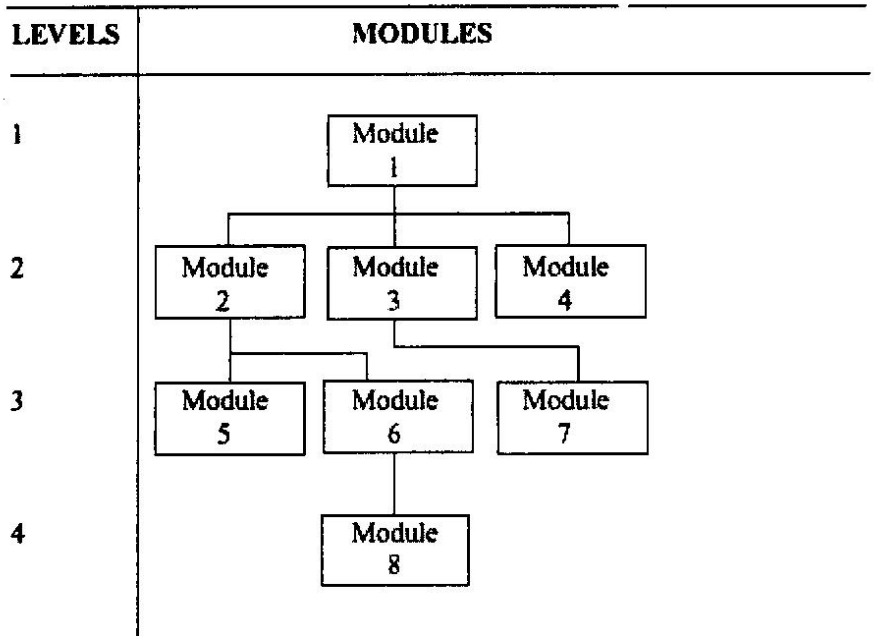
3.2 Structured Programming Methods

In this section, you will be introduced to the two methods employed in developing a structured program. The two methods are:

- Level programming
- Path programming.

Level Programming
Programming by level is also called Top-down programming technique and it is characterized by writing all the program modules on level one or top-most level before programming the modules on level two. In the same vein, the modules on level two are programmed before the modules on level three, and so on.

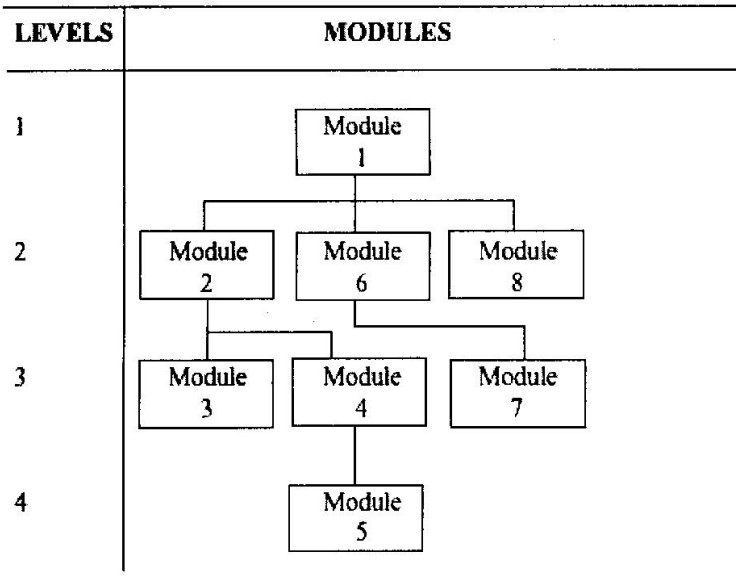
An illustration is as follows:



Then, look at the second technique illustrated below:

Path Programming

This is a method where all modules along a logical path in the program are developed in sequence. See the illustration below:



The Path Programming is also called Backtracking programming. The method helps the programmer to backtracks to the first unwritten module that is directly connected to the path being developed, until all modules are developed.

Before you round up this unit, remember you have been taught in the unit that in structured programming, each module should have only one entry point and one exit point. Do you remember what we call a program characterized by one entry point and one exit point? It is called a Proper Program. The modules are usually linked together by DRIVER PROGRAMS. A driver program simply directs the computer to ENTER the appropriate module, and when the computer EXITS the module, control is returned to the driver program. You will now round up the unit.

4.0 CONCLUSION

In this unit, you have been introduced to the basic logical structures that characterize structured programming. A very major objective of structured programming as you have learnt in this unit is to increase program clarity by reducing complexity. This is the fundamental principle behind the use of modular approach and top-down technique in structured programming.

The three basic control structure, c: patterns employed in structured programming are Sequence, Selection and Iterative structures. The illustrative flowcharts for these patterns were presented in this unit.

5.0 SUMMARY

This unit has basically focused on the various types of logical structures you need to employ to develop structured programs. Very much related to the subject of structured program Hug are the two well-known programming techniques called Level Programming and Path Programming. These two methods were illustrated in this unit. Based on what you have learnt in this unit, some programming languages will be introduced to you later in this course with some typical examples that employ the principles already learnt in this unit.

6.0 TUTOR-MARKED ASSIGNMENTS

1. Why is structured programming called the GO-TO-LESS Programming?
2. Among the three types of patterns that characterize structured programming, identify the structure that seems to be inherent in the other two.

3. (a) Construct a flowchart for a FOR-Loop when the step size is negative.

   (b) Which of the following types of loop structures a FOR-Loop is a special case?

   - WHILE Loop
   - DO-UNTIL Loop.

7.0 REFERENCES/FURTHER READINGS

Brightman, R.W. and Dimsdale, J.M., Using Computers in an Information Age, Delmar Publishers Inc., 1986.

Mandell, S.L., Computers and Data Processing, West Publishing Company, 1985.

Salaria, R.S., Computer Oriented Numerical Methods, Khanna Book Publishing Company, Delhi, 1999.

MODULE 2

UNIT 1 INTRODUCTION TO BASIC PROGRAMMING

CONTENTS

1.0 INTRODUCTION

This Unit is aimed at introducing you to the rudiments of BASIC language. As you already know in the previous unit, BASIC stands for *Beginner's All-purpose Symbolic Instruction Code.* The language was developed in the early 1960s by John Kemeny and Thomas Kurtz of Dartmouth College, as a teaching language. There are many versions of the language. Examples are:

i. BASICA - Advanced BASIC
GW BASIC - Eagle BASIC QBASIC - Quick BASIC
iv. Turbo BASIC
v. Visual BASIC
vi. BASIC 4GL
vii. KBASIC
viii. FreeBASIC
ix. SpeedBASIC
x. ExtremeBASIC
xi. MediaBAS1C
xii. YaBAS1C
xiii. xBASIC
xiv. Liberty BASIC
xv. Just BASIC
xvi. wxBASIC

xvii. smallBASIC
xviii. Gambas

Versions like BASICA and GW BASIC are already taken over by other versions listed above. Visual BASIC is a version of the language specially designed for Windows platforms and is today one of the programming tools with very high preference among programmers. However, it is interesting to know that BASIC language remains one of the best programming tools gaining wide acceptance even in the face of rapid developments of various computing platforms.

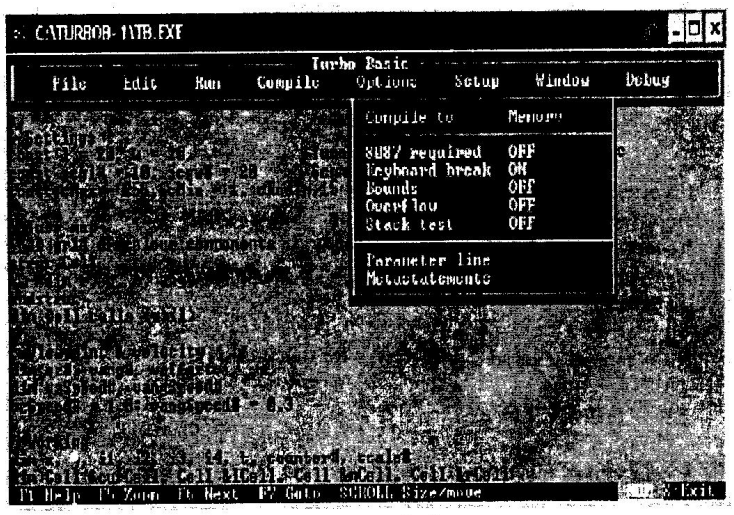
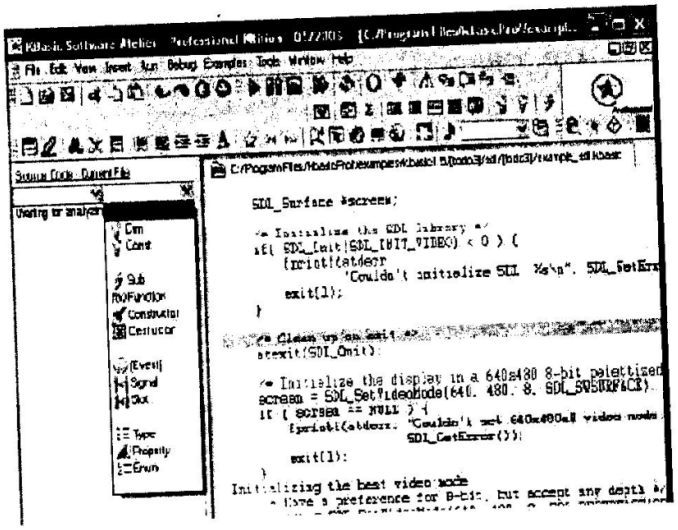


Figure 1: Turbo Basic Screen

For example, SmallBASIC has been designed to run on various platforms including Windows, Linux and Pocket PC platforms. Gambas is designed for Linux operating system, while most of the versions listed in (vi) — (xvi) are for Windows platforms and they compete favourably with Visual Basic. Specifically, SpeedBASIC is an attempt to create an object oriented BASIC style language, similar to Visual Basic, with its own Integrated Development Environment (IDE). The Basic source code is converted to C++ while a compiler is used to compile and link the C++ source with libraries handling Graphic User Interface (GUI), File I/O and etc. For example, below, is a screen shot of KBASIC Professional, Version 1.4, released May 2006, to show you that BASIC has worn a beautiful object oriented look and it's yet reigning alongside other programming languages today. II



In brief, this unit will concentrate on the variables and characters employed in BASIC programming. Since every language has its Reserved Words, this unit will also introduce you to some BASIC reserved words that are common with most BASIC interpreters and compilers.

Now, look at your study objectives for this unit.

2.0 OBJECTIVES

By the end of this unit, you should be able to:

- Define the types of variables used in BASIC language statements.
- Describe the use of the major characters and symbols used in BASIC
- Explain the meanings of some BASIC reserved words.

3.0 MAIN CONTENT

3.1 Basic Variables and Characters

The BASIC Interpreter or Compiler (since you can compile your BASIC program with Turbo BASIC, KBASIC, MediaBASIC and others, for Example) handles variables as names of memory locations. Memory can just be visualised as a litn2.e Post Office with its Post Office boxes as memory locations or cells. You know that each Post Office box has an address to identify it. The contents of the box can change from time to time but the address remains uniquely unchanged. The same occurs in the computer memory. Now, in addition to identifying memory locations, variable names are also used to identify the types of data being stored in the computer memory. Though there are

several types of data, in this unit on BASIC, you will only be introduced to two, and hence two types of variables, namely:

- Numeric Variables
- String Variables

NUMERIC VARIABLES

Generally in BASIC, variable names must start with a letter and can be followed with numbers or other letters, For example, a variable name can be as long as 40 characters (as in MS QBasic). So, examples of variables names are Al, B3, AB4, JONH, REJ2, Y5, and so on.

Numeric variables are simply those that represent numbers. In BASIC, such numbers can be whole numbers, decimals, zero, positive or negative integers. In other several programming languages, you may need to identify specifically the type of the variable, either as REAL, INTEGER, LOGICAL, COMPLEX or CHARACTER. However, BASIC is a more accommodating language and most versions may not bother you on identifying the variable type.

STRING VARIABLES

Variable names are also employed to represent character data in memGry. Character variable names are similar to numeric variable names, but they refer to memory locations containing *character strings,* where collections of characters are called *String Variables.* To specifically distinguish a string variable, the dollar symbol $ is used at the end of the variable name. Examples of string variable names are therefore as follows:

A$, JOHN$, B2$, REJ$, etc

It is good for you to also know that the dollar sign $ also appears at the end of almost every BASIC reserved word that deals with strings. Now, while the values of numeric variables are simply numbers, the values of string variables are given as characters enclosed between double quotes, such as "NAME", "OK", etc. the space character can also be made as part of the string.

You may find the following points very helpful as you choose your variable names:

- Keep your variable name as short as possible since you need to type it every time you need it in your program code (though you can copy and paste, every time).
- Select a meaningful variable name to assist you in remembering what it represents.

SELF ASSESSMENT EXERCISE 1

Identify the following variables as acceptable or unacceptable, giving your reason(s) if unacceptable in BASIC.

i. ADA
ii. $x
iii. TAX
iv. 8BIG
v. W.3

Answers

i. Acceptable
ii. Unacceptable — the dollar sign ($) must be placed after the name.
iii. Acceptable
iv. Unacceptable — a letter must begin the name and not a number.
v. Unacceptable (for common versions) — only letters and numbers are allowed in the name.

Below, you will now see the meanings of some characters or symbols used in BASIC language.

The characters will be grouped for your understanding as follows:

- Alphabetic characters
- Numeric characters
- Data — type Suffixes
- Mathematical Operators
- Special or other characters

ALPHABETIC CHARACTERS

a — z or A — Z

That is. BASIC accepts letters A to Z, either in lower or upper cases.

NUMERIC CHARACTERS

0 — 9 and A — F (or a —1) for hexadecimal numbers

DATA-TYPE SUFFIXES

$ String
! Single Precision
Double Precision
% Integer

& Long Integer

MATHEMATICAL OPERATORS

+ Addition
* Multiplication
- Subtraction
= Relational symbol or Assignment symbol
< Less than
> Greater than
. Decimal point
∧ Exponentiation
<> Not equal to
/ Division
<= Less than or equal to
>= Greater than or equal to
\ Integer division

Special car Other Characters

: - (Colon) separates multiple statements on a single line
; - (Semicolon), controls PRINT statement output
, - (Comma) also controls PRINT statement output
‘ - (Single Quote) used for comment line in place of REM.
? - Used in place of PRINT. BASIC also uses it as INPUT statement prompt

You will see the use of some of the above symbols and characters in subsequent units on BASIC in this course. In the next section, you will now be introduced to some keywords or reserved words in BASIC

3.2 Reserved Words Or Keywords In Basic

Just as words have their meanings in the natural language, the same thing applies in programming languages, reserved words generally describe the operations to be performed by the computer. If your reserved words are wrongly coded, you will definitely receive syntax error message during the running of the program. As a vital programming rule, it is essential to avoid using any of the reserved words as a variable name to avoid program errors during execution. Now, for your understanding, the BASIC reserved words are grouped below according to their programming tasks.

No	Programming Task	Reserved Words
1	Mathematical Computations (or Library Functions)	ABS, ASC, ATN, CDBL, CINT, CLNG, COS, CSNG, CVDMBF, CVSMBF, EXP, INT, LOG, RANDOMIZE, RND, SGN, SIN, SQR, TAN, TIME $ (Function).
2	Control of Program Flow	DO ... LOOP, END, EXIT, FOR... NEXT, IF... THEN...., GOSUB..., RETURN, SELECT, CASE, STOP
3	Declaration of Constant and Variable	CONST, DATA, ERASE, DIM, OPTION BASE, READ, REDIM, REM, RESTORE, SWAP, TYPE...
4	Definition and Call of BASIC procedures	CALL, DECLARE, EXIT, FUNCTION, RUN, SHELL, SHARED, STATIC, SUB
5	Display of Graphic Images	CIRCLES, COLOR, GET(Graphics),LINE, PAINT, PALETTE, PCOPY, PMAP, POINT, PRESET, PSET, PUT(Graphics), SCREEN(Statement), VIEW, WINDOW
6	Management of Memory	CLEAR, FRE, PEEK, POKE
7	DOS File system commands	CHDIR, KILL, MKDIR, NAME, RMDIR
8	Device Input/ Output	CLS, CSRLIN, INKEY&, INP, INPUT, KEY, (assignment), LINE INPUT, LOCATE, LPOS, LPRINT, LPRINT USING, OPEN, COM, OUT, POS, PRINT, PRINT USING, SPC, SCREEN(function), TAB, VIEW, PRINT, WAIT, WIDTH
9	File Input/Output	CLOSE, EOF, FILEATTR, FREEFILE, GET,(File I/O), INPUT, INPUT$, LINE INPUT, LOC, LOCK, LOF, OPEN, PUT(File I/O), SEEK(Function), SEEK(Statement), UNLOCK, WRITE
10	Manipulation of Strings	ASC, CHR$, HEX$, INSTR, LCASE$, LEFT$, LEN, LSET, LTRIM$, MID$ (Function), MID$(Statement), OCT$, RIGHT$, RSET, RTRIM$,SPACE$, STR$, STRING$, UCASE$, VAL.
11	Setting Traps for Events and Errors	COM, ERDEV, ERDEV$, ERL, ERR, ERROR, KEY(Event Trap), ON COM, ON ERROR, ON KEY, ON PEN, ON PLAY, ON STRIG, ON TIMER, PEN, PLAY(Event), RESUME, RETURN, STRIG, TIMER (Function), TIMER(Statement)

What a long list! Yes, but you may not use most of the above keywords in your programming lifetime!. Some programmers don't even know that some of the above keywords exist in BASIC since their program demands do not necessitate their use.

The above list of keywords simply shows you that BASIC is a very rich language in terms of programming tools. Getting grounded in BASIC will help you a lot in studying other object oriented versions BASIC such as Visual Basic.

Some of the reserved words as seen above appear under more than one programming task. In the subsequent units, you will see the use of some of the commonly used keywords in some examples.

4.0 CONCLUSION

This unit has introduced you to the two common types of variables used in BASIC programming, namely:

- Numeric Variable
- String Variable

The unit also shows you the various types of characters generally

employed in BASIC. As in natural language and in other programming languages, there are specific meanings to keywords in BASIC and the unit has provided you a very extensive list of reserved words used in BASIC.

5.0 SUMMARY

BASIC language is still today a good language with its wide range of programming tools for the development of a wide range of applications. It's basic knowledge is very essential for programmers who want to quickly understand for example other object oriented versions of BASIC such as Visual Basic, KBASIC, etc.

What you have learnt in this unit on BASIC will be employed in various examples you will come across in subsequent units on the language in this course.

6.0 TUTOR — MARKED ASSIGNMENTS

1. State four (4) types of variables including the two types commonly associated with BASIC programming.

2. Using the appropriate symbols, write the following expressions in BASIC code:

   (a) $\frac{4ac}{b^2} + 6bc^3 \in \frac{b^2}{2}$

   (b) $\frac{4ac}{b^2} + 6bc^3 \in \frac{b^2}{2}$

3. State three reserved words in BASIC that can be used under more than one programming task.

7.0 REFERENCES/FURTHER READINGS

Brightman, R. W. and Dimsdale, J. M; Using computers in an Information Age, Delmar Publishers Inc; 1986

Microsoft Corporation, MS DOS Quick Basic Version 1.1, 1992.

KBASIC Software Corporation, KBASIC Professional Edition, 2006

http://www.kbasic.com

http://www.freeprogranmmingresources.com

UNIT 2 STARTING WITH BASIC PROGRAMMING

CONTENTS

1.0 INTRODUCTION

Having been introduced to the common types of variables used in BASIC and a summarized grouping of its reserved words, this unit is aimed at leading you gently into BASIC programming. The unit will therefore introduce you to the most commonly used statements you can hardly do without in BASIC code.

As already learnt in the last unit, BASIC programming has its peculiar features. For example, you have learnt that a variable name can be as long as forty (40) characters in BASIC programming. That simply suggests to you that BASIC Programming language has its own environment bounds. This unit will therefore also introduce you to the limits that characterize BASIC Programming environment before you go into in-depth programming using the language.

2.0 OBJECTIVES

By the end of this unit, you should be able to:

- Start up a BASIC programming environment.
- Use the common statements in BASIC programming code.
- State the environment limits that characterize BASIC programming.

3.0 MAIN CONTENT

3.1 Commonly Used Basic Statements

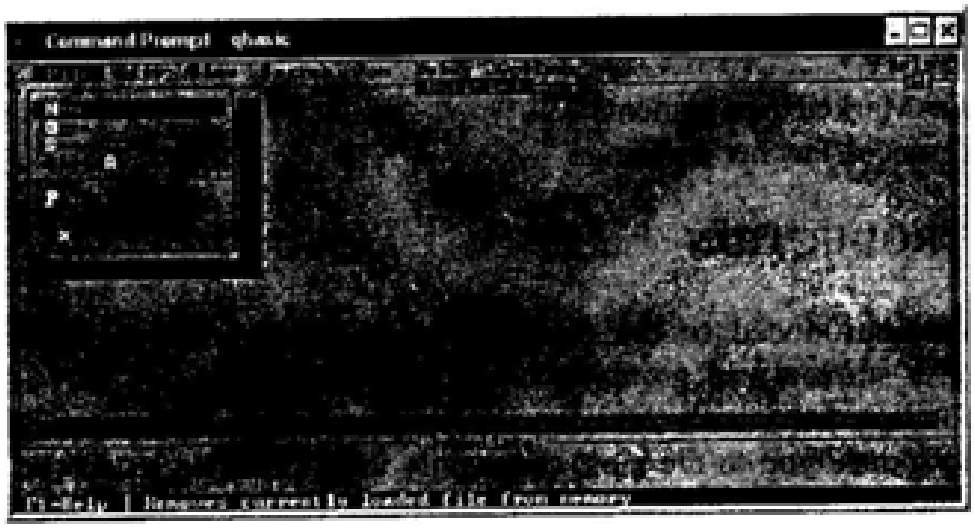
Before introducing you to the common statements in BASIC programming code, you need to see the programming environments for the following commonly available versions of BASIC Interpreter.

- Microsoft Quick BASIC
- Turbo Basic.

Microsoft QBASIC Interpreter

Below is the Quick BASIC Interpreter window environment:

Figure 1



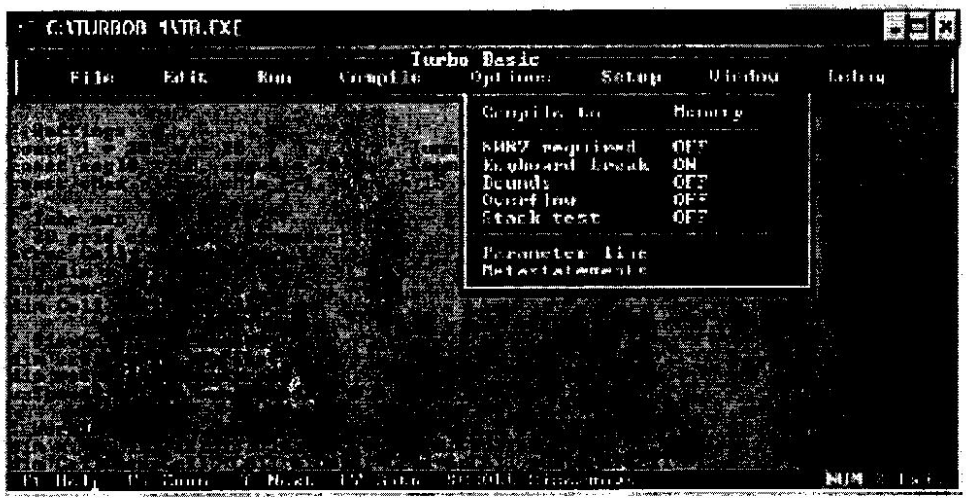
The Quick BASIC program is a DOS program usually shipped with MS-DOS operating system. Thus the program is readily available on any computer system running on fully installed MS-DOS Version 5 for example.

As you can see in the above figure, the Interpreter is a menu-driven environment where you can type your BASIC code and run the program immediately. With this version of BASIC coding environment, you cannot compile your program into an executable form as you can do in Turbo Basic environment shown below.

Turbo Basic Interpreter

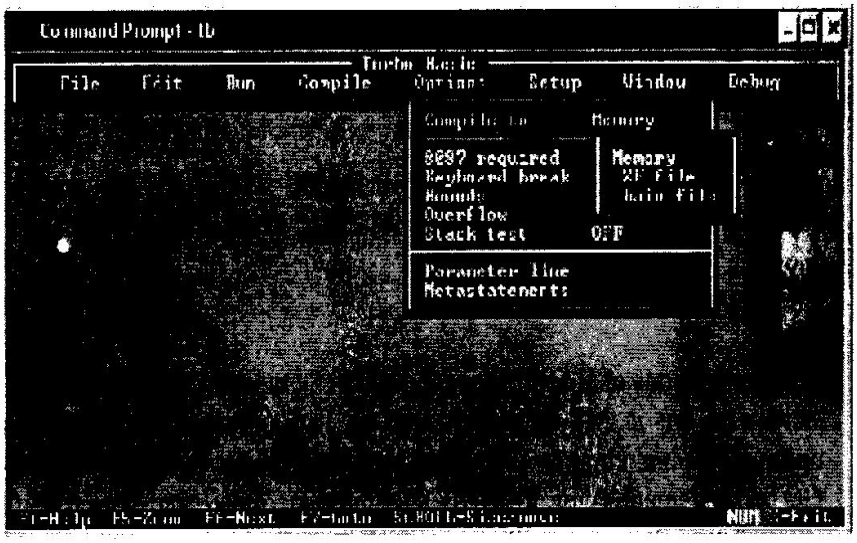
The Turbo Basic Interpreter or Compiler environment is as seen in the following figure (as preizenud in Unit I of this Moduic

Figure 2



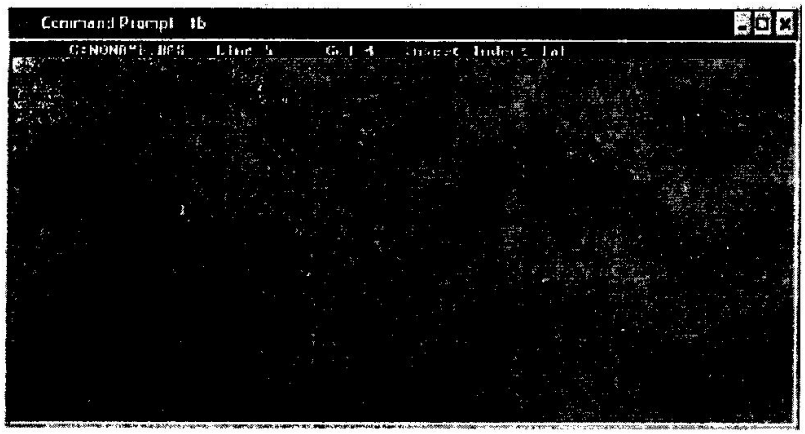
As you can observe from the above figure, the environment is also menu-driven but with available option to compile the source program into an EXEcutable file. This is shown in the next figure below:

Figure 3



The editing window changes to the window when you are ready to type your BASIC source code:

Figure 4



Now, you will be introduced to the following very common statements:

- REM Statement
- Assignment or LET Statement
- INPUT Statement
- PRINT Statement
- GOTO Statement
- IF... THEN... ELSE Statement
- END Statement.

Before you see the forms of the above BASIC statements, it is good to state that a BASIC statement has the following general form:

Line Number *(optional) statement*

In other versions of BASIC like GW-BASIC, you must number every statement, but in MS-QUICKBASIC or Turbo Basic, line number is optional. Line numbers are necessary when your program contains GOTO statement(s) or GOSUB statement(s). Now, get started with BASIC statements stated above.

REM Statement

The *REM* or REMark statement is simply a comment statement that provides information about the program or any of its segments to the programmer. The REM statement provides no information to the computer itself and hence the statement is never executed during the execution of the program.

For example, a REM statement may explain what a variable name represents or what program segment (or module) does. Example of a REM statement is as follows:

10 REM This program finds the average of 'two number

The above code has a REM statement broken into two lines. In line 20, the single quote character is used in place of the reserved word REM. You may have a REM statement without any comment just to create an empty line within the code for good readability. Thus you may have REM statements that look like the following:

40 REM
50 REM **********

Assignment or LET Statement

The assignment or what is also called the LET statement is simply used to assign values to variables. The general format is as follows:

line # (optional) LET *variable = expression*

or

line # (optional) *variable = expression*

where *line #* represents LINE NUMBER.

In BASIC, the equality (=) sign is the assignment symbol. In the above general format, the expression on the right hand side can be any of the following:

- A constant
- An Arithmetic formula
- A variable.

As you can see in the above general format, some versions of BASIC (such as MS-Quick BASIC and Turbo Basic) do not require the reserved word LET in an assignment statement. Hence it is optional. Examples of assignment statements are as follows:

10 X = 25 B$ = "NOUN"
*20 Y=X^2 + 3*X-4*
30 N$ = B$
40 T = Y

SELF ASSESSMENT EXERCISE I

Identify the types of the "expression" in the above assignment statements.

Answer

Line 10:

	25	- Numeric constant
	"NOUN"	- String constant.
Line 20:	X ^{A} 2 + 3*X — 4	- A formula
Line 30:	B$	- String variable
Line 40:	Y	- Numeric variable.

For your observation, line 10 has two statements combined into one line by using the colon (:) symbol as already introduced to you in the last Unit.

INPUT Statement

When a computer program is designed to run many times, each time working with different input data, an INPUT statement is necessary in the code. In using INPUT statement, a single statement can be used for multiple variables in place of multiple INPUT statements. Examples of INPUT statements are as follows:

10 INPUT X1: INPUT A$
20 INPUT X2, X3, B4
30 INPUT "SCORE =", EX

In line 30, a string constant is used in the statement to help the program user see on the screen an information about the value to be entered for variable "EX". That is, when line 30 is executed, you will see something like the following on the screen.

SCORE =?

The question mark (?) simply indicates that the program is requesting for an input from the keyboard.

PRINT Statement

Very central to the output phase of any BASIC program is the PRINT statement. This is used to display the results of computer processing. To send an output to a printer, you use LPRINT instead of the PRINT statement.

The general format is as follows:

Line # (optional) PRINT *expression.*
The expression in the above statement can take the form of the

following:

- Variables
- Arithmetic expressions
- Literals
- Combination of the above three types.

By a literal, we mean an expression that consists of alphabetic, numeric or special characters.
Examples of PRINT statements are as follows:

```
10    PRINT X : PRINT
20    PRINT A, B
30     ? C; D
40    PRINT “VALUE=”, V
50    PRINT 3* A — B
60    PRINT T$
70    PRINT "THE SOLUTION IS"
80    LPRINT W
```

In line 10, the second PRINT statement prints an empty line. This is necessary to provide a good space between output lines. In line 30, the question mark (?) is used in place of the reserved word PRINT. In QuickBasic, the symbol is automatically changed to the word PRINT when you move out of the line to the next. When line 80 is executed, the value of the numeric variable W is sent to the printer.

Now, you will observe that semicolon (;) is used in the PRINT statement instead of the comma (,) symbol. These two characters give different formats to the result output of a PRINT statement. Assuming that the values of A,B,C and D are as follows: A = 2, B = 1, C = 23, D = 0. When line 20 is executed, you will have an output like this:

2 1

while the execution of line 30 will appear like this:

23 0

You will observe that the values of C and D are printed immediately after the other while the values of A and B are printed with spaces between them, specifically, there will be 14 spaces between them. Generally, BASIC divides your output screen into what are called PRINT ZONES. There are five (5) of them, the first starting at column 1, the second at 15, the third at 29, the fourth at 43, the fifth at 57. Using semicolon in your PRINT statement is a way of giving enough

space between the printed values. However, a more flexible way to format your output is to use the TAB function which will be introduced to you in the next unit.

GOTO Statement

Very often, when you use an IF ... THEN statement, one condition causes the computer to execute the next line of BASIC code whit? another condition requires it to execute some code somewhere else if the program. Therefore, in order to move or branch to the code that doesn't follow the IF... THEN statement, you need an unconditional branch statement, which is the GOTO statement. That means that, every time the computer encounters a GOTO statement, it branches to the specified program line irrespective of any condition in the program. As you will remember, you have learnt in this course that structured programming discourages GOTO statements.

Now, an example of GOTO statement is seen in the following code:

```
 10    CLS : INPUT A
 20     IF A <3 THEN GOTO 40
30     ? : PRINT
40     END
```

IF ... THEN... ELSE Statement

This statement works just like IF ... THEN statement used in the above BASIC code, except that the ELSE part is executed if the condition is not satisfied.

An example is as follows:

```
 10    CLS
20     INPUT X
30     IF X> 0 THEN PRINT "POSITIVE" ELSE? X
40     END
```

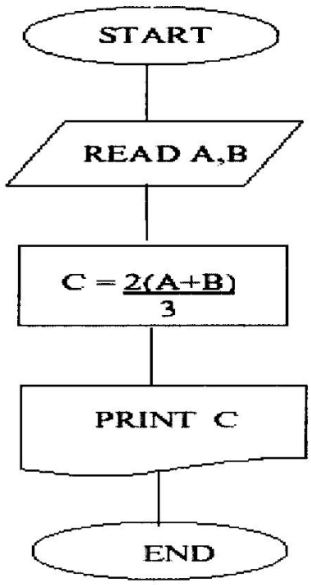
In the above code CLS statement simply clears the screen.

END Statement

Every program has a terminal point. In BASIC, the END statement signifies the end of the program as used in the above immediate two BASIC codes. Since line numbers are optional in some BASIC versions, therefore an END statement can appear anywhere in the BASIC program where the program logically ends. Some Basic versions don't even require an END statement, such as QuickBASIC.

SELF ASSESSMENT EXERCISE 2

Study the following flowchart and provide the equivalent BASIC code:



Answer

The BASIC code is as follows:

```
10    INPUT A,B
20    C = 2 * (A+B))/2
30    LPRINT C
40    END.
```

Now, you will be introduced to the BASIC Programming environment limits.

3.2 BASIC Programming Environment Limits

Below are the programming environment limits that are allowed in BASIC. They are simply the minimum and maximum values or sizes for the following:

- Names
- Strings
- Numbers
- Arrays
- Procedures
- Files

Name, String and Numbers

Descriptions	Minimum	Maximum
Variable Name Length	1 Character	40 Characters
Integers	-32, 768	32, 767
Long Integers	-2, 147, 483, 648	2, 147, 483. 647
String Length	0 Character	32, 767 Characters
Single Precision Numbers: (i) Positive (ii) Negative	 2.80297E-45 -3.402823E+38	 3.402823E+38 -2.802597E-45
Double Precision Numbers: (i) Positive (ii) Negative	 4.94065645841 2465D-324 -1.797693148 6231D+308	 1.7976931348623 1D+308 -4.9406564584 12465D-324

Array Limits

Descriptions	Minimum	Maximum
Array size: (i) Static (ii) Dynamic	 1 Byte -	 65,535 Bytes 65,535 Bytes
Number of Dimensions allowed	1	60
Dimension allowed if unspecified	1	8
Array subscript value	-31,768	32,767

Procedure and File Limits

Descriptions	Minimum	Maximum
Procedure Size	0	64K
Number of Arguments passed	0	60
Data File Numbers	1	255
Data Record Number	1	2,147,483,647
Data Record Size	1 Byte	32K
Date File Size	0	Available Disk Space
Path Names	1 Character	127 Characters

The knowledge of the above limits is very essential as you develop programs using the BASIC language. Some of the BASIC versions mentioned in Unit 1 of this module might have their limits a bit varied from the above. You will now round up this unit.

4.0 CONCLUSION

In this unit, you have been introduced to the common statements you are going to frequently use in BASIC programming. The most common BASIC Interpreters today are the Microsoft QuickBasic and Turbo BASIC Interpreters with the latter having a menu facility to compile your BASIC code into an executable file. Remember that in Unit 1 Module 2, you were introduced to many other versions of BASIC Interpreters and Compilers.

The Unit has introduced to you the various limits that characterize the BASIC programming environment.

5.0 SUMMARY

The unit having shown you some common statements in BASIC programming, you are now ready to get started with programming in BASIC language. You were equally introduced to the programming environments of the most two common BASIC Interpreters, namely, the MS-QUICKBASIC and the Turbo BASIC interpreters.

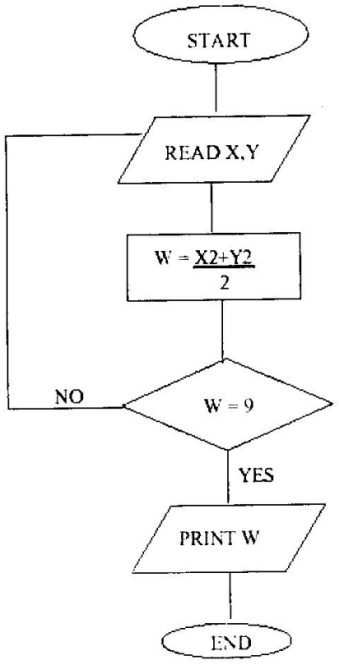
As you have learnt in this unit, there are same programming limits in BASIC associated with the following:

- Names
- Strings
- Numbers
- Arrays
- Procedures
- Files

Knowledge of these limits is very essential for your successful running of your BASIC programs. In the next unit, you will be introduced to more BASIC statements that are not covered in this unit.

6.0 TUTOR-MARKED ASSIGNMENTS

1. Study the following flowchart and write the corresponding BASIC code.



2. (a) Give the 1 — character alternatives you can use in place of
the following statements.

(b)When is GOTO statement needed in BASIC programming,?

3. What do you observe from the programming environment limits for Single and Double Precision Numbers?

7.0 REFERENCES/FURTHER READINGS

Borland International Inc., Turbo Basic Version 1.1, 1987.

Brightman, R.W. and Dimsdate, J.M., Using Computers in an Information Age, Delmar Publishers Inc., 1986.

Mandell, S.L., Computers and Data Processing West Publishing Company, 1985.

Microsoft Corporation, MS-QUICKBASIC 1.1, 1992.

UNIT 3 MORE PROGRAMMING STATEMENTS IN BASIC

CONTENTS

1.0 INTRODUCTION

In the previous two units, you have been introduced to how to get started with BASIC programming. Definitely, as you have seen under the reserved words in BASIC, there is so much to learn about BASIC statements that cannot be covered in this course since you are expected to learn some fundamental concepts in other programming languages as well.

In this unit, you will therefore be introduced to some additional BASIC statements that are frequently encountered, while a good number of other BASIC statements would definitely be studied by you as you make progress in your computer programming courses or as you are confronted with programming problems that might require their use. If you remember what you learnt under Structured Programming, there are basically three types of structures, namely, sequence, selection and looping structures. As you will observe, the first two types of structures have been encountered in the last unit in the use of some of the BASIC statements. For example, IF... THEN... ELSE statement describes a selection structure while the sequence structure is inherent in almost all programming statements.

Now in this unit, you will be introduced to BASIC statements that describe looping structures. However, in the process of introducing one particular BASIC statement, attempt would be made to also introduce you to more statements in order to cover a good number of BASIC statements. The first part of this unit will first of all show you additional method of entering data into BASIC programs.

2.0 OBJECTIVES

By the end of this unit, you will be able to:

- Explain the different methods of entering data into BASIC programs by using different types of BASIC statements.
- Use additional BASIC statements not covered in the last unit.
- Write a BASIC code that performs a repetition, otherwise called looping.

3.0 MAIN CONTENT

3.1 Data Entry Statements

In BASIC, there are three commonly used methods of entering data into BASIC programs. The three methods use the following BASIC statements:

- INPUT Statement
- LET or Assignment Statement
- READ/DATA Statement.

Good enough, you have been introduced to the first two types of statements in the last unit. You will therefore be taught in this unit the use of the READ/DATA statement as another way of entering data into your BASIC program. Before then, see below the general guidelines on using the three statements given above.

- Use the LET or Assignment statement when the data to be used by your program are constant. Moreover, the LET statement is often used to assign a starting value to a variable, such as zero. This is normally called an INITIALIZATION. Remember that, you can simply state your assignment without the LET keyword.
- Use the INPUT statement when a Question-and-Answer mode is required by your program. This is the method you also use when input data are likely to change frequently.
- Use READ/DATA statement when you are to enter many data values. This is the same method you use when entering data into arrays.

READ and DATA Statements

The READ and DATA statements always work together because values contained in the DATA statements are usually assigned to variables listed in the READ statements. The general format for the READ and DATA statements is as follows:

Line # READ *variable list*
Line # DATA *data list*

A DATA statement can have one or several data items and each of the data is known as a *Data Element.* DATA statements are not executed but only provide information to the program. Because of this, you can place a DATA statement anywhere in a program with the corresponding READ statement.

Now examples of READ/DATA statements are as follows:

```
10      READ B, C, D$
20      READ X, Y
.
.
.
60      DATA 20, 40.2, "DAY"
 70    DATA 60, 32.
```

Generally, the READ statement tells the computer to search through the BASIC program to locate the first DATA statement. The computer then assigns your data values consecutively to the variables in the READ statement. You will therefore see the reason why the third value in line 60 in the above code is a string constant, because the third variable in line 10 is a string variable.

BASIC maintains what is called *Data Pointer.* The pointer simply keeps track of the next data element to be read. Thus if a READ statement is attempted after the data list has been exhausted, a message is going to alert you that the end of the data list has been reached. The BASIC Interpreter or Compiler takes all the data items in all the DATA statements and forms one combined data list, ordering the DATA statements from lowest line number to the highest and then using the data from left to right.

Now, though you have already been told that DATA statements may appear anywhere in a program, however it is a common programming practice to group them together either at the beginning or the end of a program. Such a practice makes program debugging easier.

Look at the following code:

```
10      CLS
20      READ DS, ER, B$
30      PRINT TAB (2); DS; TAB (4); ER; TAB (8); B$
40      DATA 23, 45.5, BIRTH =
50      END
```

Do you notice any error(s) in the above code? There is no any error at all. The interesting thing about DATA statement is that it is the only place where character strings do not have to be enclosed within quotation marks as used in line 40 above for the string variable (B$) value.

In the above code, attempt has been deliberately made to introduce you to the TAB function which is normally used with PRINT statement to provide flexibility in your output format. The general format for the TAB function statement is as follows:

TAB *(expression)*

where the expression in the parentheses may be a numeric constant, numeric variable or arithmetic expression, to tell the computer the column at which printing is to start. For example in the above code, the computer is to tab to column 2 before printing the value for variable DS. The semi colon after the TAB function simply instructs the computer to start printing at the next column, i.e. column 3.

SELF ASSESSMENT EXERCISE 1

What happens if comma (,) is used in place of semicolon (;) after the TAB function?

Answer

The printing will be done in the pre-defined Print Zones, ignoring the columns specified in the TAB function expression.

3.2 Using Loops in BASIC

There are programmable problems that require execution of the same instructions a number of times. For example, a payroll program requires calculation of some pay values repeatedly for every employee until the employee list is exhausted. The BASIC statements to execute such repetitions are the FOR and NEXT statements.

The general format for the statements are as follows:

line # (optional) FOR *loop variable initial value* TO *terminal value* STEP *step value*
.
.
.# (optional) NEXT *loop variable.*

In the above general format, the loop variable is also called the INDEX variable. The STEP statement is optional but only necessary when step

value is different from the default value **1.** As you have already learnt in unit 5, the step size can indicate increment or decrement.
An example of FOR ... NEXT statement is seen below:

```
10    PRINT
20    FOR X = 1 TO 3 STEP 0.5
30    Y = SQR (X)
40    W = INT (X)
50    PRINT Y, W
60    NEXT X
70    END
```

The output of the above program is as follows (using QuickBASIC Interpreter):

1	*1*	*1*
1.5	*1.224745*	*1*
2	*1.414214*	*2*
2.5	*1.581139*	*2*
3	*1.732051*	*3*

Again, the above program has included two BASIC Library functions:

- SQR — Square Root Function
- INT — Integer Function

where the Integer function returns the number if it is an integer or the next lower integer, if otherwise.

SELF ASSESSMENT EXERCISE 2

What other expression can you use in place of the SQR function?

Answer
SQR(X) = X ^ 0.5

Nested FOR and NEXT Statements

There are cases when you can have nested loops. That is, all of one loop can be within another loop. See an example below:

```
10      FOR 1 = 1 TO 4
20          FOR J = 1 TO 9
30              PRINT I; J
40          NEXT J
```

50 NEXT 1

As you can observe above, it is customary to indent the inner loop to enhance readability. In the nested FOR and N F XT statements as you have in the above example, always be careful o to mix the FOR from one loop with the NEXT from another. In other words, one loop must be completely inside another. In the above example the J loop is the Inner Loop which is completely inside the I loop which is the Out Loop.

It is interesting for you to also know here that FOR... NEXT loops equally help you to enter successive data items into a program instead of using the INPUT statements.

Now, below are very essential rules you should remember when using FOR... NEXT loops.

i) -The initial value must be less than or equal to the terminal value when using a positive step value, i.e. when implementing an increment.
ii) When implementing a decrement, i.e. when the step value is negative, the initial value must be greater than or equal to the terminal value.
iii) The step value should never be o, since this would result in an endless loop.
iv) Transfer can be made from one statement to another within a loop. However, you cannot make a transfer from a statement within a loop to the FOR statement.
v) The value of the index (or loop) variable should not be modified by program statements within the loop.
vi) The initial, terminal and step expressions can be made up of any numeric variable, constant or arithmetic expression (or formula).
vii) Each FOR statement must have an associated NEXT statement.
viii) FOR and NEXT loops can be nested. However, the NEXT statement of the inner loop must precede the NEXT statement of the outer loop.

SELF ASSESSMENT EXERCISE 3

What are the two basic steps taken by the computer when it encounters a FOR statement?

Answer

These are steps:

- It sets the index variable to the initial value specified.
- The first time the FOR... NEXT loop is executed, the FOR statement tests to see if the index variable value exceeds the

specified terminal value. If the value does not exceed the terminal value, the statements in the loop are executed. Otherwise control is transferred to the statement following the NEXT statement.

Using FOR.. .NEXT Statements in Arrays

It is usually valuable to be able to store many values under the same variable name. Though these data values are not stored in the same memory locations, but they have the same identifier. Such variable are called *Subscripted Variables.* But more precisely, a set of variables with the same name and different subscripts is called an *ARRAY.*

By the above definition, it then means that simple variables are Unsubscripted Variables.

Usually, the BASIC interpreter or compiler is designed to assume that an array will have no more than ten (10) or eleven **(11)** elements or subscripts (depending on the BASIC version). But a programmer can specify the number of elements in an array for which memory space must be reserved, if he needs more than 10. You can do this by using the DIM statement or the DIMension statement, with the general format as follows:

Line # DIM *variable I* (limit 1), *variable 2* (limit 2), *variable* 3 (limit 3), ...

The variables are the array names while each limit is an integer constant that represents the maximum number of memory locations required for a particular array. You don't actually require a DIM statement for arrays of ten or fewer elements.

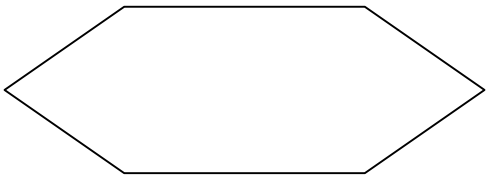
As a programming rule, DIM statements must appear in a program before the first reference to the arrays they describe, and as a good practice, you should place the DIM statements at the beginning of your program.

SELF ASSESSMENT EXERCISE 4

What is the flowchart symbol appropriate for the DIM statement?

Answer

The symbol is the Preparation Symbol:



Examples of DIM statements are as follows:

```
10    DIM A(50), B(2)
20    DIM D$(3), PR(K)
30    DIM ITEM$ (J+K).
```

As you can see above, the subscript of an array variable can be:

- a Number
- a Numeric variable
  c
- an Arithmetic expression.

Now, back to FOR and NEXT statements. They can be an efficient method of reading data into an array. Look at the following example:

```
10    FOR T = 1 TO 9 STEP 2
20          Y (T) T + 3
30          Y (T+1) T * 3
40    NEXT
```

From the above code, you have the following results of processing:

- T is assigned the value of 1 at the beginning.
- (T + 3) or 4 is stored in Y (1) while (T * 3) or 3 is stored in Y (2)
- Then T is stepped to 3 and (3+3) or 6 is stored in Y (3) and (3* 3) or 9 is stored in Y (4).
- The above process continues until T = 9.

Before rounding up this unit, look again at the following example:

```
10    DIM N$ (20)
20    FOR J = 1 TO 3
30          READ P(J)
40          N$(J) = "PRICE"
50         PRINT N$(J); J, P(J)
60    NEXT J : GOSUB 200
70    DATA 2.23, 4.5, 6.3
80    END
```

```
200   REM Below is a SUBROUTINE
210   PRINT "PRICES INSUFFICIENT"
220   PRINT
230   RETURN.
```

The output of the above program is as follows:

```
PRICE 1      2.23
PRICE 2      4.5
PRICE 3      6.3
PRICE INSUFFICIENT
```

In the above program code, you have been introduced again to another statement, namely, the GOSUB statement which is a special case of the GOTO statement. The GOSUB statement is specifically used to transfer control to a program SUBROUTINE, which is a block of code that performs one task each time it is executed. When you divide your BASIC program into sections such as subroutines, you are using a form of structured programming which you already know is called Modular Programming. Usually, you end a subroutine with a RETURN statement while it is a good custom to begin your subroutine block with a REM statement as seen in the above code.

You will now round up this unit.

4.0 CONCLUSION

In this unit, you have been taken through additional BASIC statements such as:

- READ/DATA statements
- FOR.. .NEXT statements
- DIM statements.
- GOSUB...RETURN statements
- Some library function statements.

The unit has explored different ways of using some of the statements to enter data into your program apart from the INPUT and the assignment statements treated in the previous unit.

Specifically, this unit has also introduced you to how to implement loops by using the FOR.. .NEXT statements with the associated rules when using the statements.

5.0 SUMMARY

This unit, apart from introducing you to more BASIC statements has equally treated the fundamental concept of an ARRAY and how it is treated in BASIC. Though the examples shown focused mainly on One-Dimensional arrays, you can equally encounter Two-Dimensional arrays such as matrices in your programming assignments. Two-Dimensional arrays are simply represented as follows:

- AU, .1) - for numeric variables.
- B$(K, S) - for string variables.

You have already learnt in this unit that the DIM statement is normally used to specify the number of array elements. The unit has also introduced you to the GOSUB statement which is a special case of GOTO statement.

With this unit, being the third unit dedicated to BASIC programming, you can now get started with BASIC programming problems while the table of reserved words in unit 6 will help you to develop yourself in BASIC programming beyond what you have covered in this course.

6.0 TUTOR-MARKED ASSIGNMENTS

1. State the various types of BASIC statements you can use to enter data into your program.
2. State an important rule to be followed when using nested loops in BASIC.
3. Identify the error(s) in the following program:

```
10    PRINT
20    FOR I = I TO 4 STEP -2
30        PRINT 2 *1
40        FOR J = I TO 5
50            PRINT J
60        NEXT I
70    NEXT J
```

7.0 REFERENCES/FURTHER READINGS

Brightman, R.W. and Dimsdate, J.M., Using Computers in an Information Age, Delmar Publishers Inc., 1986.

Mandell, S.L., Computers and Data Processing West Publishing Company, 1985.

Microsoft Corporation, MS-QUICKBASIC 1.1, 1992.

UNIT 4 INTRODUCTION TO FORTRAN LANGUAGE

CONTENTS

1.0 INTRODUCTION

In this unit, you are going to be introduced to the fundamentals of Fortran language. Fortran, as you will remember is an acronym for FORmula TRANslator which was developed by John Backus and his team between 1953 and 1957. Since the appearance of its first compiler in 1957, Fortran has undergone various revisions as follows:

- Fortran II - in 1958
- Fortran IV - in 1962
- Fortran 66- in 1966
- Fortran 77 - in 1978
- Fortran 90/95 - in the 90s.

Fortran is the first High-Level Language which was designed principally for scientific and engineering environments. In order to favourably compete with other programming languages which are well structured, an ANSI committee was inaugurated in 1980 to formulate a new standard for Fortran., This led to the Fortran 90 version which incorporates all of Fortran 77 features and can compete well with structured languages such as Pascal and C++. However, presently, Fortran 90 compiler is still uneasy to come by and so, the most commonly used compilers are still for Fortran 77.

To start with, this unit will give you a snapshot of Fortran language features, especially, some rules guiding its program coding before introducing you to descriptions of variables and constants in Fortran. Your study objectives are as follows:

2.0 OBJECTIVES

By the end of this unit, you will be able to:

- Describe the basic rules guiding the coding of Fortran programs.
- Distinguish between the various types of variables and constants in Fortran.

3.0 MAIN CONTENT

3.1 Getting Started with Fortran Programming

Later in this course, you will be introduced to some few features of Fortran 90, but meanwhile, unless otherwise stated, most of the features you will be going through below are Fortran 77 adapted. However, just as you have been told of various versions that exist for BASIC language interpreters, the same applies to Fortran compilers.

Below are some available compilers for Fortran 77 language:

- Microsoft Fortran 77
- FORTE — Fortran Environment
- Fortransoft
- ProFortran
- WATFOR 77— Interpretive Fortran Version.

Each line of Fortran code can be typed using any ASCII (text) editor such as Microsoft Notepad editor. However, the code lines must be entered according to the following rules:

Column 1:	Used for comment line by entering letter C (or symbol * for some compilers).
Columns 2-5:	Statement number or label.
Column 6:	Blank or character for continuation line (e.g. +, $, depending on your compiler).
Columns 7-72:	Fortran statement s.
Columns 73 – 80:	Optional sequence number.

It is good to state here that Fortran 90 allows free form source coding, that is, there is no restriction on where Fortran statements are to be located as specified in the above Fortran 77 rules. Moreover, a comment can be placed at the end of a statement provided an exclamation mark (!) precedes the comment.

Before you get introduced to the Fortran variables and constants, below are the common characters recognized by Fortran language:

- The 26 letters: A — Z (upper or lower cases).
- The 10 digits: 0 — 9
- The 12 symbols: = + - * / ( ) , . $ ‘:

- The Blank character.

Remember that the above list depends on the **Fortran** compilers. You can also see immediately below the operators **employed** by Fortran.

Fortran Operators

The Fortran operators are classified as follows:

- Arithmetic Operators
- Relational Operators
- Logical Operators.

Arithmetic Operators

+	-	Addition
-	-	Subtraction
*	-	Multiplication
/	-	Division
**	-	Exponentiation
=	-	Assignment.

Relational Operators

(Please, take note of the dots (.) before and after the two characters)

.LT. -	Less than (<)
.LE. -	Less than or equal to (=)
.GE. -	Greater than or equal to (=)
.GT.	Greater than (>)
.EQ. -	Equal to (=)
.NE. -	Not equal to (?).

Logical Operators

.AND.	AND (∧)
.OR.	OR (∨)
.NOT.	NOT (~)

3.2 Fortran Variables and Constants

You will start with the variables since a great deal **of** flexibility is gained when you allow a quantity to be referred to by a name rather **than a** constant value. Generally, the major aim of any computation is **to find** unknown values from known ones.

Just as in any other programming languages, variables in Fortran can

be given names consisting of several alphanumeric characters, but for most compilers of Fortran 77, the number of characters can be as many as six, the first being a letter as in BASIC language. In Fortran, there are several types of variables as follows:

- REAL
- NTEGER
- OUBLE PRECISION
- OGICAL
- COMPLEX
- CHARACTER

Among all the types of Fortran variables, the most common two are the REAL and INTEGER types. A very peculiar characteristic of Fortran is that it assumes that the type of a variable is implied by its spelling. The integer and real variables are identified as follows:

REAL : Al, X2, B8, DAY, ...
INTEGER : JOS, 12, K4, MONEY, ...

Precisely, an INTEGER variable starts with any of the characters I, J, K, L, M, N while REAL variable begins with any other character outside I - N (i.e. A — H and O — Z)

For example, the following are not valid variable names in Fortran

4AB	It begins with a number (4)
BX.5	It contains an illegal character (.)
C$	It contains an illegal character ($).

The above method of specifying variable types in Fortran is one of the rigid rules you find in Fortran. Care should always be taken in handling variables in Fortran especially in assignment statements. Mixing of different types of variables in the same expressions can easily lead to errors without conversion functions.

Though Fortran assumes the type of a variable by its first letter, it however gives you the opportunity to change the presumed type by using declaration statements such as follows:

INTEGER X, Y, ZERO
REAL 12, JOS, NUM.

By default, variable name ZERO for example, is a real variable since it starts with Z, but with the above declaration statement, ZERO will be treated as an integer variable in the statements following such

declaration. As a common practice, DECLARATION statements are usually given at the beginning of your program before the first references to the names defined in the declaration statements.

In case you intend to have mixed-type expressions, that is expressions containing different types of variables, you will need to use the Fortran Intrinsic functions such as REAL and NT for real and integer expressions respectively.

See the following examples:

*B = (3/5) * (F — 32)*
*NUM = R *J/3*

The above two Fortran assignment statements will definitely give you errors when executed. The errors can be corrected as seen in the modified expressions below:

*B = (10/5.0) * (F— 32.0)*
*NUM = INT 9R *REAL (J)/3.0).*

From what you can see above, it is always advisable to use decimal points in real numbers to avoid mixed-type errors.

SELF ASSESSMENT EXERCISE 1

Explain the roles of the REAL and NT functions used in the second Fortran statement above.

Answer

The variable NUM is an integer variable name and therefore represents a memory location to be occupied by an integer value. Therefore, the expression on the right hand side should be completely of an integer type.

By writing:

*R *REAL (J)/3.0*

the expression becomes of type real. Finally, by writing:

*INT (R * REAL (J)/3.0)*

the expression on the right hand side can now be assigned to NUM, which is an integer. Another way of carrying out the conversion on the right hand side is by writing the whole assignment as follows:

NUM = INT(R) J/3*

This last form of expression is simpler and may still yield the same result like the other one but INT function is a *Truncator*.

You will now be introduced to the Fortran constants. Just as in variables, the most commonly used constants in Fortran are the real and integer constants.

An *Integer Constant* is a positive, negative or zero whole number without a decimal point or commas. Therefore, Fortran assumes your number having no decimal point as an integer. Examples are 4, 32, 6932, ...

A *Real Constant* on the other hand is a positive, negative or zero number with a decimal point. You can express real constants in

- Scientific Notation — e.g. 2.304E-3, 3.42E+3
- Double Precision Notation — e.g. 2.304D-3, 3.42D+3.

Just as there are several types of variables apart from the real and integer variables, there are also other constants allowed in Fortran in addition to the real and integer constants. For example, in Fortran, you also have the following types of constants (to be treated later in other units):

- Complex Constants
- Logical Constants
- String Constants.

SELF ASSESSMENT EXERCISE 2

Identify the error(s) in each of the following, if any, and attempt correcting them:

i. Fortran Variables: $AB, 62A, BB4, X4+2

ii. Assignments:

a. A = X**2 + J**2

b. MAX3 = J4 — K2

c. NAT = I**2 + 4.5 *J

Answers

i. Variables: The variable names SAB and 62A are not Fortran valid variable names while X4+2 is containing an operator.

ii. Assignments:

a. A = X**2 + REAL (J) **2

b. No any error
c. NAT = TNT (REAL (I) **2 I- 4.5 * REAL (J).

4.0 CONCLUSION

In this unit, you have been taken through the fundamental rules guiding Fortran programming and coding. As you have learnt in this unit there are some rules to be followed while coding your Fortran source programs. For example, column 1 is specially reserved for character "C" to enter your comment. This is analogous to using REM in BASIC language. Your Fortran statements are to be entered between columns 7 and 72. However, the column rules that exist in Fortran 77 are rather non-existent in the Fortran 90 version whose compiler is not yet widely available.

This unit has equally shown you the types of variables and constants employed by Fortran. While emphasis has been laid on the most commonly used types, that is, the real and integer types, you will be introduced to the other types later in the course.

5.0 SUMMARY

As you have learnt in this unit, Fortran is very quick to assume the type of your variable by the first letter of the variable. Every variable name that starts with I, J, K, L, M and N is assumed to be an integer unless you change the presumed type by using the declaration statement keyword REAL or its form of intrinsic function. Starting a variable name with other letters is assumed that the variable is real by Fortran.

Remember that the unit also introduced you to the types of operators used in Fortran language programming. These operators will be used in some examples in the subsequent units. With all that you have learnt in this unit, you are now ready to start using some simple statements in the next unit.

6.0 TUTOR-MARKED ASSIGNMENTS

1. From what you have learnt about Fortran 77 in this unit, state the two main areas of strict compliance with Fortran rules.....
2. State types of variables and constants available in Fortran programming .....
3. Rewrite the following expressions to correct the errors:

(a) J2 = K**2.0 + 3 *M4
(b) REJU = I - 4*B**3
(c) NAME = J/3 + 4 * R.

7.0 REFERENCES/FURTHER READINGS

Fatunla, S.O., Fundamentals of FORTRAN Programming, ADA + JANE Press, 1993.

Monro, D.M., FORTRAN 77, Edward Arnold, 1987.

UNIT 5 FORTRAN KEYWORDS AND LIBRARY FUNCTIONS

CONTENTS

1.0 INTRODUCTION

In this unit, you are going to be introduced to the common reserved words or keywords available in most Fortran compilers. During your study of the BASIC language, you have seen the need of having a good knowledge of a language's keywords since they provide commands to the compiler of the language. The keywords or reserved words also provide the tools needed to solve your programmable problems.

Library functions are very essential in most programming problems since they are the same as quick methods of processing a number of program statements in a single statement. In this unit, you are also going to be introduced to Fortran library functions.

A very simple program statement characterized by simple data entry and output of processing will also be introduced in this unit using the common Fortran statements acceptable by most compilers.

2.0 OBJECTIVES

By the end of this unit, you should be able to:

- Identify Fortran reserved words and library function names.
- Discover the differences in some Fortran keywords and their BASIC language counterparts.
- Write simple Fortran programs using the READ and WRITE statements.
- Describe the various types of specifications used in FORMAT statements.

3.0 MAIN CONTENT

3.1 Fortran Reserved[3] Words and Library Functions

The Fortran reserved words are listed below with the general format of using them, as supported by most Fortran compilers.

Reserved Words	General Format of Usage
ASSIGN	ASSIGN label TO ivar
BACKSPACE	BACKSPACE <clist>
BLOCK DATA	
CALL	CALL name [<arguments>]
CHARACTER	CHARACTER <list>
CLOSE	CLOSE <clist>
COMMON	COMMON [/name/<list>/name/<list>…]
COMPLEX	COMPLEX <list>
COMPLEX	FUNCTION name <[dummy arg]>
CONTINUE	
DATA	DATA <list>/constants/[, <list>/ constants/…]
DIMENSION	DIMENSION <list>
DO	DO label variable = value, value
DOUBLE PRECISION	DOUBLE PRECISION <list>
	DOUBLE PRECISION FUNCTION <[dummy arg]>
ELSE	
ELSE IF	ELSE IF <logical expression> THEN
END	
END FILE	END FILE <clist>
END IF	
ENTRY	ENTRY name [<dummy arg>]
EQUIVALENCE	EQUIVALENCE <list> [, <list>…]
EXTERNAL	EXTERNAL <list>
FORMAT	FORMAT <specification>
FUNCTION	FUNCTION name <[dummy arg]>
GO TO	GO TO label
	GO TO ivar, <label [, label…]>
IF	IF <arithmetic expression> label, label, label
	IF <logical expression> statement
	IF <logical expression> THEN
IMPLICIT	IMPLICIT <list>
INQUIRE	INQUIRE <clist>
INTEGER	INTEGER <list>
	INTEGER FUNCTION name <[dummy arg]>
INTRINSIC	INTRINSIC <list>
LOGICAL	LOGICAL <list>

	LOGICAL FUNCTION name <[dummy arg]>
OPEN	OPEN <clist>
PARAMETER	PARAMETER <name=constant[,name = constant]...>
PAUSE	PAUSE [constant]
PRINT	PRINT fmt [, list]
PROGRAM	_____
READ	READ fmt [, list]
REAL	REAL <list>
	REAL FUNCTION name <[dummy arg]>
RETURN	RETURN [ival]
REWIND	REWIND <list>
SAVE	SAVE [list]
STOP	STOP [constant]
SUBROUTINE	SUBROUTINE name [<dummy arg>]
WRITE	WRITE <list> [list]
	WRITE fmt [, list].

In the above table:

ivar	=	integer variable
ival	=	integer value
fmt	=	format identifier
clist	=	control list
list	=	list of items (rules vary from one compiler to another).
[ ]	=	optional items for common compilers

Below, you are also introduced to the Fortran Generic Intrinsic functions with their meanings, caution should be taken in using Fortran library functions since specific names are given to the functions according to the type of value expected from processing, that is, whether real, integer, complex or double precision value.
Now, study the table below:

Function Name	Meaning	Specific Name	Mapping (Argument ? Result)
ABS (.)	Absolute Value (|.|)	IABS ABS DABS CABS	I? I R? R D? D C? R
CMPLX (.) DBLE (.)	Complex Type Double Precision Type	CMPLX DBLE	I, R, D, C → C, D
INT (.)	Integer Type	INT IFIX IDINT	I, R, D, C → R
REAL (.)	Real Type	REAL, FLOAT	C, D → R
ACOS(.) AINT (.) ANINT (.) ASIN (.) ATAN (.)	Cos^{-1} (.) Truncation Nearest Whole N0. Sin^{-1} (.) Tan^{-1} (.)	ACOS, DCOS AINT, DINT ANINT, DNINT ASIN, DASIN ATAN, DATAN	R? R D? D
ATAN2 (.,.) COSH (.) LOG^{10} (.) SINH (.) TAN (.) TANH (.)	tan^{-1} (%) cosh (.) log_{10} (.) sinh (.) tan (.) tanh (.)	ATAN2, DATAN2 COSH, DCOSH $ALOG^{10}$, $DLOG^{10}$ SINH, DSINH TAN, DTAN TANH, DTANH	R? R D? D
NINT (.)	Nearest Integer	NINT, IDINT	R? D D? D
COS (.) EXP (.) LOG (.) SIN (.) SQRT (.)	Cos (.) $?^{(.)}$ log_e (.) sin (.) $\sqrt{}$ (.)	COS, DCOS, CCOS EXP, DEXP, CEXP ALOG, DLOG, CLOG SIN, DSIN, CSIN SQRT, DSQRT, CSQRT	R? R D? D C V C
CMPLX(.,.) AIMAG (.) CONJG (.) DPROD	Complex Conversion Complex Imaginary Complex Conjugate Double Precision Product	CMPLX AIMAG CONJG DPROD	I,R,D ? C C ? R C ? C R ? D
CHAR (.)	Character Conversion	CHAR	I ? CH
ICHAR (.) LEN (.)	Character Conversion Character Length	ICHAR LEN	CH ? I CH ? I

SIGN (..)	Sign Transfer	ISIGN, SIGN, DSIGN	I ? I R ? R D ? D
MOD (..)	Remainder	MOD, AMOD, DMOD	
DIM (..)	Positive Difference	IDIM, DIM, DDIM	
MAX (.,....,)	Largest Value	MAX1, AMAX1, DMAX1, MAX0, AMAX0	I ? I R ? R D ? D
MIN (.,....,)	Smallest Value	MIN1, AMIN1, DMIN1 MIN0, AMIN0	

In the above table, the fourth column is to guide you on what data type to expect when any of the intrinsic functions is evaluated on different types of arguments. For example, "C ? R" shows that the function ¯ CABS" produces a real number, taking a complex expression or value as the argument. Some of the above functions will be used in some examples in this course.

Now, get started with learning the most frequently used Fortran Statements in the next section.

3.2 **Using READ, WRITE and FORMAT Statements**

One of the most common and simple statements in programming is one that directs you to enter some few data into the program to see some results of processing an assignment statement. This you will see below, using the READ and WRITE statements which are the Fortran standard INPUT and OUTPUT statements. You can also use the PRINT ATiement in place of the WRITE statement.

Now, for, accepting data during the execution of a program, Fortran employs the READ statement in a list directed form as follows:

READ * [, list]

Please, you should remember that the square brackets are not part of Fortran but simply denote an optional item. That is, the comma and the list are optional.

For example. the following statement:

READ*, X, Y, Z

causes the computer to request for real value data inputs from the keyboard, that is, the *STANDARD INPUT DEVICE.* Remember that X, Y. 7 are assumed to represent real numbers by Fortran. So, when entering the values, they must contain decimal points.

Another way of using the READ statement is as follows:

READ(*, *) [, list]

This form of READ statement requests for input data list from the keyboard, giving you no specified data format. A more general form of this variant of READ statement is as follows:

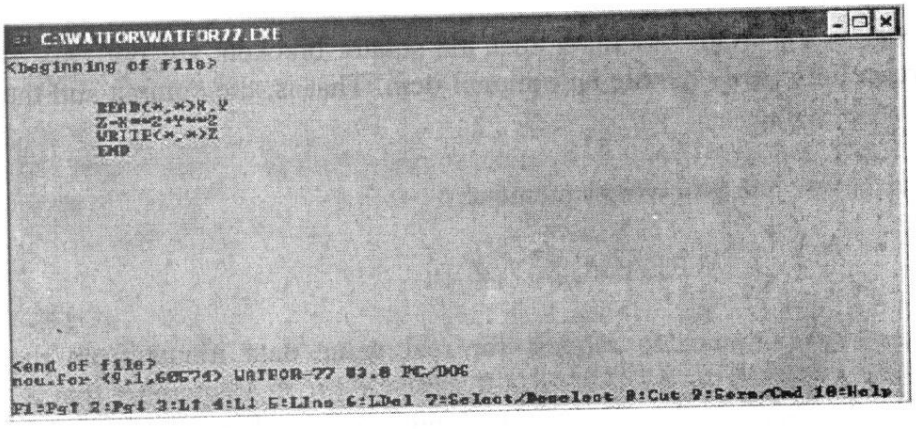
READ (unit, 0 <list>

where 'unit' identifies the INPUT device from which the data list is to be read. The parameter `f" indicates format description which is usually a number that locates the FORMAT statement to be used. The 'unit' parameter is also a number specified in an OPEN statement where the path and the name of the input data file is specified. Before you see some examples, see the general forms of WRITE or PRINT statements also below:

WRITE (unit,f) <list>
WRITE (*,*) [,list]
PRINT*, <list>.

In the above forms of output statements, the `*' parameter denotes the *STANDARD OUTPUT DEVICE,* that is, the MONITOR. This is when 'unit' = `*' in the WRITE statement.

Now, look at the following example as shown in the two figures below, using the WATFOR77 interpretive environment. The WATFOR 77 acts like an Interpreter and helps you to receive immediate results of executing your Fortran program.
Figure 1



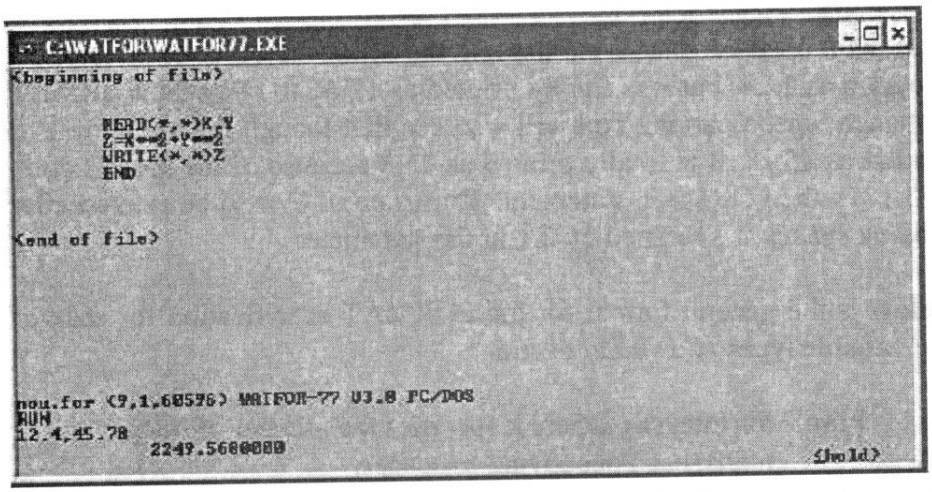
By typing 'run' or 'RUN' at the command prompt, the program gives you an input prompt of blinking cursor to enter your data. In the above

program, two real values are requested whose sum of squares is to be assigned to Z. Entering:

X = 12.4
Y = 45.78

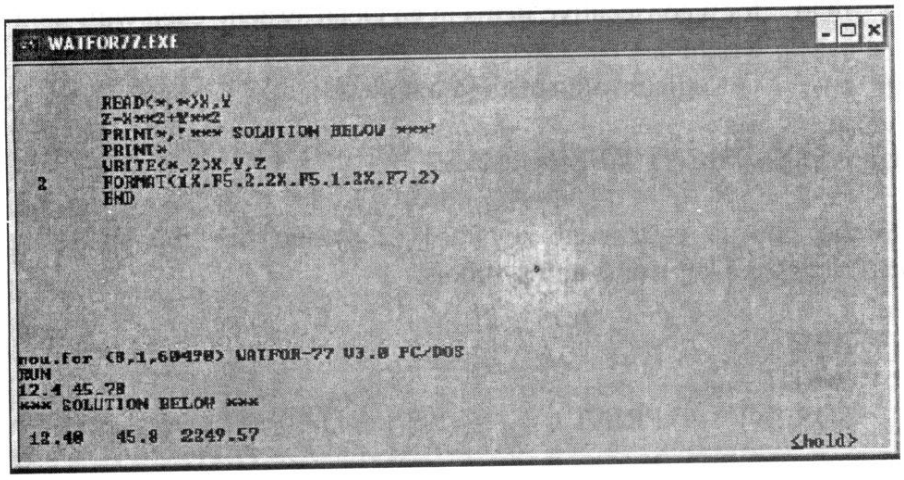
the result yields Z = 2249.5680000

Figure 2



Below, you will now see the use of FORMAT statement, which is usually used to add flexibility to READ and WRITE statements. See this in the Fortran program code below, using the WATFOR 77 program editor and interpreter.

Figure 3



In the above program, There are three output statements: the first prints the string:

`*** SOLUTION BELOW***'

The second prints an empty line while the third prints the values of X, Y and *Z*, using a FORMAT statement in line number 2. The FORMAT statement directs the computer to print 1 blank space before printing the first real or what is also called Floating Point number X. The number should be of length 5 and 2 decimal places. Usually the decimal point is part of the length. The specification is given by "F5.2".

You will observe that even though the value of X is read into the program as 12.4, but it is finally printed as 12.40 to obey the FORMAT statement specification. You will also see that though the value of Y is read in as 45.78, it is finally printed as 45.8 because of the specification "F5.1" in the FORMAT statement. The value of Y is to be printed after 2 blank spaces as specified by 2X in the statement.

Below is the general format for the FORMAT specification for each of the variable types allowed in Fortran:

i) kIm: for Integers, where k specifies the number of integer fields, each occupying m spaces.
ii) kFl.j : for floating point or real numbers, where k specifies how many real numbers in I space each with j decimal places.
IcEl.j: for k real numbers in scientific notation each of length 1 and j decimal places.
iv) kDI.j: for k Double Precision numbers, each of length 1 and j decimal places.
v) kAw: for k characters of length w.
vi) nZd: for n consecutive items in a hexadecimal system, each with field length d
vii) nOd: for n consecutive items in an Octal system, each with field length d.
viii) Lw: for logical value of field length w.

SELF ASSESSMENT EXERCISE 1

State the Fortran equivalent keyword or symbol for the follow BASIC language keywords and symbols:

i. REM
ii. INPUT

iii. SQR (iv) ^ (v) PRINT

Answers

i) C — in the first column
ii) READ
iii) SQRT
iv) **
v) PRINT or WRITE.

Look at the following example before you round up this unit.

```
      INTEGER A, B
C     Program To Calculate Average of Two Numbers
      READ *, A, B
      MEAN = (A +B)/2
      WRITE (*,*)
      PRINT 5, A,B, MEAN
5     FORMAT (11Cl2,13,14)
      END.
```

The above program converts A and B into integer variables from their presumed real type. Here you see that the PRINT statement is used with the FORMAT statement with line label 5. In the FORMAT statement, A is to be printed as an integer of length 2, B of length 3 and MEAN of length 4. To take care of the length specification, Fortran adds blank spaces behind the values to make up for the number of spaces.

You will now round up this unit after this exercise below:

SELF ASSESSMENT EXERCISE 2

Rewrite the FORMAT statement in the above program if A, B and MEAN are to be of the same length 4 when printed.

Answer

```
5     FORMAT (IX, 3I4).
```

This is because there are 3 integers to be printed, each of length 4.

4.0 CONCLUSION

In this unit, you have been introduced to the reserved words in Fortran and the generic intrinsic functions. Just like BASIC language, you need to be acquainted with the Fortran keywords to be able to use the rich resources of its programming features.
And you have seen in the list of library functions, Fortran somehow

"discriminates" between the types of the functions according to their data type. Like in variable names, the first letter of the function name specified the type of value you should expect when using the function. Fortran is very unique in its way of accepting input data and producing the output of results. FORMAT statement is usually employed in conjunction with the READ and WRITE statements to provide flexibility.

5.0 SUMMARY

In this unit, you have really got started with Fortran programming with the introduction to the reserved words and library functions. Get acquainted with the common keywords to start with and you will soon discover that Fortran is not difficult to learn.

In the next unit, you will be introduced to additional Fortran keywords as employed in some examples.

6.0 TUTOR-MARKED ASSIGNMENTS

1. State the differences between the following Fortran reserved words:
   a. COMPLEX and CMPLX
   b. INTEGER and INT
   c. DOUBLE PRECISION and DBLE.
2. a. Classify the following functions into integer, real, complex and double precision types:
      i) DSIN ii) AMOD iii) IDIM iv) CABS
   b. What are the meanings of the above functions in (a)?
3. Write a Fortran program that requests for three real numbers from the keyboard, subtracts 48 from their sum and then divides the result by 6 before sending the output to the screen. In the input statement, let the program use the following specification for the three numbers: length is 8, decimal places are 3. The output should be unformatted.

7.0 REFERENCES/FURTHER READINGS

Fatunla, S.O., Fundamentals of Fortran Programming, ADA + JANE Press, 1993.

Monro, D.M, Fortran 77, Edward Arnold, 1987.

MODULE 3

UNIT 1 USING LOOP AND SELECTION STRUCTURES IN FORTRAN

CONTENTS

1.0 INTRODUCTION

As you will remember under BASIC programming, there is a way of executing repeatedly, a set of program instructions. Just as BASIC has its own form of statements to perform a loop operation, Fortran also has special statements in handling loops in a program.

This unit will therefore introduce you to the **DO ... CONTINUE** statements employed in Fortran for program loops.

The unit will also show you how to handle alternation or selection in Fortran through the use of **IF ... THEN ... ELSE** statements. This will therefore afford you the opportunity of seeing how relational operators are used in Fortran language. In the process of going through some examples in this unit, you will be introduced to some additional Fortran statements. Your study objectives for the unit are as follows.

2.0 OBJECTIVES

By the end of this unit, you should be able to:

- Use the DO statement to process simple program loops.
- Write Fortran programs using the IF ... THEN ... ELSE statements.

- Use some additional Fortran statements to solve some simple problems.

3.0 MAIN CONTENT

3.1 The Fortran DO Loop

Before seeing the practical examples of using the DO statement in Fortran, it is good to emphasize that the loop structure helps to conserve both space and time in program coding when a problem involves repeated execution of a number of program instructions.

In general, a loop segment can be divided into three main sections as follows:

- the loop HEADER
- the BODY of the loop.
- the loop TERMINATOR.

The general form of the DO statement is as follows:

DO label [,] *variable = expression I, expression 2.*

On the other hand, you can also express the general DO statement as follows:

DO label *counter = lower limit, upper limit, step size.*

As already stated above, every loop is expected to have a terminator, and the 'label' in the above general forms of DO statement is actually the statement label of the loop terminator. In Fortran, the terminator of a loop initiated by a DO statement is given by a CONTINUE statement. Now see a simple **DO...** CONTINUE statement below:

```
DO 10 J = 1.40
      .
      .
      .
BODY OF LOOP
      .
      .
      .
10 CONTINUE.
```

The above example simply directs the computer to repeat 40 times the statements between the DO and CONTINUE statements. The block of

statements which constitutes the body of the loop is also called the RANGE of the DO statement (including the terminating statement).

You learn how to program by studying program examples. So, study the following program closely because it consists of additional features in Fortran to be explained shortly:

```
      PROGRAM ITER
      REAL X, T, FF
C     BELOW IS A USER-DEFINED FUNCTION
       F(X) = X**2 +3.0
      OPEN (7, FILE = 'C:\NOU\OUT.DAT',
       $ STATUS = 'NEW')
      WRITE (7,10)
10     FORMAT (2X, 'T' 10X, 'FCT))
      DO 20 T = 0, 4, 0.5
            FF = F(T)
             WRITE (7,*) T, FF
20    CONTINUE
      END
```

Definitely, you will observe that a few keywords have been added in the above program. In brief, below are some additional features beyond what you have learnt on Fortran in this course:

- The use of the keyword PROGRAM
- The use of a user-defined function
- Introduction of the OPEN statement
- The use of a line continuation symbol ($).

The keyword PROGRAM is used in Fortran as a Program Header statement, usually to give a name to your program code.

Below the comment line is an assignment statement which defines a function F(X). The use of user-defined function is essential in programming especially when a program has repeated references to such a function at different points of the program. As you can see in the above program, the function statement is used in the line following the DO statement. See the next explanation on the OPEN statement.

The OPEN Statement

The OPEN Statement is used primarily to connect a file or the printer to your program. Below is the general form of the statement:

OPEN [UNIT =] number, filespec list)

where "filespec" stands for file specification and there are various types of Fortran specifiers. The most common specifiers are:

FILE =
STATUS =
ACCESS =

Others are:

IOSTAT =
ERR=
FORM =
RECL =
BLANK =

As you can see in the example above, two specifiers were used in the OPEN statement, namely, "FILE =" and "STATUS =".

Briefly, in the above example, the OPEN statement connects your program to a device with unit label specified by the number 7 as indicated in the 'WRITE statements. In your case, the device is a hard disk and the file to be used is specified to reside in the folder "NOU", whose name is "OUT.DAT".

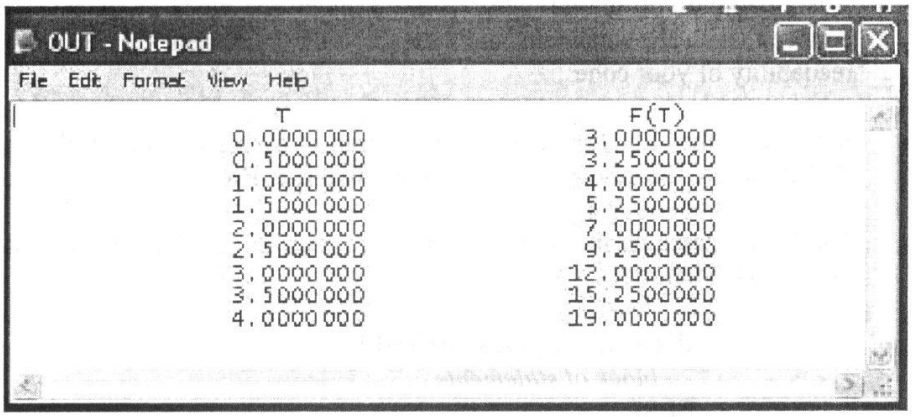
The STATUS specifier simply directs the computer to create the file during the program execution, hence its status type is "NEW". Other types of STATUS identifiers are:

- OLD
- SCRATCH
- UNKNOWN

A file identified as OLD for example, is a file already existing on the drive before the execution of your program.

Now, back to the DO... CONTINUE statement in the above example, the body of the loop is to calculate the values of the function F(T) for values of T from 0 to 4 with step size 0.5. The results are to be written to the file created by the OPEN statement. The output of the program is as seen below:

Figure 1



As you can see from the output above, the file "OUT.DAT" can be opened by you by any ASCII editor such as "Notepad".

In your DO ... CONTINUE statement, if the step size is not given, the Fortran compiler takes the step size as 1, just like the BASIC language interpreter.

SELF ASSESSMENT EXERCISE 1

There was a statement in the above example that was not necessary. What is the statement and why?

Answer

The statement is

REAL X, T, FF

The declaration statement is not necessary because the variables X, T, FF are already assumed to be real by default.

3.2 Using IF... THEN... ELSE Statements

It is good to state here that the use of IF ... THEN ... ELSE statement is very important in Fortran because it enable s Fortran programs to be written in a way more harmonious with structured programming.
The simple IF ... THEN statement has the following form:

IF *(logical expression)* THEN
block of statements
END IF
next statements.

It is a common programming practice to indent the block of statements between the IF statement and the ENDIF statement to enhance readability of your code.

Remember that ENDIF keyword in Fortran is a single word unlike in BASIC where the keyword is broken into two (i.e. END IF).

The ELSE statement is used in conjunction with IF ... THEN statement as expressed below:

IF *(logical expression)* THEN *block of statements*
ELSE
block of statements
ENDIF.

Now, look at the following example:

```
READ (*,*) X, Y
IF (X+Y . EQ. 0.0) THEN
WRITE(*,*) 'X = -Y'
        STOP
ELSE
        Z = (X+Y) + 2.0
        PRINT *, X, YZ
        END IF
```

In the above program code, on reading in the values of X and Y from the keyboard, the IF statement checks if X+Y = 0. If this is true, the control is transferred to the next two statements. The STOP statement terminates the program execution at this point. If the value of the logical expression is .FALSE., the control is passed to the statements following the ELSE statement.

In the above program, take note that the relational operator .EQ. is used and not the equality (—) sign.

SELF ASSESSMENT EXERCISE 2

Give the IF statement version for the above program for checking the following conditions:

i. X > Y
ii. X = Y

Answers

i. IF (X .GT. Y) THEN

or

IF (X-Y .GT. 0.0) THEN

ii. IF (X .LE. Y) THEN

or

IF (X-Y .LE. 0.0) THEN.

Before you round up this unit, see below the various forms of IF statements:

- The logical IF
- The block IF
- The ELSE IF

Their general forms are as follows:

Logical IF

IF (logical expression) statement

An example is as follows:

IF (J .GT. 4) GO TO 40

Block IF

IF (logical expression) THEN

An example is already given in the above Fortran code.

ELSE IF

ELSE IF (logical expression) THEN.

The use of IF... THEN... ELSE IF statement is essentially necessary to avoid having too many ENDIF's when using the IF statement to check many conditions in a multi-selection structure. A more general format for the ELSE IF statement is as follows:

IF (first logical expression)
THEN first block of statements
ELSE IF (logical expression) THEN
second block of statements
ELSE IF (logical expression) THEN
ELSE IF (last logical expression) THEN
last block of statements
ELSE
block of statements

ENDIF.

You will now round up this unit.

4.0 CONCLUSION

Definitely, you cannot cover everything about Fortran programming in this course as you should have observed in this unit. This unit has simply attempted to introduce you to additional common statements you will frequently need to use in Fortran programming. Developing programs that involve loops and conditional statements are common features in programming. This unit has shown you how to handle loops in Fortran by using the DO... CONTINUE statements. You have also seen the use of IF statements in its various forms and also how to use the relational operators in Fortran code.

5.0 SUMMARY

Fortran is very easy to learn as you have seen in this unit and the last two units. This unit has extended the use of Fortran statements to the basic features of structured programming available in the language.

With these three units on basic features of Fortran programming, you have been equipped with the fundamental knowledge of Fortran to get started with the language

In the subsequent units, you will also be introduced to the fundamental knowledge of some other languages so as to have a broad basic knowledge of the commonly used programming languages.

6.0 TUTOR-MARKED ASSIGNMENTS

1. Use the WATFOR 77 software to code and run the following Fortran program:

```
INTEGER SUM
SUM= 0
DO 10 D = 1,7
      IF (K .GT. 3) THEN
      SUM = SUM + K
ELSE
      SUM = SUM * K
ENDIF
PRINT*, SUM 10 CONTINUE END
```

2. a. State the three segments that characterize a loop and hence write out the segments in the loop contained in the program above.

b. Describe the use of the following status identifiers employed in OPEN statement:

OLD
SCRATCH

3. What is the difference between the STOP and END statements?

7.0 REFERENCES/FURTHER READINGS

Fatunla, SD., Fundamentals of Fortran Programming, ADA + JANE Press, 1993.

Monro, D. M., Fortran 77, Edward Arnold, 1987. WATCOM System Inc., WATFOR-77 V3.0, 1988.

UNIT 2 INTRODUCTION TO PASCAL LANGUAGE

CONTENTS

1.0 INTRODUCTION

This unit will introduce you to the third language in this course, namely, the PASCAL Language. In this unit, you will be discovering for the first time what was never necessary in the first two languages you have studied. Pascal has its own unique way of code development as you will remember in Unit 3 of this course when studying some basic elements of algorithms. You will remember that Pascal was said to have its own specific form of pseudocode that you can easily transform into Pascal code.

In this unit, you will therefore be introduced to the basic characteristics of Pascal language and its keywords and functions. Now, look at your study objectives below:

2.0 OBJECTIVES

At. the end of this unit, you should be able to:

- Describe the fundamental characteristics of Pascal programs
- Distinguish some differences between Pascal language and other programming languages
- Identify the essential operators and keywords used in Pascal programming
- Describe the Pascal Library Functions.

2.0 MAIN CONTENT

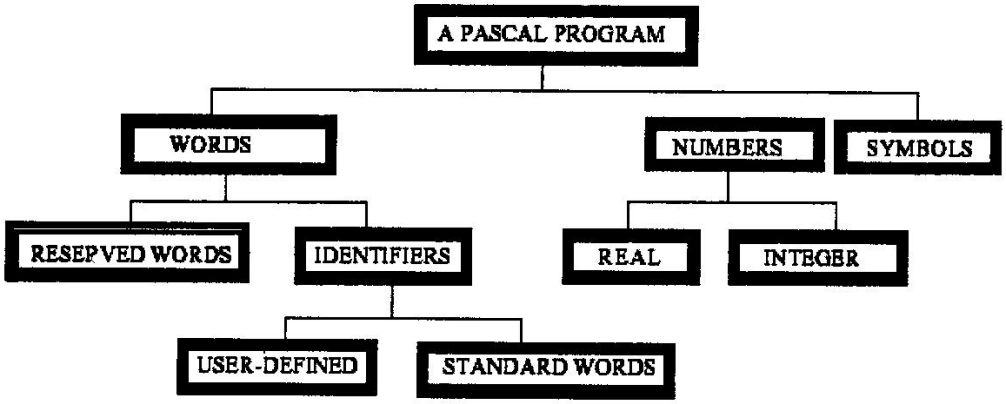
3.1 Getting Started with Pascal Programming

To start with, it is good to state that Pascal was designed to be compatible with structured programming concepts. Essentially, the language offers alternative logic structures to avoid the use of branching (or GO TO) statements that is highly discouraged in structured programming.

Pascal allows variables of any length, although only the first six characters have meaning to the computer and it is more English- like than the Fortran language. Generally, Pascal has been observed to have poor input and output (I/O) capabilities, however, it has very good graphic capabilities unlike BASIC and Fortran.

Below is the summary of the elements of a Pascal program illustrated in the following chart.

Figure 1



Generally, each Pascal program has two basic parts:

- HEADING
- BODY.

The program heading is for definitions and declarations. In fact, the program heading starts with the word PROGRAM followed by an identifier and program parameters. You will see all these in examples very shortly.

The program body simply consists of Pascal statements, each of which must be separated from its successor by a SEPARATOR which is usually a semicolon. However, it is good to remark that certain reserved words also serve as separators in Pascal coding

To learn how to program and explain some important concepts, you need to study program examples. You will now see a program below to learn the general basic characteristics of a simple Pascal program.

```
PROGRAM Circumference (INPUT, OUTPUT);
(This program finds the circumference of a circle)
CONST
        Pi = 3.14159;
 VAR
        Radius, circum: REAL;
BEGIN
        READ (radius);
        Circum: = 2 * Pi * radius;
        WRITE (' Circumference =', circum);
END.
```

Before explaining the features of Pascal programming in the above program, do the following exercise.

SELF ASSESSMENT EXERCISE I

Construct the Pascal Pseudocode for the above Pascal program. (Remember you studied this in Unit 3 of this course).

Answer

```
ALGORITHM Circumference
DECLARE
    Pi = 3.14159
    radius, circum : REAL
EXECUTE
    INPUT radius
    Circum ? 2 * Pi * radius
    Output 'circumference =', circum, MORE
END circumference.
```

Now, closely looking at the above program example, you will observe that a Pascal program can be said to consist of the following sequences:

- Definitions
- Declarations
- Statements
- Optional Remarks.

When you divide a Pascal program into two major parts: Heading and Body, the heading identifies the program name and it is better and conventional to use an identifier that suggests what the program is expected to do. The name of the program is followed by (INPUT, OUTPUT) which indicates to the computer that the program will receive data from its standard input device and also transmit data to its standard output device. The words INPUT and OUTPUT are in this context

called the *Program Parameters.*

Below the program header is a comment enclosed within braces { }. So, Pascal uses braces instead of using a keyboard as in BASIC or a character as in Fortran. The comments can be placed anywhere in a program where a space is allowed. Some Pascal compilers also accept double asterisk symbols (*...*) to enclose comments.

There are two word symbols:

- CONST
- VAR.

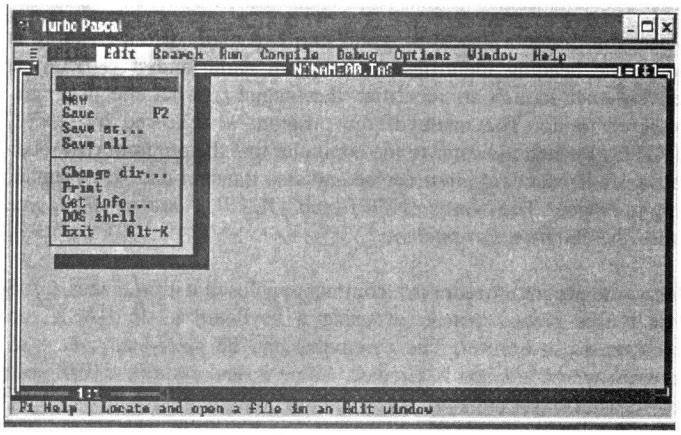
The word symbol CONST indicates that the following data items will *define* the CONSTANTS to be used by your program. The word symbol VAR also indicates the beginning of the *declarations* of the names or identifiers to be used for VARIABLE data.

The body of the program starts with the word symbol BEGIN and terminates with the word END. Remember that the END statement is followed by a full stop (.) as an essential part of the program code. END statements in subprograms end with semicolons. From what you have seen above, a Pascal program is *punctuation — sensitive.*

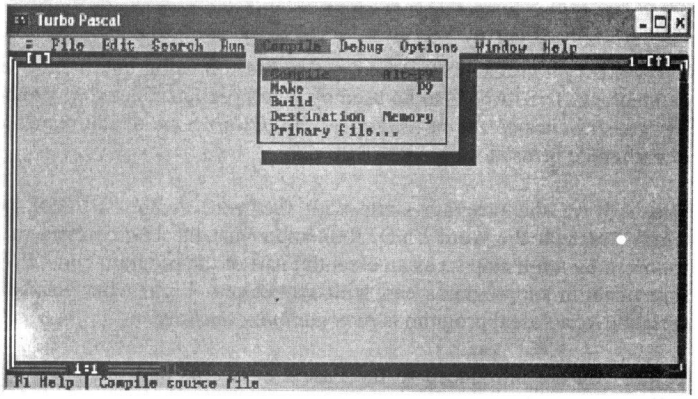
Before you are introduced to the Pascal keywords and operators, you will observe in the above program example that the assignment operator in Pascal is made up of a combination of two single symbols (:=).

To fully get started with Pascal programming, you need to be introduced to the Pascal programming environment. Below is the programming environment for Turbo Pascal Compiler:

Figure 2



See the next menu screen too:



Looking at the above two screen windows of the Turbo Pascal compiler, you will observe some similarities with the Turbo BASIC environment. You will be using the compiler to run some Pascal programs later in this course.

3.2 Pascal Keywords and Operators

The reserved words used in Pascal are as follows:

AND, ARRAY, BEGIN, CASE, CONST, DISPOSE, DIV, BOOZEAN, CHAR, DO, DOWNTO, ELSE, END, FILE, FOR, FUNCTION, GET, GOTO, IF, IN, INTEGER, INPUT, FALSE, LABEL, MOD, NEW, NOT, OF, OR, OUTPUT, OTHERWISEW, MAXINT, PACKED, PACK, PAGE, PROCEDURE, PROGRAM, PUT, READ, RECORD, READLN, REAL, REPEAT, RESET, REWRITE, SET, THEN, TO, TRUE, TYPE, UNPACK, UNTIL, VAR, WHILE, WITH, WRITE, WRITELN, TEXT.

Already, some of the above reserved or pre-defined Pascal words have been used in the program example considered in the last section. Looking at the example closely, the following reserved words were employed in the program.

- PROGRAM
- CONST
- VAR
- BEGIN
- REAL
- READ
- WRITE
- END
- INPUT
- OUTPUT.

Below, you have the operators being used in Pascal programming as classified generally:

Arithmetic Operators

+	**:**	Addition
-	-	Subtraction
*	-	Multiplication
/	-	Division
:=	-	Assignment

Relational Operators

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

Boolean Operators

AND	:	The AND Operator
OR	:	The OR Operator
NOT	:	The NOT Operator.

SELF ASSESSMENT EXERCISE 2

There is an operator that exists in BASIC and Fortran but is missing in the groups of operators listed above. What is it?

Answer

The Exponentiation Operator.

The single operator for exponentiation is a uniquely absent feature in Pascal.

You will now be introduced to the groups of Pascal library functions and procedures as follows in the next section.

3.3 Pascal Standard Functions and Procedures

The commonly used Pascal library functions and procedures are classified as follows:

- Arithmetic Functions
- Transfer Functions
- Boolean Functions
- Procedures.

Arithmetic Functions

SQR (X)	:	Square of X (i.e. X2)
SQRT (X)	:	Square Root of X (i.e. √X)
ABS (X)	:	Absolute Value of X (i.e. |X|)
EXP (X)	:	Exponential Function of X (eX)
LN (X)	:	Natural Logarithm of X (logeX)
SIN (X)	:	Sine of X
COS (X)	:	Cosine of X
ARCTAN (X)	:	Arctangent of X (tan-1(X))

Transfer

CHR (X)	:	Character corresponding to the value of the integer type X.
ORD (X)	:	Ordinal number of the scalar X.
TRUNC (X)	:	Integer result of truncating X.
ROUND (X)	:	Integer result of rounding real X to the nearest integer

Boolean Functions

EOF (filename)	:	Returns TRUE if end of file is reached
EOLN (filename)	:	Returns TRUE if end of line is reached in the file.
ODD (X)	:	Returns TRUE if the integer parameter X is an odd number.

Procedures

NEW (X)	:	Creates a new address for the pointer X.
DISPOSE (X)	:	Disposes of the address of the pointer X.

The following two functions are also employed in Pascal:

SUCC (X)	:	Supplies the value after the current value of scalar type X within the ordered set of values that the scalar type on take.

SELF ASSESSMENT EXERCISE 3

Discuss how to express the exponentiation operation and tangent function in Pascal since there are no direct ways of expressing them.

Answer

The exponentiation operation can be expressed by using the SQR function. This can also be done by the EXP and LN functions, carefully. The tangent function also can be expressed by using the SIN and COS functions.

You will now round up this unit, having been introduced to the basic knowledge of Pascal to get you started with the programming language.

4.0 CONCLUSION

In this unit, you have been introduced to the general features of the Pascal language.

Something unique with PAS as you have seen in this unit is the peculiar punctuation marks that are essential in its code unlike in BASIC and Fortran that you have studied in this course.

The unit has also introduced you to the commonly used reserved words and operators with the language library functions.

5.0 SUMMARY

In this unit, you have seen that as good as the Pascal language is, being developed to be compatible with structured programming features, yet the language has few demerits.

For example, there are no direct ways of expressing exponentiation and tangent function as in other languages.

Having been introduced to the general features of the Pascal language, you will in the next unit be taken through some examples that employ the reserved words and functions studied in this unit.

6.0 TUTOR-MARKED ASSIGNMENTS

1. Apart from the semicolons, state three reserved words that also serve as separators in Pascal code.
2. a. State where remarks cannot be placed within Pascal code.
   b. How do you have access to the output window of the Turbo Pascal Compiler?
3. Write a Pascal program that finds the area of a rectangle, using the standard I/O devices.

7.0 REFERENCES/FURTHER READINGS

Huggins, E., Mastering Pascal Programming, the Macmillan Press Ltd., 1983.

Salaria, R.S., Computer Oriented Numerical Methods, Khanna Book Publishing Co. Ltd., 1999.

Borland International, Inc., Turbo Pascal Version 6.0, 1990.

UNIT 3 STRUCTURED PROGRAMMING WITH PASCAL

CONTENTS

1.0 INTRODUCTION

As already learnt in the last unit, Pascal was designed to be compatible with the structured programming features. Do you remember the three basic structures of structured programming? They are:

- Sequence Structure
- Selection Structure
- Iterative Structure.

In this unit, you will be introduced to the Selection and Iterative Structures as they are used in Pascal. You will remember that the sequence structure is embedded in every other structure.
There is GOTO statement still made available in Pascal but is rarely used since it generally upsets a program structure and reduces the clarity of the program. It is therefore a common practice to confine the use of GOTO statement to those very few instances when there is no alternafive construction.

2.0 OBJECTIVES

By the end of this unit, you should be able to:

- Identify various types of iterative structures available in Pascal.
- Use some additional keywords available in Pascal Programming language.
- Use the IF-THEN Statements to perform alternation in Pascal.

3.0 MAIN CONTENT

3.1 Iterative Structures in Pascal

The iterative structures in Pascal are as follows:

- FOR loop
- WHILE loop

- REPEAT loop.

The FOR Loop

One of the major strengths of a computer is its ability to repeat selected instructions with different data. That is what the FOR statement does when used in Pascal programming. The general construction form for the FOR statement is as follows:

FOR < *control var* >:= <*expr*> TO <*expr*> DO <*statement*>

where <*control var*> *is* an integer variable and <*expr*> is any expression that can be evaluated into a value of the same type as the control variable. The component <*statement* > may be any statement (simple or compound) or another FOR statement.
Look at the example below:

Example 1

```
PROGRAM Example 13 (OUTPUT);
VAR
    a : REAL;
    b : INTEGER;
BEGIN
    a: = 1.24
    FOR b = 0 TO 5 DO
    BEGIN
        WRITELN (b, a);
    END;
END
```

The output of the above program is as follows:

```
0       1.2400000000E+00
1       6.2000000000E+00
2       3.1000000000E+01
3       1.5500000000E+02
4       7.7500000000E+02
5       3.8750000000E+03
```

In the above program, the compound statement bounded by the inner BEGIN and END delimiters has been repeated six times under the execution of the FOR statement. You will however observe that the inner END statement is followed by the normal separator, semi colon, since it is not the end of the program.

The WHILE Loop

The WHILE ... DO statement is another iterative statement available in Pascal. The general representation is as follows:

WHILE <*expression*> DO <*statement*>

See the following program that uses the WHILE statement.

Example 2

```
PROGRAM Example 13a (OUTPUT);
CONST
      firstNumber = 1;
       Multiplicant = 0.05
 VAR
       number, lastNumber : REAL;
BEGIN
       number: = firstNumber;
       WHILE number > 0 DO
       BEGIN
            lastNumber : = number;
            number : = number * 0.05;
       END;
       WRITELN (Number =', number);
       WRITELN (The Last Number =', lastNumber);
END.
```

From the program above, the WHILE statement will be repeated until the value of *'number'* becomes so small beyond what your computer can recognize as zero.

The REPEAT Loop

The REPEAT statement is very similar to the WHILE statement The general form of its representation is as follows:

REPEAT <*statement*> UNTIL <*Boolean expr*>

where <*Boolean expr*> is a Boolean expression.
There are two major differences between the WHILE and REPEAT loops;

- The REPEAT and UNTIL statements act as delimiters in the same way as BEGIN and END statements. So, the REPEAT loop doesn't need to start with BEGIN statement and end with

END statement as in WHILE loop.

- I
  n
  the WHILE loop, the condition of the Boolean expression is tested before the loop is entered, whereas a REPEAT loop will always be executed at least once, even if the condition is false at the beginning.

Now, see an example of the REPEAT loop below:

Example 3

```
PROGRAM Sales (INPUT, OUTPUT);
 VAR itemNumber, quantity : INTEGER;
        total, unitPrice, totalPrice: REAL;
BEGIN
        Total: = 0;
        WRITELN
                (19/N          Qty              Unit Price Total);
REPEAT
        READ (itemNumber, quantity, unitPrice);
        totalPrice: = quality + unitPrice,
        WRITTELN (totalPrice : 38: 2);
        Total: = total + totalPrice;
        UNTIL itemNumber = 0;
        WRITTEN
        ('Grand Total = N', total : 9: 2);
END.
```

SELF ASSESSMENT EXERCISE 1

Run the above program.

Answer

The output is as follows:

S/N	**Qty.**	**Unit Price**	**Total**
1.	3	5.50	16.50
2.	45	0.20	9
3.	12	3.40	40.80
Grand Total =			N66.30

To terminate the REPEAT loop, you simply enter the digit 0 as indicated in the UNTIL statement. In the above program, a WRITELN statement such as

WRITELN (totalPrice : 38 : 2);

simply states that the value of "totalPrice" should be printed with length 38 and 2 decimal places. You will now be introduced to the next structure to be treated in this unit.

3.2 IF-THEN Statements in Pascal

As already learnt in the previous two languages you have studied, that is, BASIC and Fortran, the IF-THEN statement is meant for performing an alternation process on fulfillment of a condition.

As in other languages the general form of its representation in Pascal is as follows:

IF <*Boolean expression*> THEN <*statement*>

The difference between the WHILE and IF statements is that whereas the WHILE statement will repeat the statement until the associated conditional expression is satisfied, the IF statement will not. If the expression following an IF is true, then the statement following its THEN part is executed (once only). If, on the other hand, the expression is false, then the statement is ignored by the computer and the control is passed to the next instruction in the program.

The usage is seen in the following example before you round up this unit.

Example 4

```
PROGRAM Example 13b (INPUT, OUTPUT);
VAR ch: CHAR;
BEGIN
      READ (ch);
      IF (ch> = '0) (ch < = '9) THEN
      WRITE (digit);
          IF (ch> = 'A) AND (ch < = 'Z') THEN
                OR (ch = 'I') OR (ch = 'E')
              OR (ch = 'U') THEN WRITE ('vowel');
ELSE WRITE ('consonant');
ELSE WRITE ('error');
END.
```

Definitely, the above program has introduced you to more features of the IF statements. For example, the program incorporates the IF-THEN-ELSE construction whose general form is as follows:

IF <*expression*> THEN <*statement*> ELSE <*statement*>

For nested IF statements, the following rule should be followed:

IF <*expression*> THEN
IF <*expression*> THEN <*statement*>
ELSE <*statement*>
ELSE <*statement*>.

In brief, the rule is that, an ELSE is paired with the nearest preceding otherwise unpaired THEN.
In the above program, you will notice that if the input is a digit, your output will be

"digiterror"

This is as a result of an incorrect use of the ELSE statement.

SELF ASSESSMENT EXERCISE 2

What statement do you amend to correct this error?

Answer

The first four statements after the BEGIN word symbol should be as follows:

READ (ch);
IF (ch > = '0') AND (ch < '9') THEN
WRITE (digit);
ELSE IF (ch >= 'A') AND (ch <= 'Z') THEN.

The above program has introduced you also to the use of the Boolean operators and of course a variable type CHAR.

With all that you have learnt in this unit and the last unit, you have actually started with Pascal programming. These two units are just intended to do that. A complete course on Pascal Programming will let you dig deeper into the usage of the good features of Pascal programming.

4.0 CONCLUSION

This unit has taken you through the features of structured programming embedded in Pascal programming language. The iterative structure can be carried out with either the FOR, WHILE or REPEAT statement as you have seen in this unit.
The selection structure is implemented in Pascal the same way as in

other languages. The unit has shown you the careful manner the ELSE statements should be paired with IF statements.

5.0 SUMMARY

Structured programming features are well supported in Pascal language as you have seen being demonstrated in this unit. Though the GOTO option is still available in Pascal, it is rarely used since the iterative and selection structures can be used in place of such branching statement. With all that you have learnt in this unit, you have the fundamental knowledge needed to start programming in Pascal. An extended knowledge to cover the other reserved words listed in this unit will require a treatment in a complete course on Pascal programming. Iii the next unit, you will be introduced again to another programming language.

6.0 TUTOR-MARKED ASSIGNMENT

1. Run the Pascal program in Example 2.
2. Write a very short Pascal program that gives the following output:

```
True =    1
False =   0
```

3. Study the following Pascal code:

```
IF ((a = 1) AND (b = 1)
    OR (a <>1) AND (b <> 1)
        THEN answer : = TRUE
        ELSE answer : = FALSE.
```

Give a single assignment statement which is equivalent to the above 4 — line code.

7.0. REFERENCES/FURTHER READINGS

Huggins, E., Mastering Pascal Programming, The Macmillan Press Ltd., 1983.

Borland International Inc., Turbo Pascal Verb o n 6.0, 1990.

UNIT 4 INTRODUCTION TO C++ LANGUAGE CONTENTS

1.0 INTRODUCTION

This unit is going to introduce you to one of the most versatile programming languages today. That is the C++ language which was designed by B. Stroustrup and published in his book, *"The C++ programming Language"* in 1986. The C++ has been fundamentally derived from the original C language which was published by B.W. Kernighan and D. M. Ritchie in 1978. The C++ designation is simply related to the expression C++ which you can write in a C program to increment a variable C.

One of the attractive features of C++ is that the language offers good facilities for Object-Oriented Programming (00P). In brief; the C++ language is a better C language since there are some facilities available in C-H- which are not available in the original C. This unit will simply give you an overview of the C++ language with its keywords and operators. Now, look at your study objectives below.

2.0 OBJECTIVES

By the end of this unit, you should be able to:

- Describe the general form of a C++ program.
- Explain the role of Header files in C-H- programs.
- Identify the different categories of operators used in C++ language.
- Write simple C++ programs and use a compiler to run them.

3.0 MAIN CONTENT

3.1 An Overview of C++ Language

In this unit, you will be using the Turbo C-1-1- compiler to run the few programs you will encounter in the study. The compiler offers you the flexibility of compiling both your C and C++ programs. Just simply

remember to save your source program with extension C or CPP for the original C or C++, respectively.

Now, you can only develop programs when you are familiar with the grammar rules governing the language just as you have in other languages. For now, C++ programs can be said to majorly consist of the following:

- Comments
- Header Files
- Identifiers
- Statements
- Functions.

Remember, just like in other languages, the program statements consist of the operator allowed by the C++ language.

In brief, a simple C++ program has the following general representation:

```
// <comments> ...
# include <header file>
.
.
.
 # include <header file>
main (<parameters>)
 {<identifiers and statements> ;
 .
 .
 .
 }
```

You will now see the explanation on the above general form as follows:

Comments

Generally, it is a good programming practice to start your program with meaningful comments. Just as there are ways of inserting comments in other languages, C++ also has its own way of identifying a comment. In C++ language, you begin your comment with the two characters (or double slash):

//

The end of the line is simply the end of your comment. Analogously, the original C language uses the two characters:

/*

to begin your comment and the two characters:

*/

to end the comment.

Include Lines

The include lines are used to "include" the header files. The header files, usually with extension .h, are files employed by C++ to declare functions and to define macros. They serve in reducing the amount of source code or program lines. Thus logically, an include line is used to include the contents of a header file.

Include lines usually begin with:

#include

and they should be on separate lines of their own.

Below is a list of some header files available in a specified directory created for them when you install your C++ compiler:

alloc.h, assert.h, bcd.h, bios.h, complex.h, ctype.h, conio.h, dir.h, dos.h, errno.h, float.h, graphics.h, io.h, iomanip.h, limits.h, math.h, mem.h, stddelh, stdio.h, stream.h, iostream.h, stdlib.h, time. h, values. Ii, etc.

The above files are STANDARD header files residing in the INCLUDE subfolder of your main Turbo C++ directory. You can also write your own header files.

Functions

Being a structured programming language, C++ program contains one or more functions, one of which is called "main" as used in the above general form.

main *(<parameters>).*

As seen in the general construction, you do not need the term "function" to designate it. A function in C++ simply denotes a concrete program segment. In C++, your functions may or may not have parameters. Thus you can simply have something like this:

```
Main ( )
{
.
.
.
}
```

It should be noted that the body of every function in C++ is embedded between braces { ...}. The two braces can be written on the same line or in the same vertical column.

Statements

The C++ statements can be DECLARATIONS or executable statements. Declaration statements are used to specify the variable types. In C++, you can include your declarations in an assignment as you will see below in some examples.

Just as you have in Pascal, it is interesting to state that C++ statements end with semicolons, as indicated in the general form above.

Before you are introduced to a simple C++ program, you need to know of the C++ variable types.

Variable Types

Variables in C++ are of the following types with their fixed sizes in memory:

Variable Types	Sizes in Memory
char	1
double	8
enum	2
float	4
jut	2
long	4
(or long Mt)	4
long doubk	10
short (or short int)	2

Now, see your first C++ program below, being a program which reads two real numbers from the keyboard to compute their average:

```
// A program that finds the average of two numbers
# include <iostream.h>
main ()
{ cout << "Enter Two Number: ";
```

```
float x, y;
 cm n >> x >> y;
float mean = (x + y)/2;
 cout << "Average = " << mean << endl;
 }
```

The above program uses the 'iostream.h' header file in its include line because of the standard input and output streams 'cin' and 'cout used in the program. In the original or traditional C language, you can use the standard functions 'scanf" and 'printf" which are contained in the `stdio.h' header file.

As explained above, you need some knowledge of the contents of some header files so as to know what to use in your include lines.

The statement:

cin >> x >>*y;*

reads two values from the standard input stream (i.e. from the keyboard) and stores them into the variable x and y. The operator used here appears like an arrow head which sends the values from 'cm' to x and y. It is called a *"Shift Operator"*.

In the same way, the statement:

cout << "Enter Two Numbers:";

uses the standard output stream 'cue which sends the characters between the double quotes to the video screen.

As you can see in the above program, variables x and y are declared to be floating numbers. The declaration of the variable *'mean'* is done with the assignment.

The use of "endl" in the program simply specifies that you are at the end of the line. Instead of using "endl", you can also use

‘\n’ or “\n”.

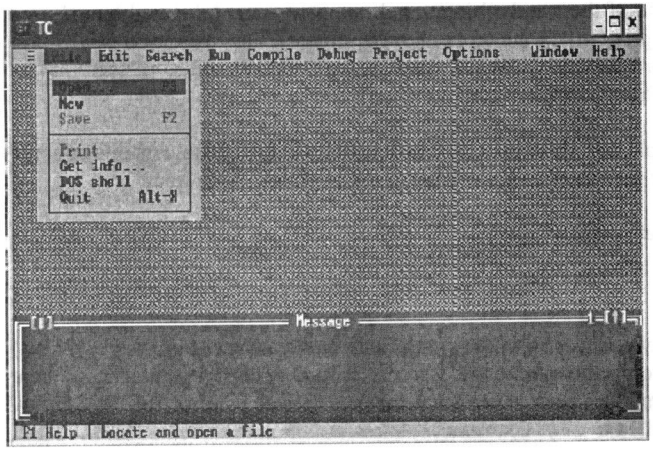
Now, when the above program is compiled, your output will look like this:

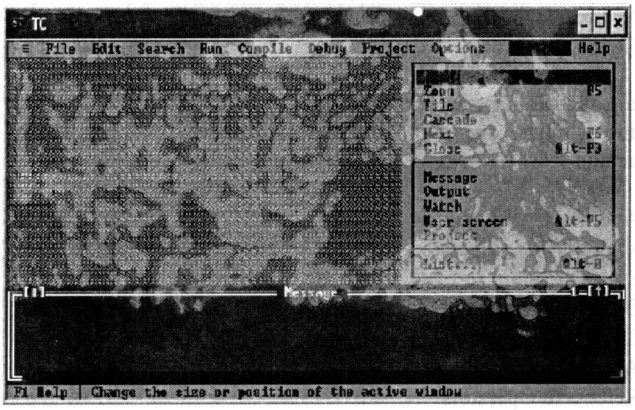
Enter Two Numbers: 2.4 4.6
Average = 3.5

Below are two window screens of Turbo C++ compiler. The "Message" window is meant to display error messages or compilation messages.

The final output of your program can be viewed by selecting "Output" from the "Window" menu as shown in the figure below:

Figure 1





3.2 OFF Keywords and Operators

Already, in the above example you considered in the last section, you have seen some of the C++ keywords and operators.

Keywords

By keywords, the reserved words are also included in the list below:

> *asm, auto, break, case, catch, char, class, const, continue, default, delete, do, double, else, enum, extern, float, for, friend, goto, g inline, int, long, new, operator, private, protected, public, register, return, short, signed, sizeof static, struct, switch, template, this, throw, try, typedof union, unsigned, virtual, void, volatile, while, etc.*

As you can see above, C++ has many friendly reserved words, in fact one of them is even named 'friend' !. Generally, as in other languages, the needs of your problem will suggest the keywords to be employed in your C++ program.

C++ Operators

Generally, C++ has many operators and the language treats as operators things that other programming languages do not. This is what makes some people to see the C++ language as seemingly difficult. However, the language is not as difficult from the little you have seen in this unit already.

The operators will be classified as follows for easy understanding:

- Arithmetic Operators •
- Binary Operators.

However, the arithmetic operators are also included under some categories of binary operators. It is therefore sufficient to just see the operators under the following categories of the binary operations:

- Additive Operators
- Multiplicative Operators
- Shift Operators
- Bitwise Operators
- Logical Operators
- Assignment Operators
- Relational Operators
- Equality Operators.

There are other operators too that may not be appropriate to classify under the above groups. The language has many operators as you have been told. It is an operator-driven and function-driven language.

Additive Operators

+ : Addition
- - Subtraction.

Multiplicative Operators

* : Multiplication
/ : Division
% : Remainder.

Shift Operators

<<	:	Shift Left
>>		Shift Right

Bitwise Operators

&	:	Bitwise AND
∧	:	Bitwise X OR (Exclusive OR)
|	:	Bitwise OR (Inclusive OR) — the "Pipe" character on your keyboard

Logical Operators

&&	:	Logical AND
||	:	Logical OR (double pipe)
!	:	Logical NOT

Assignment Operators

=	:	Assignment
*=	:	Assign Product
/=	:	Assign Quotient
%	:	Assign Remainder
+=	:	Assign Sum
-=	:	Assign Difference
<<	:	Assign Left Shift
>>	:	Assign Right Shift
&=	:	Assign Bitwise AND
∧=	:	Assign Bitwise XOR
|=	:	Assign Bitwise OR

Relational Operators

<	Less than
>	Greater than
<=	Less than or equal to Greater than or Equal to
>=	

Equality Operators

= =	:	Equal to
!=		Not equal to

There are other operators, and one of such is:

sizeof : To check the size of a variable type.

As you have seen in the list above, you will agree that C++ is really an operator-driven language as earlier asserted. You need to know how C++ handles a particular type of variable described below.

Printers in C++

The C++ language uses extensively what are called POINTERS. In brief, a variable that has an *address* as its value is called a POINTER. Pointers are therefore employed to generally store ADDRESSES just as you use arithmetic variables to store numbers.

Now, C++ uses the unary operator "" to designate a pointer variable. Thus a declaration such as follows:

float *r;

defines r as a *pointer-to-float* variable while the value of r itself is an address. There is therefore an operator, called the 'ADDRESS OF' operator, represented by '&'. This is an inverse operator of '*'. The operator '*' is also called t he INDIRECTION operator, and when you write:

*r

it simply means *"the contents of"*. From all the explanations above, if you have for example:

this simply meani r.

SELF ASSESSMENT EXERCISE 1

There is an operator indirectly referred to in the introduction of this unit which is not listed above. What is it?

Answer

The operator is the first of the two listed below. The second is its inverse:

++	:	Increment
--	:	Decrement.

Before you round up this unit, look at the following example:

```
#include <iostream.h>
main ( )
{ int i, &x = i;
i = 4; x += 10;
cout << "New Value of i = "<< i << ".\n";
```

```
 cout << "Using x, we have i = "<< x <<".\n";
}
```

The output of the above program will give you the following:

New Value of i = 14.
Using x, we have i = 14.

In the above program, x is a reference variable, using the "address of' operator. The program also uses the "Assign Sum" operator += to add 10 to 4.

Instead of using the "endl" keyword, the program uses "\n" alternative to end the output line.

You will round up this unit now since all about C++ cannot be exhaustively covered in this course. The course is meant to give you a broad view of programming tools available in programming languages.

4.0 CONCLUSION

In this unit, you have been introduced to the general fundamental characteristics of the C++ language. You have seen the peculiar features in the language that are absent in BASIC, Fortran and Pascal languages that you have studied in this course. A typical example of these features is the idea of header files. You will need to have an idea about the contents of some header files so as to know what are the appropriate header files to include in your programs.

The unit has shown you the richness of C++ in terms of operators. The same thing holds for usage of functions. A complete course on C++ language will help you to understand more of the language which has been classified as a Middle-Level language because of its immense capabilities, especially for developing operating systems.

5.0 SUMMARY

Obviously, C++ language is a very rich language in terms of its capabilities and the features it offers. This unit has shown you some of the basic features of the language and with what you have learnt in the unit, you can easily get started with the language.

The header files used in the include lines can either be selected among the standard files kept in the INCLUDE subfolder of your Turbo C++ Directory or be personally developed by you and kept in the INCLUDE subdirectory or your current directory.

This unit will be the only unit devoted to C++ language just to give you an introduction to the language. However, it should be noted that a good knowledge of C++ will help you to easily and quickly understand the JAVA language. The next unit will also take you further into another language.

6.0 TUTOR-MARKED ASSIGNMENTS

1. What is the keyword ENUM used for in C++?
2. Write a C++ program that calculates the area of a circle, the value of the radius being read from the keyboard.
3. Study the following C++ program:

```
# include <iostream.h>
void f(int n)
{ if (n > 0)
        { f(n-2); cout << n <<""; f(n-1);
        }
{
main ( )
{ int k;
        cout << "Enter k: "; cin >> k;
         cout << "Output: \n";
```

a) How many functions are in the above program?
b) What is the function that calls itself in the program?

7.0 REFERENCES/FURTHER READINGS

Ammeraal, L; C++ For Programmers, John Wiley & Sons Ltd., 1991

King, M., Pardoe, J. and Vickers, P. A First Course in Computer Programming Using C, McGraw-Hill Book Company, 1995;

Borland International Inc., Turbo C++, 1990.

UNIT 5 INTRODUCTION TO HTML CONTENTS

1.0 INTRODUCTION

This unit is introducing you to a programming language that seems to be the most easiest language to learn today and also the most demanding language to publish web pages on the Internet.

HTML simply stands for "HyperText Markup Language". It is interesting to let you know that you can master enough of HTML syntax in a few hours to be able to create a wonderful web page. The unit will first of all take you through some basics of how web pages work so as to be able to understand what various HTML commands are expected to perform when executed by a web browser.

The first concepts to fully understand in HTML are tags and text formatting. This unit will get you started with the knowledge of these fundamental aspects of HTML. Now, look at your study objectives for this unit.

2.0 OBJECTIVES

By the end of this unit, you should be able to:

- Describe what a web page is and how it works.
- Explain various ways of creating HTML pages
- Explain how to use tags and format text in HTML document.

3.0 MAIN CONTENT

3.1 Fundamentals of Web Pages

By now, you are expected to have the basic idea of what is called the INTERNET, which is the network of sub-networks of computers across the world. Today, HTML pages are the standard interface to the Internet. The inter-linked HTML pages were named the World Wide Web (www) while the special programs written to view web pages are called *Web Browsers.* Examples of Web Browsers are the Microsoft

Internet Explorer, Mozilla Firefox, Netscape Navigator, etc.
Basically, Hypertext means text stored in electronic form with cross-reference links between pages. However, HTML or web pages today may include the following:

- Text
- Sound
- Video
- Animated Graphics
- Interactive Programs.

The world wide web is the unique part of the Internet that relies on HTML, but there are other services of the Internet and an example is the E-mail service.

Apart from what are listed above that can be included on Web Pages, you can also have the following as parts of the contents:

- Links
- Forms
- Frames
- Image Maps
- Java Applets
- Counters.

See few explanations on web page contents below:

Text

Text is usually said to travel quickly over the Net (short for Internet). Generally, you have lots of control over text, such as its font size, colour, alignment and other properties.

Graphics

It is a common saying that, "a picture is worth a thousand words". Though graphics are good forms of information presentation, you should know that it takes a longer time to retrieve or download them from remote computers on the Internet.

Generally, you should expect vie wing delays of web pages since most of them are located on some other computers different from your own. If web pages have many graphics, expect to spend some time waiting for them to fully be downloaded on your computer. As a website developer or programmer, learn how to include graphics that are worth seeing and small for easy retrieval. Graphics can be animated on web pages.

Multimedia

With video and sound files included on web pages, you must have suitable software to view or listen to them.

Links

No good web page without some vital links to direct visitors to other pages. These links are generally called *"hyperlinks"*.

Forms

To get feedback from web pages, the best way is to use forms. A form consists of a set of slots into which website visitors can enter some required information. However, your Internet Service Provider (ISP) must rim software that can collect data submitted by your website visitors and pass them to you.

Frames

Frames are used to divide your website into a set of separate areas, each of which can display a different file and be changed independently. They are employed to enhance the look and the usability of your website or web pages (- a website is a collection of web pages).

Image Maps

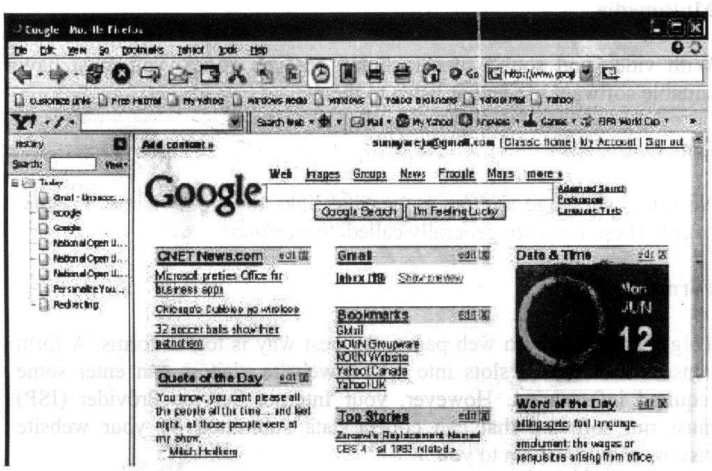
These are larger pictures having a number of hyperlinks embedded within them. They are majorly used to act as *'contents'* lists at the top of websites.

Java Applets

Java applets are little programs you can embed into your web pages to be downloaded or run within the browsers of the page visitors.

From the list above, you can see that web pages are simply "Assemblies of Elements" and most of the elements are located in many different computer files which may also be stored in different computers anywhere in the world.

Now, look at a web page below:



The above web page is a combination of text and animated graphics. Especially the Clock graphic on the right hand side of the page is animated to provide the current time and date. The web browser used in the figure above is the *Mozilla Firefox,* a free browser.

You will now see the use of tags and text formatting in HTML to create web pages.

3.2 Text and Tags in HTML

You have already seen in the list unit how web pages work, with an example of a web page.

By definition, TAGS are instructions to Web Browsers, directing them how to do the following:

- How to lay out text
- What graphics to display and where
- What distant pages to link
- And a variety of other things.

Below are some common basic rules for using tags:

- Each tag must be enclosed in <angle brackets>.
- Upper or lower case letters can be used but upper case makes them identifiable easily among HTML commands.
- Most tags are in pairs but identical while the closing tag starts with a forward slash(/).
- Browsers ignore spaces or new lines around tags.

Just have a look at part of the HTML source code of a sample web page (of the Course Developer):

Figure 2

As seen above, you can develop your HTML program code using any ASCII editor such as the Microsoft Notepad. You only need to save the document file with extension .html or .htm for the Web browser to identify it.

Below are the HTML tags every HTML page or document must have:

- <HTML>
- <HEAD>
- <TITLE> ... </TITLE>
- </HEAD>
- <BODY>
  …
- </BODY>
- <HTML>.

As you can see from the above list of common tags, they are classified into:

- Opening Tag
- Closing Tag.

In the HTML code in the above figure, do you see the above tags? Yes, however, you need to scroll down the document to see some of the closing tags.

See the brief explanations of the above common tags:

HTML: This encloses the entire HTML document.
HEAD: Encloses the header area of the document
TITLE: Indicates the title of the document
BODY: Encloses the text of the document.

Having seen the tags every HTML document must have, below are a set of tags that can be used to define headings for a range of sizes. The heading tags generally start with <H... followed by a number between 1 and 6. You will demonstrate these as seen below:

HTML>
<H1> National Open University </H1>
<H2> School of Science and Technology </H2>
<H3> Department of Computer Science </H3>
<H4> Rivers State Study Centre </H4>
<H5> Dr. Sunday A. Reju </H5>
<H6> Visiting Lecturer </H6>
This is a Test Page
<HTML>

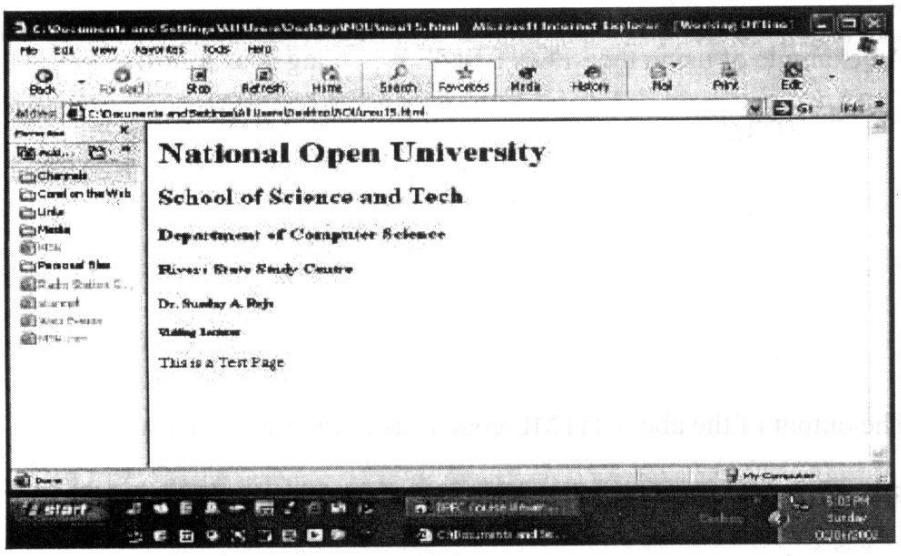
SELF ASSESSMENT EXERCISE I

Type the above into an HTML document and open the file with a browser.

Answer

The output, using Microsoft Internet Explorer is as seen in the following figure:

Figure 3



In terms of sizes, the heading tags have the following points:

<H1> : 24 point
<H2> : 18 point
<H3> : 14 point
<H4> : 12 point
<H5> : 10 point
<H6> : 7 point

Normal body text : 12 point.

Note that the text *"This is a Test Page"*, is a normal body text in the above web page. Closely associated with the <H...> tags are the <FONT SIZE = ...> tags which help you to have more control over the size of heading text. Using the <FONT SIZE = ...> tag, there are six values the size can take as stated below:

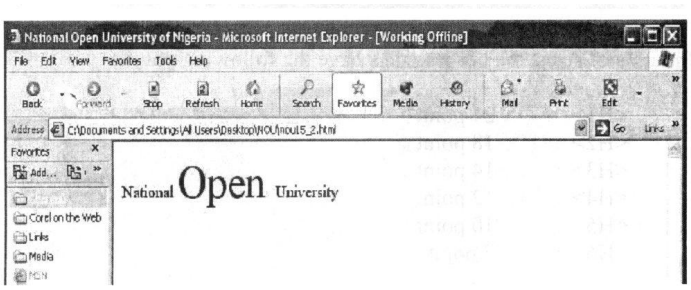
Font Size Value	Point Size	Heading Equivalent
7	36 pt	
6	24 pt	**<H I>**
5	18 pt	<H2>
4	12 pt (Bold)	<H4>
3	12 pt (Plain)	Body Text
2	9 pi	

An example of using the <FONT SIZE = ...> tag is as follows:

<HTML>
<TITLE> National Open University of Nigeria
</TITLE>
<FONT SIZE = 4> National <FONT SIZE = 7>
Open </FONT> University
</HTML>

The output of the above HTML code is as in the figure below:

Figure 4



You will see in the above figure that the text within the <TITLE> ... <TITLE> tags is the title of the web page itself.

This unit will be rounded up now while you will be privileged to see more tags in the next unit.

4.0 CONCLUSION

This unit has introduced you to the dynamics of web pages and how they are created by using the HTML (i.e. HyperText Markup Language) which was developed by the Physicist Tim Berners-Lee, as a way to easily cross-reference text on the Internet through hypertext links

The unit has specifically shown you the roles of TAGS in HTML document and a few tags that are used to format text on web pages.

5.0 SUMMARY

You need to know how web pages work so as to have a good foundation in using the programming tool used for their creation, namely, the HTML. The unit has therefore shown you various types of elements that can be embedded on a web page. These are:

- Text
- Graphics
- Multimedia clips such as video and sound
- Links
- Forms
- Frames
- Image Maps
- Java Applets.

The use of tags in HTML document has been discussed in this unit with the most common tags being identified.

To view an HTML document you need a Web Browser such as the Microsoft Internet Explorer, Mozilla, etc, as used in this unit. In the next unit, you will be introduced to additional tags and HTML commands.

6.0 TUTOR-MARKED ASSIGNMENTS

1. a) Identify two reasons why web pages experience some delays before appearing on the computer screen.
   b) What does a Web Browser do?
2. Among the following, which are the components of a web page

that are directly contained in the associated HTML document?

- HTML codes
- Text
- Sounds
- Video
- Links to other files
- Graphics.

3. Give examples of software necessary to view video and listen to the sounds embedded in web pages.

7.0 REFERENCES/FURTHER READINGS

Bride, M., HTML Publishing on the World Wide Web, NTC/Contemporary Publishing, 1998.

Microsoft Corporation, Internet Explorer Version 6.0, 2001. Microsoft Corporation, Front Page 2000, 1999.

MODULE 4

UNIT 1 FURTHER TEXT FORMATTING AND LINKS IN HTML

CONTENTS

1.0 INTRODUCTION

In the last unit, you were introduced to some essential tags used in HTML document. More specifically, the tags every HTML document must have were studied. However, there are more tags beyond those already considered in the last unit. You will therefore be using additional tags in this unit. The unit will also introduce you to special tags that help you to move from one web page to another. Now, look at your study objectives for this unit.

2.0 OBJECTIVES

By the end of this unit, you should be able to:

- Use more HTML tags for formatting text on web pages
- Identify some other HTML commands for handling web page components.
- Explain how to link web pages using the appropriate HTML commands.

3.0 MAIN CONTENT

3.1 Additional Tags for Text Formatting

In this section, you will learn about the following:

- Paragraphs and Line Breaks

- Text Alignment
- Text Emphasis
- Text Colour

Paragraphs and Line Breaks

<P>:

The tag for setting up a paragraph is the **<P>** tag. The tag marks the beginning of a new paragraph and places a blank line before it. The **<P>** tag can be placed at the end of a piece of text, at the start of the next or in between.

**
:**

The above is the line BREAK tag and marks the start of a new line.

<HR>:

The tag <HR> stands for Horizontal Rule and this is used to separate paragraphs by drawing a line between them. The line is usually a thin line with a shaded effect and extends through the width of the window.

Text Alignment

To align text, HTML uses the keyword ALIGN inside the <H...> or **<P>** tag followed by "Center", "Left" or "Right". Body text and headings are usually aligned left, hence the left alignment is optional.

Examples of HTML code using ALIGN are as follows:

<H4 ALIGN = "Center"> </H4>
<P ALIGN = "Right">.

•

Text Emphasis

The following tags can be used to emphasise a word or phrase in your text on any web page.

<B>… </B>	To set text to BOLD
<I>…</I>	To make text ITALIC
<TT> ...</TT>	To set text in courier font.

The above tags are 'classified as PHYSICAL TAGS, meaning that they only work if the web page visitor's browser can display bold, italic and courier fonts. However, see the next tag.

<STRONG> ... </STRONG>: To set text to bold, like <B>

<STRONG> is an example of a LOGICAL TAG, that is, a tag whose effect can be redefined at the receiving end.

<BLINK> ... </BLINK>: This makes text to flash to catch viewer's attention.

Text Colour

You can set the colour of the background and of the text by including either or both phrases in the <BODY ... >tag as follows:

<BODY BGCOLOR = value TEXT = value> ... >

The colours are usually set by using what look like 6-digit hexadecimal numbers. The numbers are meant to set the brightness of the RED, GREEN and BLUE components of a colour. Using a 24-bit colour display, each of the numbers can be between '00' and 'FF'. However, practically, the light values are taken as follows:

00 : for off
80 : for dipped colour
FF : for full beam colour.

Below is a table that summarises major colour values for the RGB components

R	G	B	Colour
00	00	00	Black
FF	00	00	Bright Red
00	FF	00	Bright Green
00	00	FF	Bright Blue
80	00	00	Dark Red
00	80	00	Dark Green
00	00	80	Dark Blue
FF	FF	00	Bright Yellow
80	80	00	Brown
FF	00	FF	Magenta
80	00	80	Indigo
00	FF	FF	Bright Cyan
00	80	80	Turquoise
FF	FF	FF	White
80	80	80	Grey

For example, look at the following HTML code line:

<BODY TEXT = FF0000>

The command sets the text colour to Bright Red.

To set the colour for a section of text use the keyword "COLOR =" inside a <FONT...> tag like the one below:

<FONT COLOR = FF00FF>

Sometimes, the above can be written as follows:

<FONT COLOR = "#FF00FF">

This is just used to make the colour values to be easily seen, but neither # nor the quotes are necessary.

SELF ASSESSMENT EXERCISE 1

How do you insert comments in HTML source code>

Answer

Comments are written inside <!...> tags.
You will now see the use of links in the next section

3.2 Links Within HTML Document

Generally, links can be used for:

- hypertext
- graphics.

Hypertext links are what HTML is all about, majorly.

The keyword for links is HREF

The word stands for Hypertext REFerence and it identifies the target page or point within a page. But you cannot use HREF alone, you need to ANCHOR it to a piece of text or a graphic so that there is a web page element to click on to activate the link. The ANCHOR TAGS are:

<A ...> and </A>

The tags mark the beginning and the end of the link text. An example is as follows:

<A HREF = NOU15.HTML"> NOUN Courses </A>

The above link uses a file (NOU15.html) stored in the same directory while the text 'NOUN Courses' is the text that will be underlined when viewed in your browser and can be clicked to link you to the web page (NOU15.html).

Another example of a link is below:

<A HREF = "http://www.yahoo.com"> Yahoo!</A>

This link connects you to the Home Page of the Yahoo! Directory, while 'Yahoo!' will be the word *underlined* as the link. Your computer must have an Internet connection to be able to connect to such a website page. What about linking an image?

To link an image, the HTML tag is as follows:

<IMG SRC = "filename">

<IMG SRC> stands for "Image SouRCe" and the tag will place your image against the left edge of your page, directly after any text, and with later text starting to its right. To change the positioning of the image, you need to use the tag with ALIGN. An example is as follows:

<IMG SRC = `NOU.gif" ALIGN = top>

To set the vertical position in relation to surrounding text, there are three ALIGN values:

- TOP
- MIDDLE
- BOTTOM

Generally, BOTTOM is the default position, that is, the accompanying text is placed at the bottom of the image.

The CENTER position can also be used for image positioning in the following form:

<CENTER><IMG SRC = "filename" ><CENTER>

Before rounding up this unit, see the following example:

```
<HTML>
<TITLE>
NOU Home Page
  </TITLE>
<HI> National Open University </HI>
<HR>
<I> School of Science and Tech </I>
<A HREF = "Programs.htmln"> Programs </A>
<HTML>
```

The output of the above HTML source code is as follows:

Figure 1

When the mouse pointer is on the link, the target is shown on the status bar at the bottom of the window as seen in the above figure.

You will now round up this unit.

4.0 CONCLUSION

This unit has shown you additional tags for formatting text and also for linking web pages and images.

As seen in this unit, the web page background colour and the colour of a text can also be set by using appropriate tags. To set colours, you need to know the colour values for various combinations of RGB (RED, GREEN and BLUE) components. With all that you have learnt in this unit in addition to what was covered in the last unit, you can now get started with programming with HTML.

5.0 SUMMARY

As seen in this unit, all you need to know in HTML is the appropriate tags to be used for various types of displays expected on your browser. This unit has extended the list of tags studied in the last unit.

Most importantly, the tags used for connecting web pages and files have been introduced to you in the unit. There are some other tags used in HTML which are not covered in these two units, but the first reference under 'References/Further Readings' will help you to understand more of the HTML tags.

Remember that your HTML code can be typed using any text editor, but saved with .html extension. There are also special software packages meant for website development such as the Microsoft Front Page 2000. You should interact with it.

The next unit will introduce you to another language.

6.0 TUTOR-MARKED ASSIGNMENTS

1. Explain how to do the following:
   a) How to adjust the size of a graphic.
   b) How to put a border around an image.
2. Define the following:
   - Logical tags
   - Anchor tags.
3. What tag is to be used to make sure that the BOLD text is properly displayed on the website visitor's browser?

7.0 REFERENCES/FURTHER READINGS

Bride, M., HTML Publishing on the World Wide Web, NTC/Contemporary Publishing, 1998.

Microsoft Corporation, Internet Explorer Version 6.0, 2001.

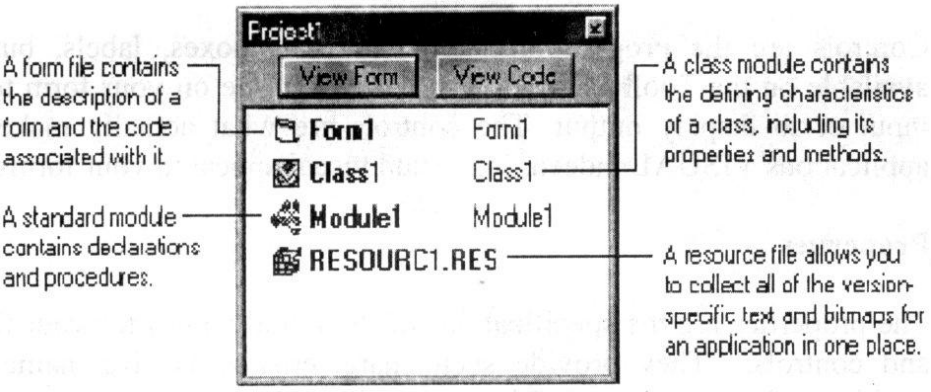
Microsoft Corporation, Front Page 2002, 2001.

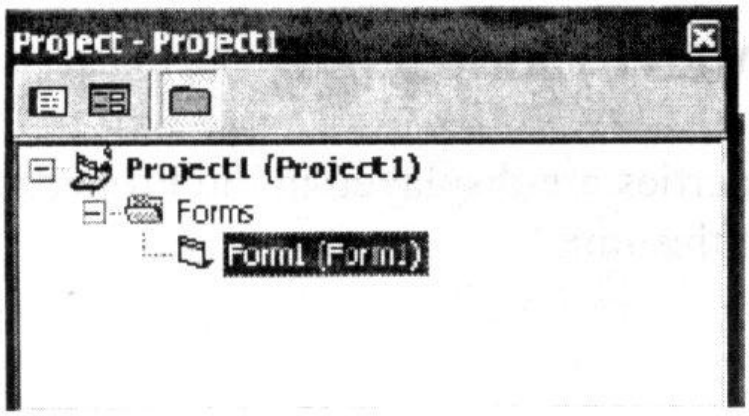
hidden windows you can load on your screen by clicking their associated icons on the Toolbar or from the Menu bar options.
The following need to be defined for you as major features that characterize the VB programming environment:

- Project
- Form
- Controls
- Properties
- Code.

Project

A PROJECT is a collection of files you create that makes up your Windows application created by Visual Basic. Precisely, it is a collection of the forms, program modules and resource files that make up an application. Your Project Explorer will usually list these files as you continue your programming development activities. See examples of Project Explorer Windows below:





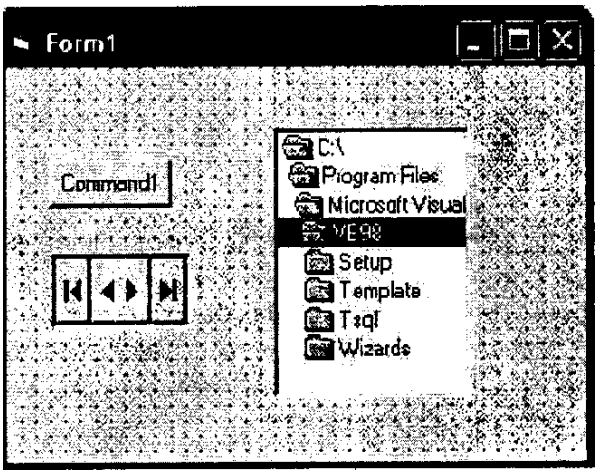
The first window above is the Project Explorer Window for VB Version 4.0 while the one below is for Version 6.0.

Form

A FORM is simply the window you create which includes the controls and the code associated with that form.

An example of a form with three controls is seen below:

Figure 3



Controls

Controls are the programming tools such as boxes, labels, buttons available on the Toolbox window which you place on your form to get input or to display output. The controls are what actually make VB applications VISUAL indeed. They add visual appeal to your forms.

Properties

The properties are the specifications of the initial values for your forms and controls. They provide such characteristics as size, name and position of your objects. The properties can be set by using the Properties Windows.

SELF ASSESSMENT EXERCISE 1

What object properties are displayed in the Properties Window shown in the first figure of this unit?

Answer

The properties of the current form, i.e. Form 1.

Code
A code is just the name given to the programming statements you write and associate with the controls on your form. You will see examples of codes in the next unit.

3.2 Programming Tools in Visual Basic

Having shown you the general features of VB programming environment, this section is just to introduce you to few tools among many that are available in VB language programming.
Look again at the following screens

Figure 4

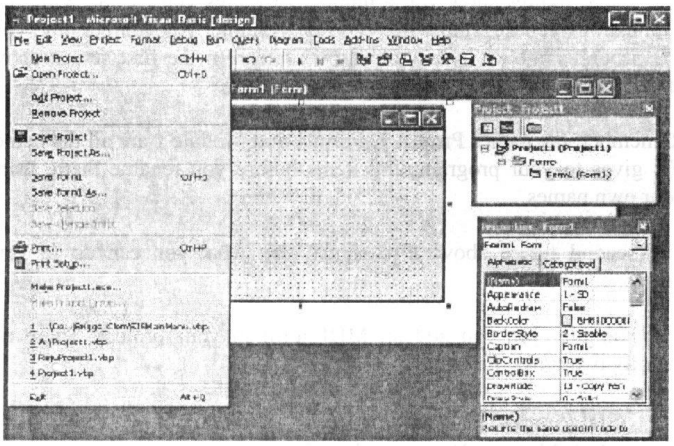
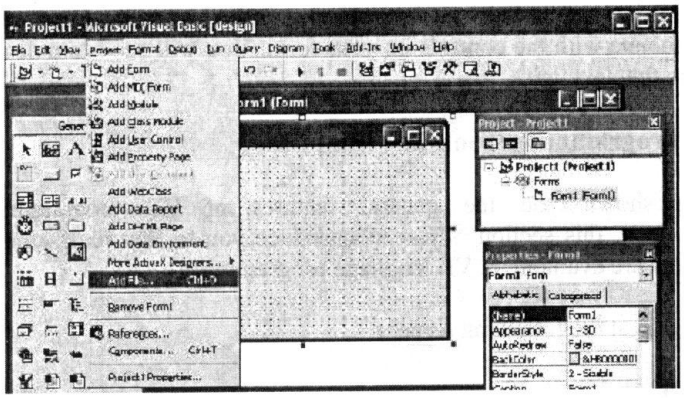


Figure 5



The above figures show you a number of things you can do with your applications you develop with VB. For example, you can compile your VB program into an executable form as seen in the first screen above under the File menu option: "Make Project 1.exe"

Remember, the names Project 1, Form 1 and Module 1 are all the names VB gives to your programming items before you rename them,

using your own names.

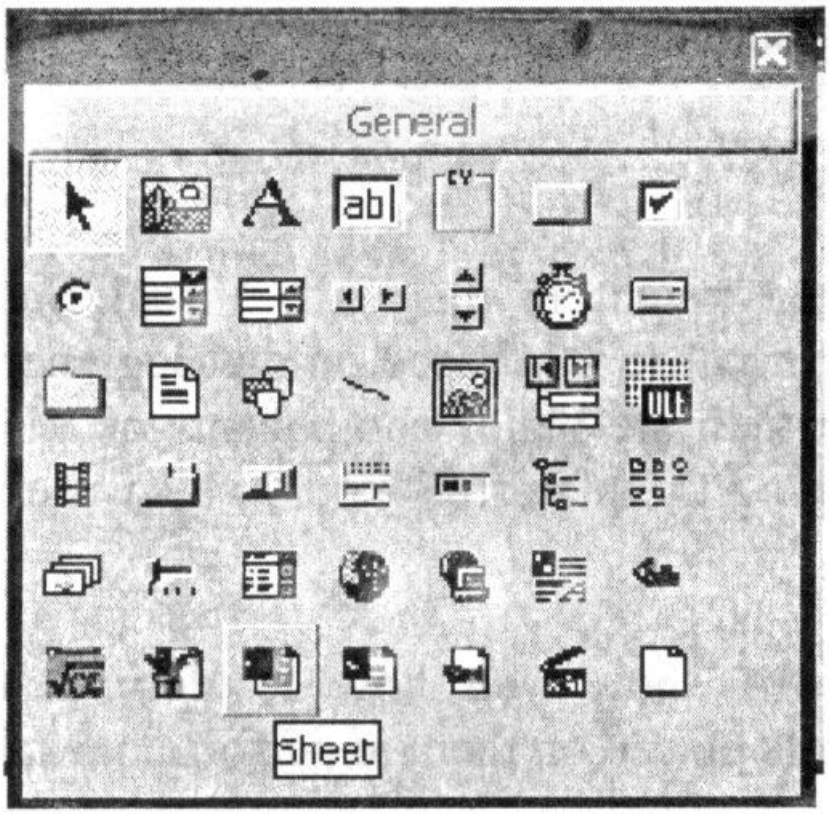
The second figure above also shows you what you can do to your Project components.

For example, you can add an MDI form to your project. There are majorly two types of forms.

- SDI (Single-Document Interface)
- MDI (Multiple-Document Interface).

An SDI application is one that requires only a single data window while an MDI application allows the opening of multiple data documents within the same application. An example of SDI application is the Windows Notepad while an example of MDI application is Microsoft Word or Microsoft Excel.

Now, look at the Toolbox below:



The controls available on your Toolbox are what make things happen on your forms. You will be using some of these controls in the next unit. Before then, you need to know of an essential concept in Visual Basic.

Events

Generally, the user interface of a Visual Basic application you create with VB is made of OBJECTS. The objects are the forms and controls you use to enable your program users to enter data and view information. Now, each of the objects you create recognizes actions users perform, such as clicking a button, opening a form or typing in a text field. These actions are called EVENTS.

Thus eprograms developed within VB environment are "Even-Driven" programs. You will round up this unit here so as to get properly started with VB simple programming in the next unit.

4.0 CONCLUSION

This unit has given you a brief but essential overview of Visual Basic language programming. In the unit, you have seen the basic things you handle within VB programming environment. Such things are:

- Project
- Forms
- Controls
- Properties
- Code.

As you have already learnt in the unit, VB programs are Event-Driven programs, and this is what makes the language a very powerful tool in the hands of programmers.

5.0 S UMMARY

As you have been taught in this unit, Visual Basic is a very easy language to learn because it is based on the Beginners language, that is, BASIC. Forms, which are major components of VB Project are where things happen, being the platforms upon which users can interact with the computer.

The next unit will show you how to design simple forms with appropriate controls placed on them to perform one task or the other.

6.0 TUTOR-MARKED ASSIGNMENTS

1. Define the following:
   - Objects
   - Project
   - Controls.
2. Distinguish between SDI and MDI and give 2 examples each of

the applications that allow Form windows in their mode.

3. Describe the similarity of capabilities of VB and C++

7.0 REFERENCES/FURTHER READINGS

Microsoft Corporation, Visual Basic Programmer's Guide, Version 3, 1993.

Microsoft Corporation, Visual Basic Version 4.0, 1995.

Microsoft Corporation, Visual Basic Version 6.0, 1998.

UNIT 3 DEVELOPING SIMPLE VB PROGRAMS

CONTENTS

1.0 INTRODUCTION

This unit will make use of the basic knowledge you have already got from the previous unit to get you started with simple programming using VB. The unit will specifically teach you how to create simple forms and place some controls on them using the Toolbox.

The unit is also going to introduce you to the code environment of VB and how to hook up your controls with their associated event-driven code. A major feature of most Windows application is the Menu facilities. In this unit, you will learn without rigour how to design simple menus on your forms.

In Visual Basic programming, there are WIZARDS that can automatically help you to create your applications. Wizards are Question-and-Answer dialog boxes that automate tasks. For new programmers, it is advisable to learn how to create applications without the wizards so as to have control over every aspect of the design and to also learn many things about the language. See below your study objectives for this unit.

2.0 OBJECTIVES

By the end of this unit, you should be able to:

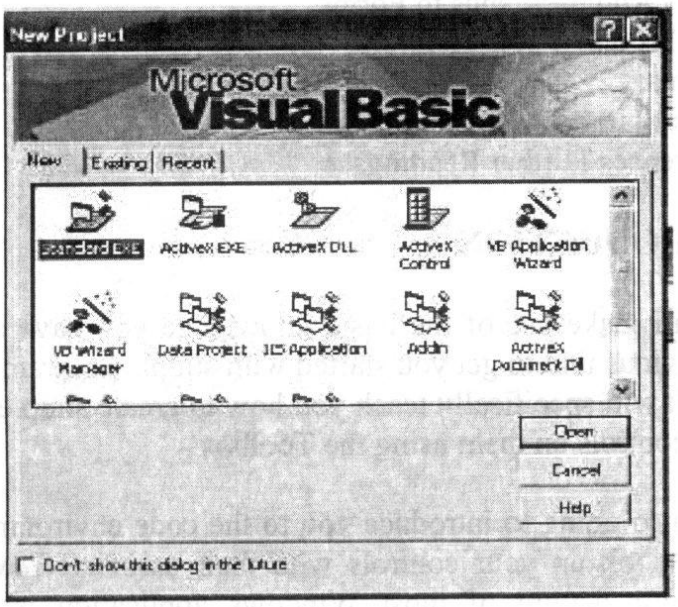
- Use a number of controls on the Toolbox
- Set properties for forms and controls in a VB application.
- Write simple code for controls placed on forms
- Design simple menus on forms.

3.0 MAIN CONTENT

3.1 Writing Event-Driven Code

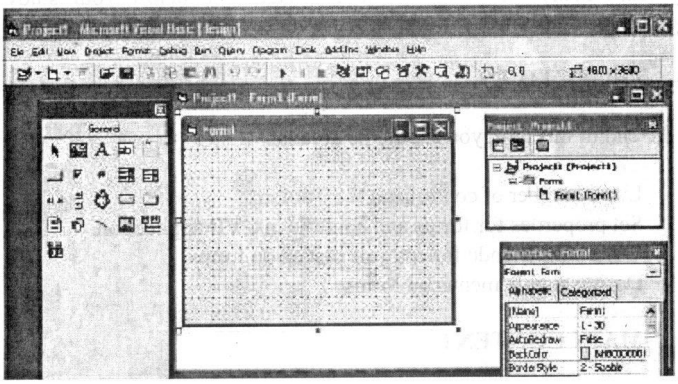
You will start first of all by seeing below what you are expected to see when you start Visual Basic programming environment. See the following figure:

Figure 1



As seen above, to start a new application from scratch, you have various options of types of application you want to select from. You are to double-click the "Standard Exe" option, after which a blank Form window will appear for you to begin creating your application by placing the controls on it.

Figure 2



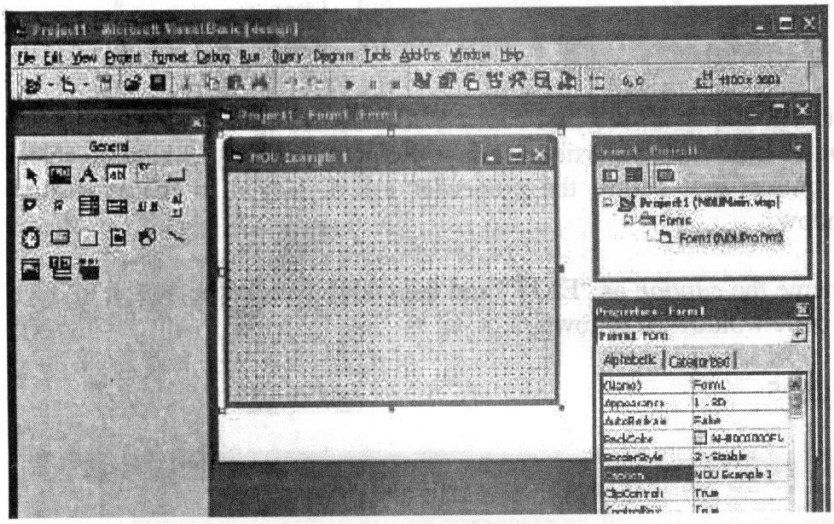
Before you go ahead, do the following:

- Use "Save Project As" from the File menu to name your project and the Form 1 file

- Use the Properties Window to rename the Caption as NOU Example 1.
-

Doing this, you will have the following screen:

Figure 3

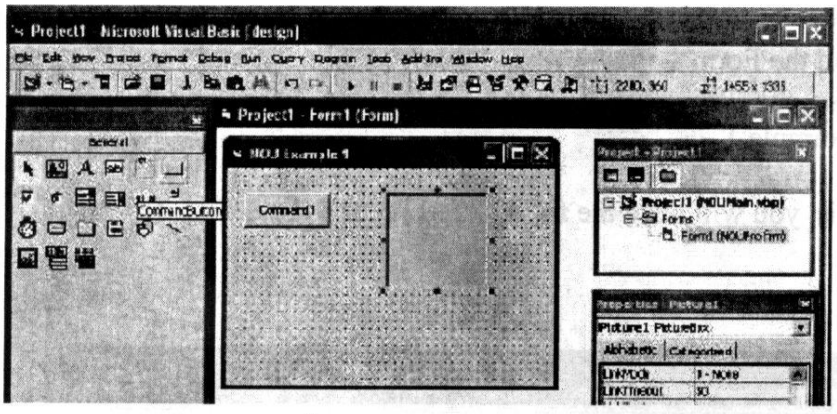


Now, get started by placing two controls on your form:

- The Command Button
- The Picture Box.

These controls are on the first row of the Toolbox. In fact you will see then name of the control displayed by just pointing to the control. On placing these controls, you have the following figure:

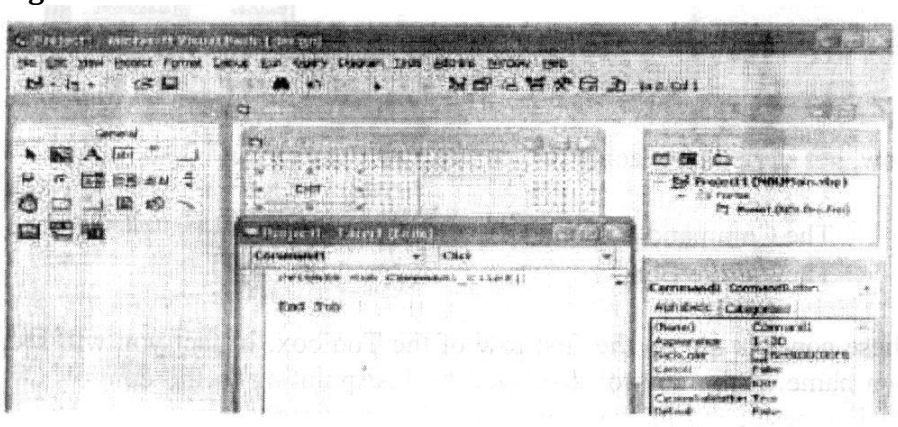
Figure 4



You will start with the first control, "Command 1" button. Rename the caption from the Properties window. Remember that you have to select the button first before the properties are displayed in the Properties window.

Rename the *caption* as "EXIT" and then double-click the button to open the code window as follows:

Figure 5

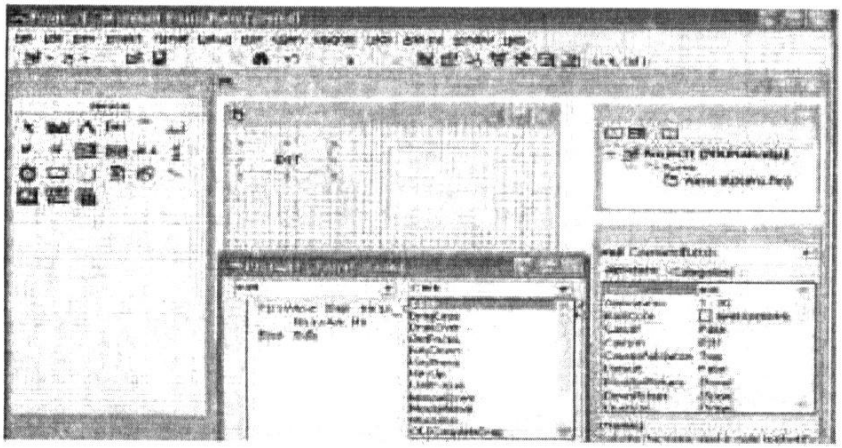


You are now about to write your first event-driven code. Visual Basic has made your work easy by already giving you the Header and the Terminator for your program segment:

```
Private Sub Command I — Click ( )
.
.
.
End Sub.
```

From the Properties window, rename the *Name* property to "exit" and type the VB command *"Unload Me"* as the event statement. The name of the Event is "Click" and to select any other type of event, you can click the downward arrow as seen below.

Figure 6



SELF ASSESSMENT EXERCISE 1

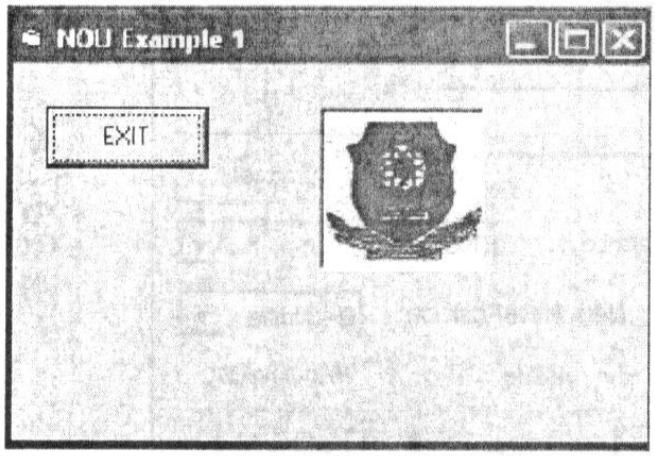
What do you think the above code will perform on clicking the button EXIT?

Answer

The action will close the Form window.

Now, for the second control on your form, select the control, and from the Properties Window, click the button in front of the *"Picture"* property to select your picture to place on the Form. Locate your picture from your directory where it is stored as indicated below:

Figure 9



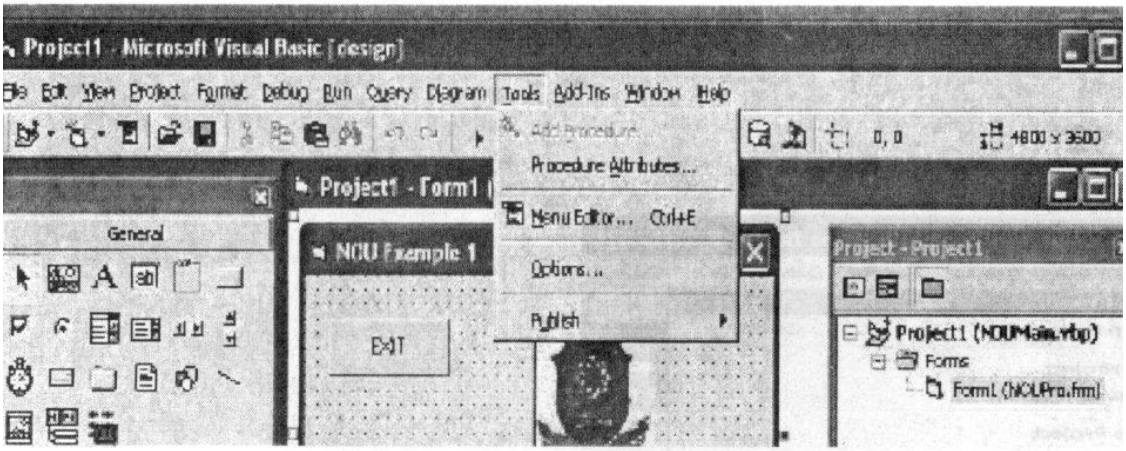
Above is the output of your application. Clicking the EXIT button will close the "NOU Example 1" window that you have created.

Since you only want to display the picture object, you may not need to add any code to the control.

3.2 Adding Menus to Forms

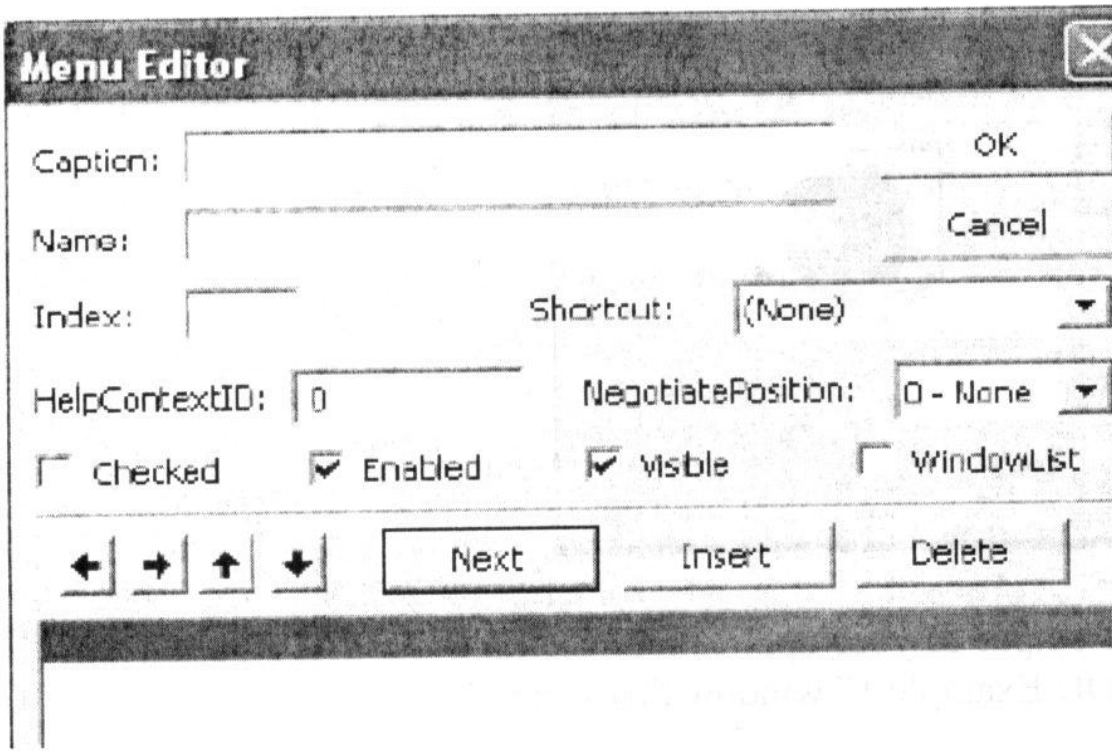
To add menus to your form created in the last section, it is very easy. In this case, you will need the "Menu Editor" which you can access from the Tools menu or the toolbar as shown below:

Figure 10



Now, open the Menu-Editor window as shown below:

Figure 11



Before you add your menus, it is good to state that Menus consist of the following:

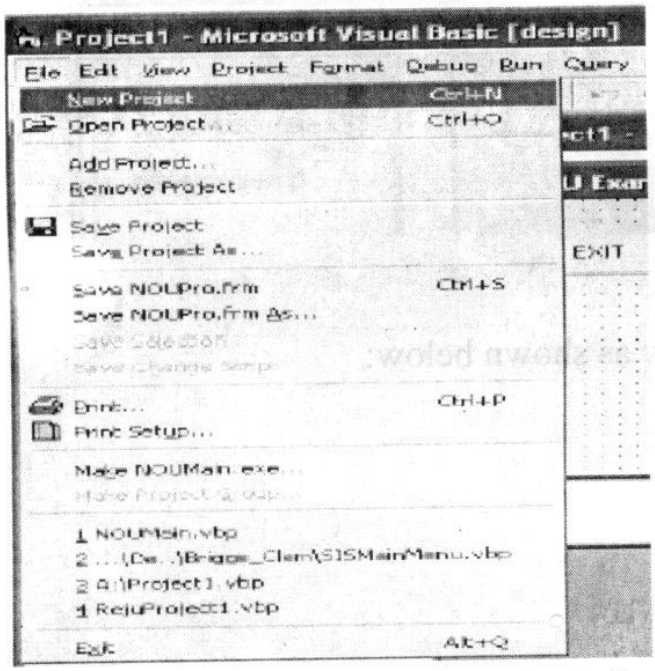
- Menu Title
- Menu Item
- Separator bar

Every part of a menu is a menu control and as a result, a menu has predefined set of properties and events.

Every part of a menu is a menu control and as a result, a menu has predefined set of properties and events.

For example, look at the menu below:

Figure 12



The Menu Tile is "File"

The Menu Items are:

- New Project
- Open Project

  .
  .
  .
- Exit.

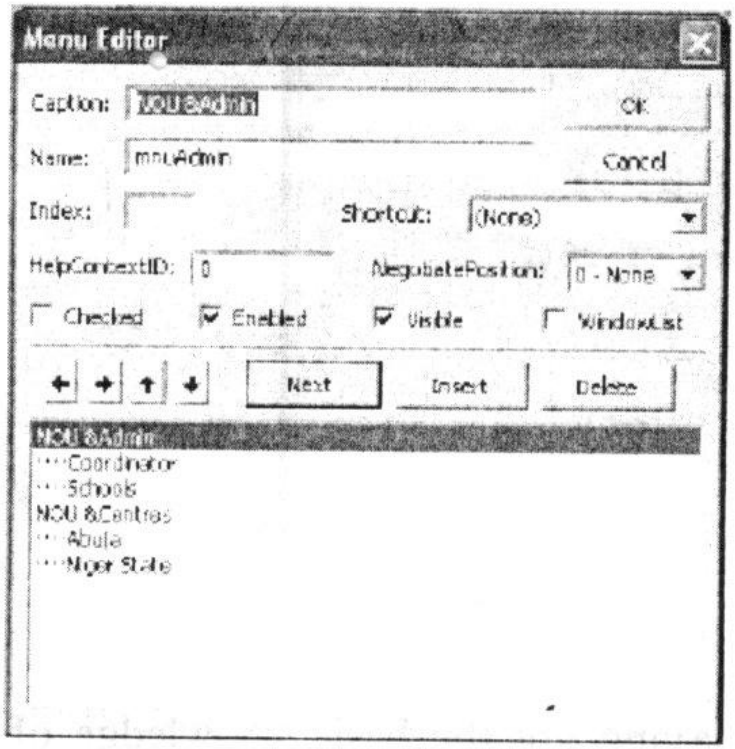
You can see that there are a number of separator bars.

Now, back to the Menu Editor, you have these three essential properties:

- Caption — this is to specify a menu title.
- Name — this is the name used to refer to the menu control in the associated code.
- Index — this is a numeric value that uniquely identifies the menu control if it is taken as a control array part.

Set the properties as shown below:

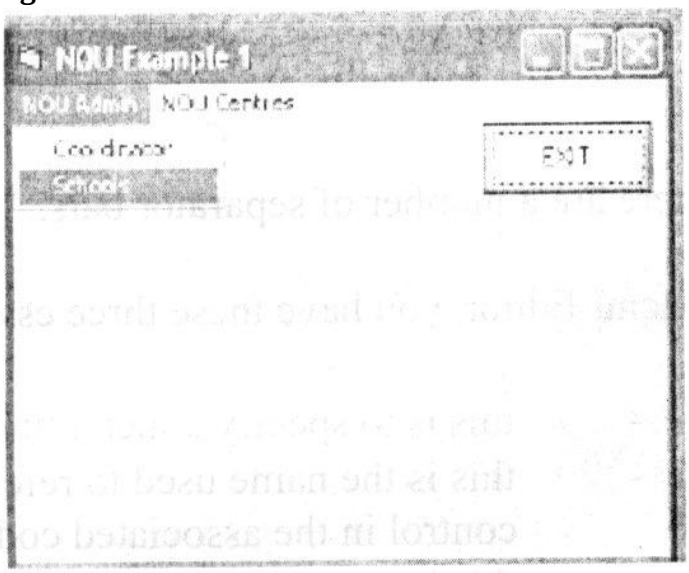
Figure 13



As seen in the above Menu Editor, the Menu captioned "NOU Admin" is expected to have two menu items — "Coordinator" and "Schools". You can use the Indent arrow to add the menu items. The character "R" placed between the characters in the title is to specify what letter to use with the ALT key to select the me nu. For example, by pressing ALT + A, you can select the "NOU Admin" menu from the menu bar.

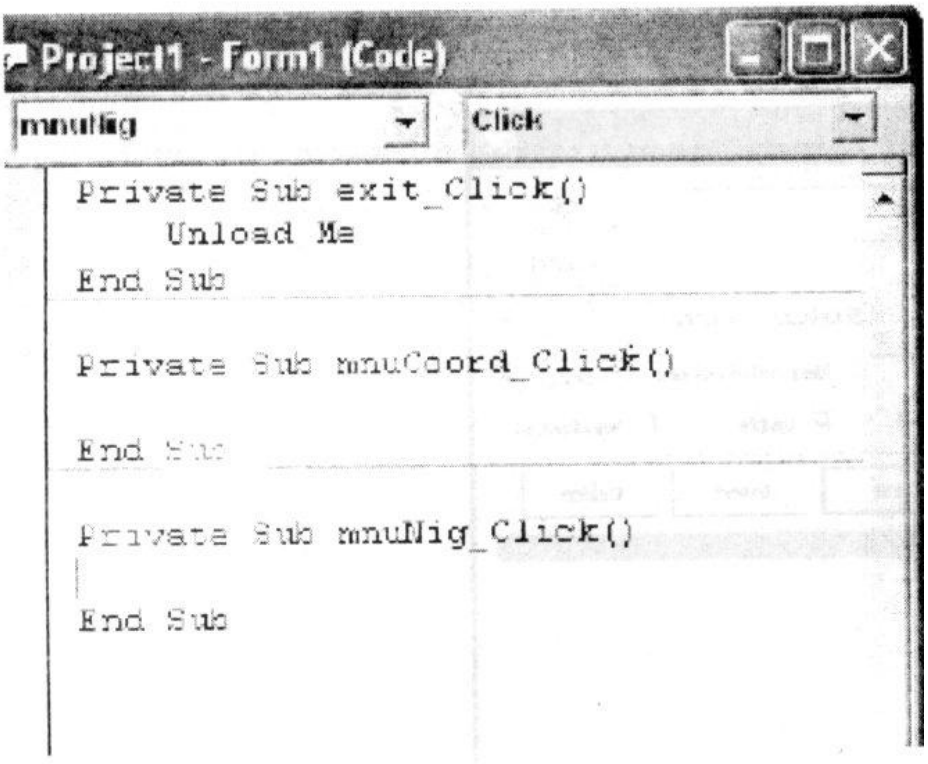
If you now select the Run command from your VB toolbar, you have the following form:

Figure 14



To add code to the menu items, simply click the menu item on the form to open the code window as follows:

Figure 15



You will now round up this unit having got the basic knowledge of creating a simple application with Visual Basic.

4.0 CONCLUSION

This unit has taken you through the few basic steps you need to follow to get started with writing event-driven programs when creating a VB application. You have seen in the unit how it is easy to place controls on your forms and how to hook up the controls with code that is expected to drive them.

You have been using menus in many applications you have interacted with such as Microsoft Word. Now, you can create your own menus as seen in this unit.

5.0 SUMMARY

The unit is not aimed at taking you through the many lists of VB keywords or operators as in other languages. You are already acquainted with some of them when you studied BASIC language. This unit has simply taught you how to add code to your controls and menus. You have seen how it is very easy to develop applications with VB.

You just need to get started with VB programming now while all other helps you need to extend your knowledge can be obtained from the "Books Online" (or MSDN CDs of your Visual Studio setup CDs) which you can access from the Help menu of your fully installed Visual Basic programming environment.

6.0 TUTOR-MARKED ASSIGNMENTS

1. Name two of the following:
a) Event Names
b) Project components.
2. State at least 8 controls you can place on your forms.
3. What do you do to access information or data in an existing database using a VB application?

7.0 REFERENCES/FURTHER READINGS

Microsoft Corporation, Visual Basic Version 6.0, 1998.

UNIT 4 PROGRAMMING WITH MATHCAD CONTENTS

1.0 INTRODUCTION

In this unit, you are going to extend your knowledge of various programming languages you have learnt in this course into that of some special software developed for programming purposes. In fact, most of these packages have their own way of program coding, call it special languages. However, their programming keywords, operators and statements are very close to most of the languages you have already studied. Some even have exactly the same operators used in some standard programming languages.

In this unit, you will be introduced specifically to MathCad Package, after giving you an overview of some of these special programming software. Your study objectives in this unit are seen below.

2.0 OBJECTIVES

By the end of this unit, you should be able to:

- Identify some specialized software for programming.
- Explain the features of MathCad Software.
- Use MathCad to solve some problems.

3.0 MAIN CONTENT

3.1 Introduction to Special Programming Software

Specialised programming software are usually developed to handle a class of common programming problems. Some common examples of these applications are as follows:

- MathCad
- Matlab
- Mathematica
- Statistica

- SPSS
- Microsoft Access
- Microsoft Front Page.

For example, from the above list, some of these packages have been treated in a course on *Software Application Skills* in the School of Science and Technology. They will therefore not be treated in this course again. Specifically, Microsoft Access and SPSS have been treated in that course. You may wish to refer to your course material if you took or are taking the course. Microsoft Access for example is a very good programming software for database problems and interestingly enough, Visual Basic that you have just studied in the previous two units have common programming tools with Microsoft Access.

Similarly, Microsoft FrontPage is a very good programming tool for development of Web Pages. In fact, there are wizards in FrontPage to help you create your HTML code very fast.

Statistica is another good programming tool for statistical analysis with more analysis tools than that of SPSS.

MathCad, MatLab and Mathematica are special programming software for advanced Mathematical and Engineering problems. You need to have some lnowledge of these applications to complete your broad knowledge of programming tools aimed at covering in this course.

Something so unique about these specialized programming applications is their capabilities to plot graphs and handle large data problems with ease. You only need to master few commands associated with their languages to see yourself solving complex problems.

Below are the figures to show you some of the programming environments for three of the applications listed above.

Figure 1 (MathCad):

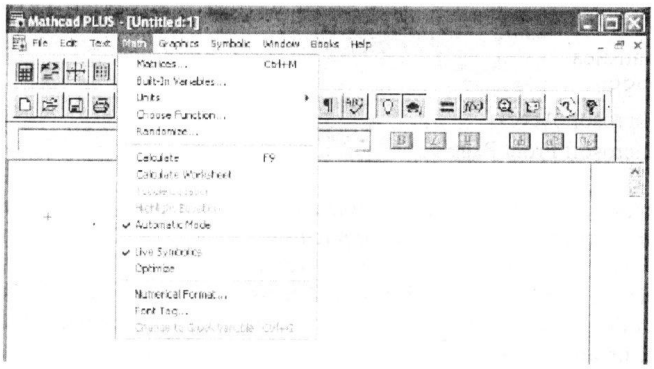


Figure 2 (Mathlab):

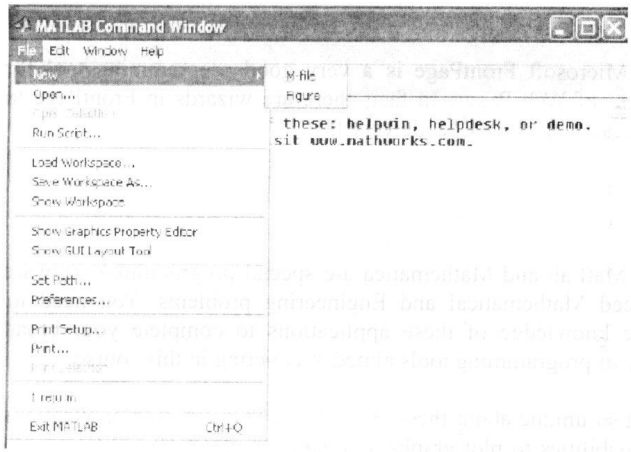
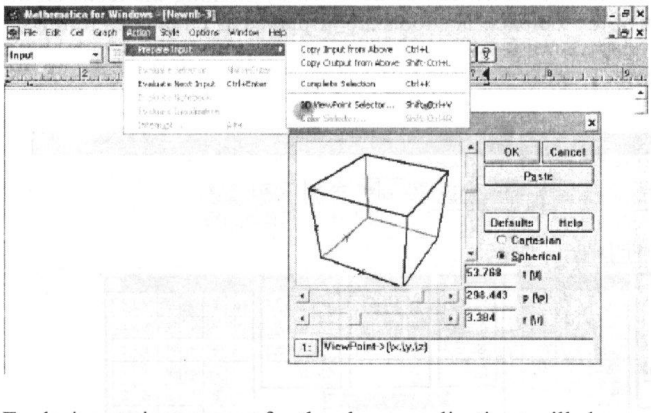


Figure 3 (Mathematica):



Exploring various menus for the above applications will show you some of their programming capabilities. However, the best way of studying the applications is to open their Help files using the Help menu.

You will now see in the next section some of the programming capabilities of Mathcad.

The Mathcad version used in this course material is the Version 6.0 of the application.

3.2 Using MathCad Programming Tools

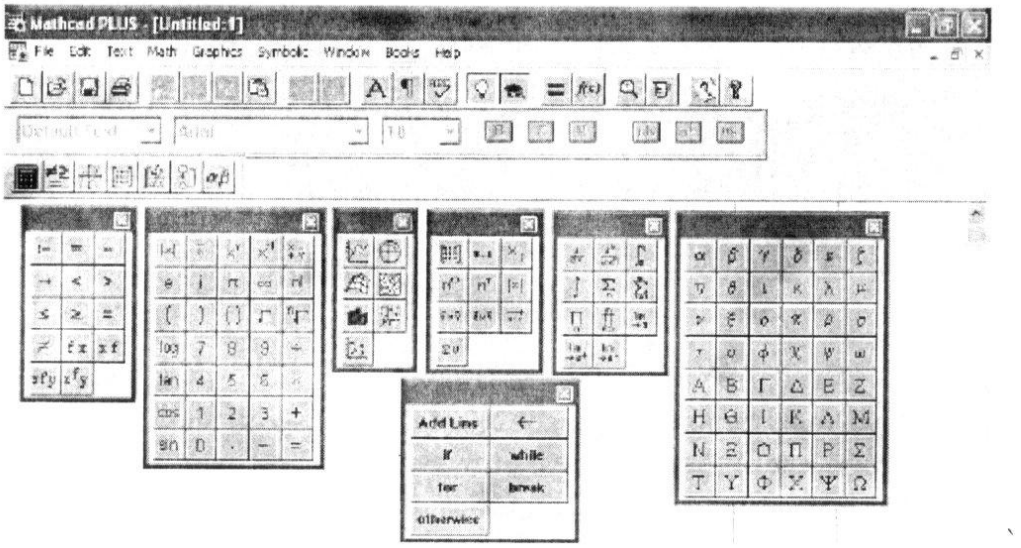
To properly understand the capabilities of MathCad, you should open the Help file from the Help menu. MathCad is a very powerful tool capable of doing the following, among other things:

- Animation
- Data Analysis
- Plotting Graphics
- Solving Equations
- Statistical Analysis
- Programming
- Symbolic Calculations
- Vectors and Matrix Analysis
- Text Editing.
- etc.

MathCad and most other specialized applications are Interpretive Programming software, meaning that they have built- in interpreter to give you the results of your programming expressions instantly.

To start with, below are some of the programming tools displayed in seven (7) windows which can be accessed by clicking the icons on the third row of the toolbars. The tools are displayed as Palettes as follows:

Figure 4



- Arithmetic Palette
- Evaluation and Boolean Palette
- Graphing Palette
  Vectors and Matrices Palette
- Calculus Palette
- Programming Palette
- Greek Letters Palette.

See the use of one of the tools on the Evaluation palette as follows:

- Position the cursor (+) at where you want expression to appear
- Select n! on the palette
- Type your number whose factorial you want to find (say 45) in the empty holder.
- Press = key on your keyboard to see the answer as follow:
-

$$45! = 1.196.1056$$

That is, 45!= 1.196 x 1056

Secondly, use the Calculus Palette to find the integral of a quadratic function between x = 0 and x = 2. The processes of carrying out these two computations are shown below with the results on the right hand side window.

Figure 5

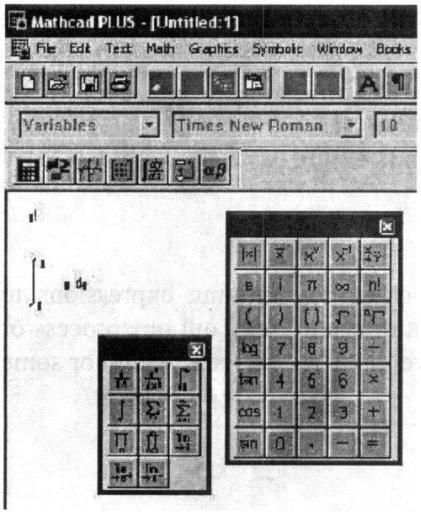
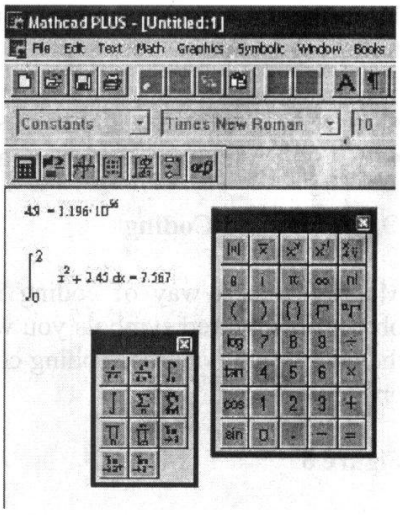
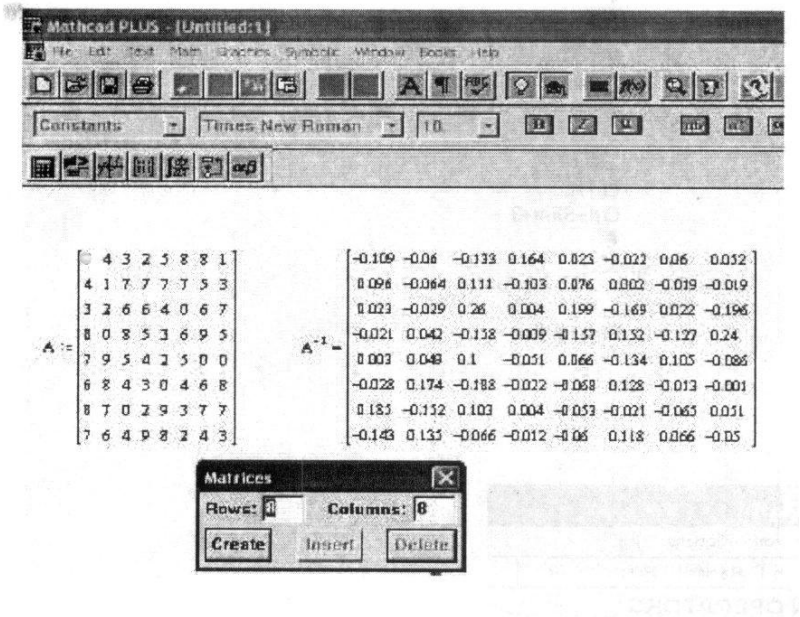


Figure 6



One more example for finding the inverse of a matrix is as follows within MathCad:

Figure 7



To obtain the- above results, use the Vectors and Matrices Palette and click the matrix button to define the rows and columns of the matrix. MathCad will give you the matrix frame to enter entries. Then simply type:

A ^ -1

to obtain A-1. Press "=" key on the keyboard to get the inverse instantly.

From the immediate example, you will observe that MathCad uses the following operators:

= for Assignment (as in Pascal)
∧for exponentiation (as in BASIC).

Operators and Coding

MathCad has a way of coding its own programming expressions to obtain the required symbols you want to use to carry out one process or the other. Some of these coding conventions are as seen below for some operators:

Figure 8

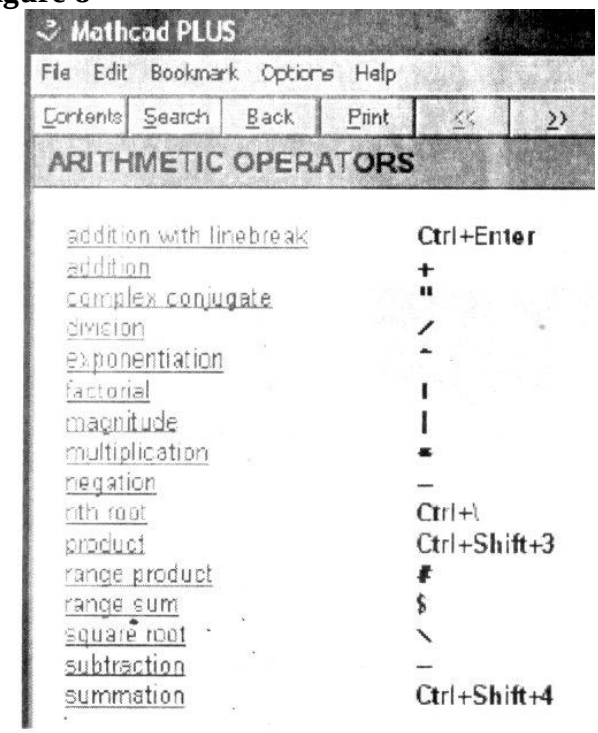


Figure 9:

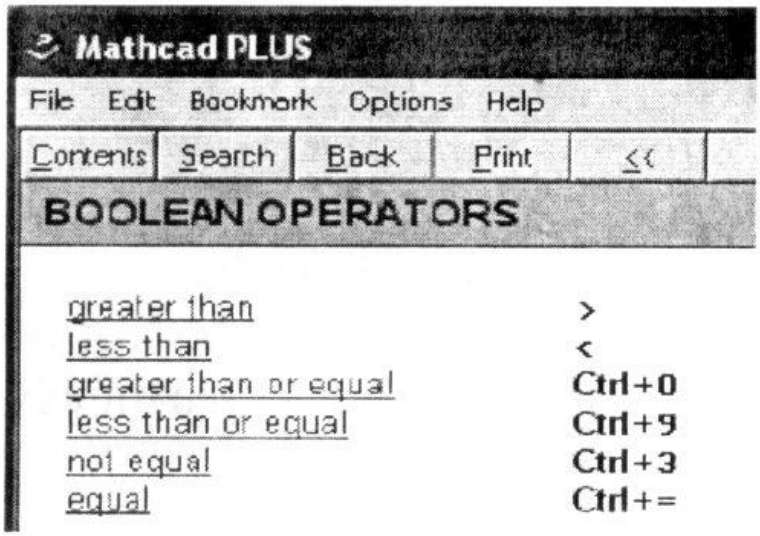


Figure 10:

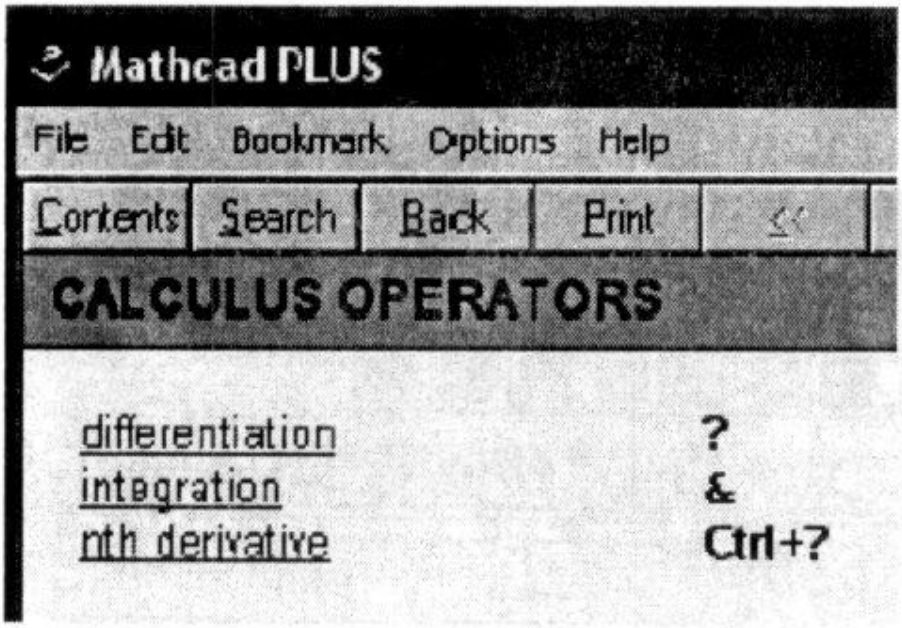
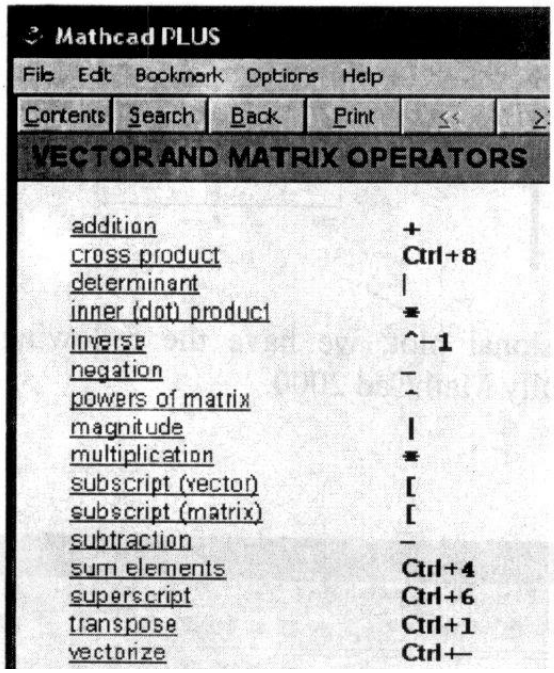
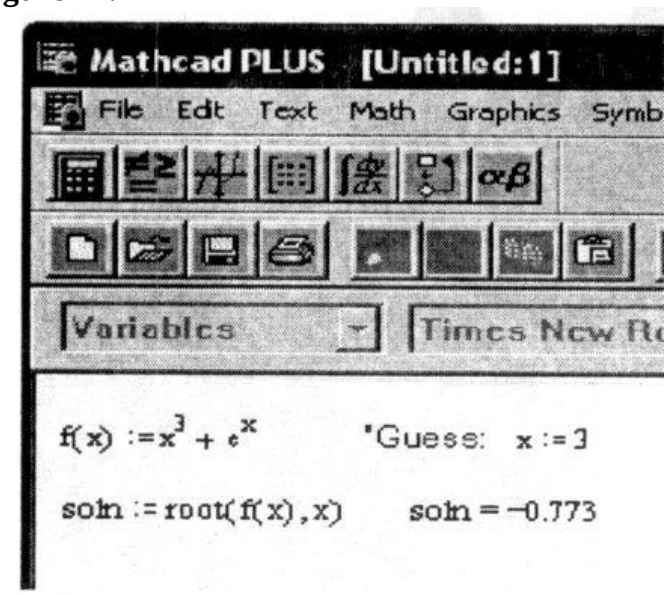


Figure 11:



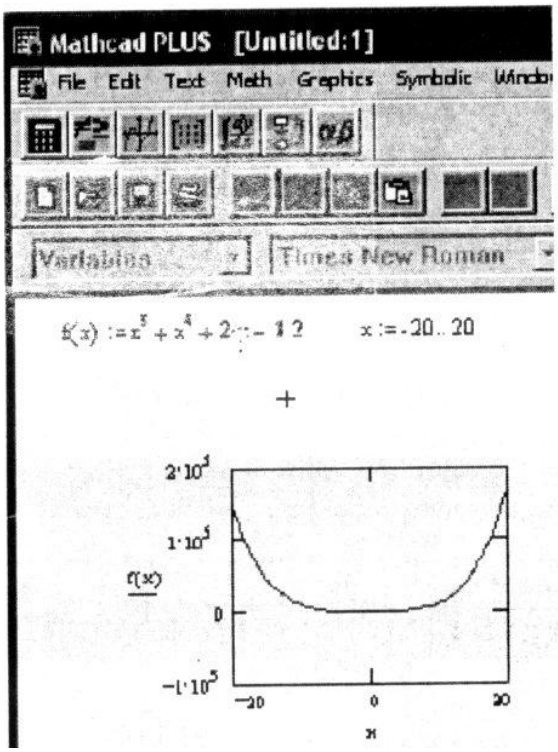
Below is a simple programming session of solving a nonlinear equation using one of the MathCad reserved words: **root**

Figure 12:



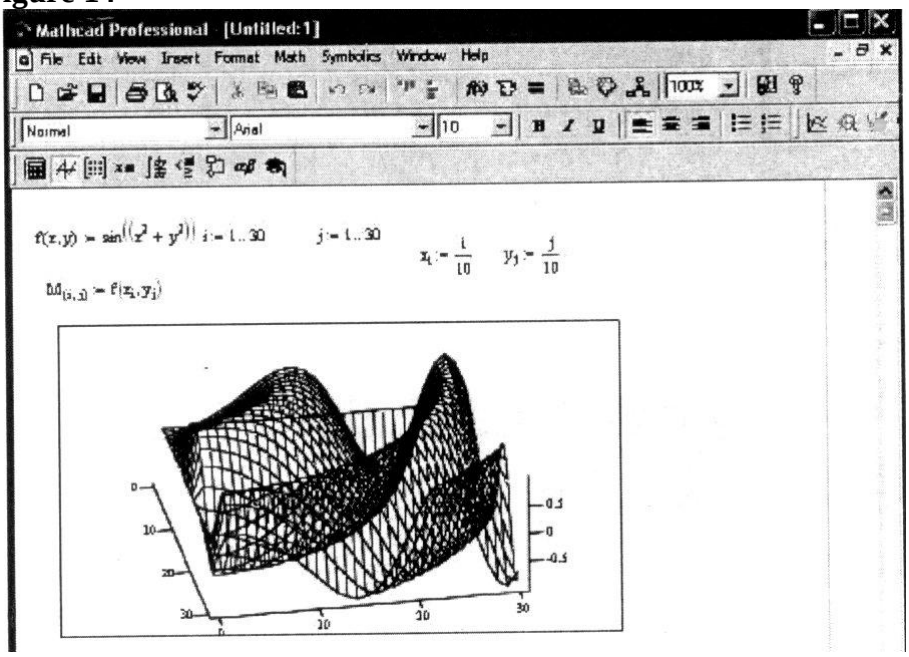
What you have below is also a graph of 2 — dimensional plot using MathCad.

Figure 13



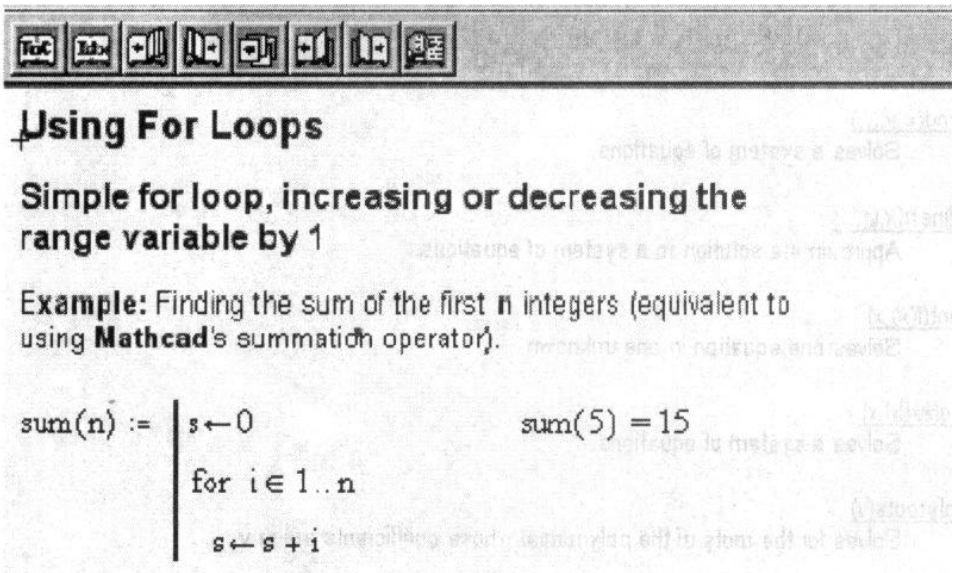
For a 3 — dimensional plot, we have the following example using MathCad (specifically MathCad 2000.

Figure 14



Finally, see a very simple MathCad program to find the sum of the first n integers:

Figure 15



You will now round up this unit.

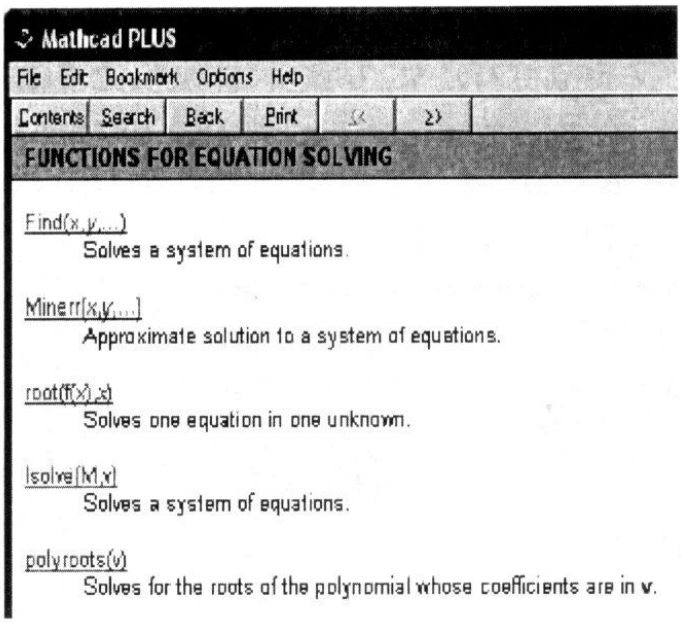
4.0 CONCLUSION

This unit has shown you the use of some specialized programming

software to solve some classes of problems. The unit specifically focused on MathCad which has a lot of tools to solve various groups of mathematical problems.

The unit showed you some examples of problem solving using some MathCad codes. As you have seen in some of the examples, MathCad has its Am built- in functions. Below are some of them summarized, for solving equations:

Figure 16



5.0 SUMMARY

This unit has introduced you to programming applications specially developed to solve problems by-passing the conventional methods of developing programs using the standard languages. The unit specifically took you through MathCad software which is an Interpretive application capable of solving a wide range of mathematical and statistical problems. As seen in the unit, MathCad has its operators very similar to those you have already encountered in other programming languages.
The next unit will also introduce you to another specialized programming application before concluding this course?

6.0 TUTOR-MARKED ASSIGNMENTS

a) State five (5) arithmetic operators employed in MathCad programming that are the same with those of two

programming languages you have studied. State the languages

b) Mention specifically an operator used in Pascal employed by MathCad.

2. Give the MathCad coding that will produce the following symbols:

- √(Square Root)
- Σ (Summation)
- # (Not Equal to)
- ∫ (Integral)
- State three (3) built- in functions available in MathCad.
- What is the extension for files created by MathCad.

7.0 REFERENCES/FURTHER READINGS

MathSoft, Inc., MathCad Plus 6.0, 1995.

MathSoft, Inc. MathCad 2000 Professional, 1999

UNIT 5 USING MATLAB PROGRAMMING TOOLS

CONTENTS

1.0 INTRODUCTION

In this unit, which is the concluding unit for this course, you are going to be introduced to one of the most powerful mathematical applications available for Mathematics and Engineering problems today. MATLAB is a very high performance language for technical computing. It consists of essential features that will make you to appreciate the power of computing.

This unit will therefore introduce you to the general knowledge of the language application and some of its programming tools. Your study objectives for this unit are as presented below.

2.0 OBJECTIVES

By the end of this unit, you should be able to:

- State the typical uses of the MATLAB tools.
- Identify some of MATLAB commands and functions.
- Use some of the programming tools in MATLAB to solve problems.

3.0 MAIN CONTENT

3.1 Introduction to MATLAB

MATLAB simply stands for MATrix LABoratory and it was originally developed to provide access to matrix application developed by LINPAC and EISPACK projects.

Matlab is an interactive package whose basic data element is an array that does not require dimensioning. This special feature has made the

software very unique for solving many technical computational problems, especially those with matrix and vector formulations.

In brief, you can use MATLAB for the following:

- Mathematical computations
- Modeling and Simulation
- Algorithm Development
- Data Analysis and Visualization
- Scientific and Engineering Graphics
- Application Development
- etc.

Specifically, MATLAB is characterized by what are called TOOLBOXES. A Toolbox in MATLAB is a family of application-specific solutions and they serve as comprehensive collections of MATLAB functions which are called M-Files (i.e. with extension .M). Available toolboxes in MATLAB are as follows:

- Simulation
- Optimization
- Neural Networks
- Signal Processing
- Fuzzy Logic
- Control Systems
- Wavelets
- and others.

MATLAB Language

The MATLAB language has the following features:

- Control flow statements
- Functions
- Data structures
- Input/Output facilities
- Object-Oriented Programming.

The language allows what have been classified into two as follows:

- Programming in the small
- Programming in the large.
- "Programming in the small" simply means the creation of quick and easily disposable programs while "Programming in the large" has to do with the creation of complete, large and complex application programs.

- See below now the startup screen for MATLAB programming environment:

Figure 2

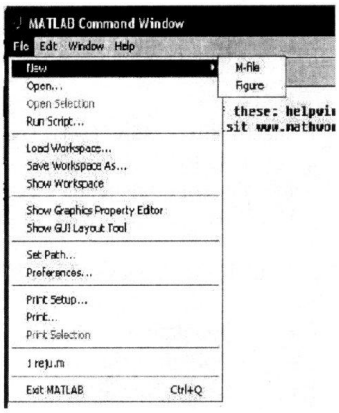
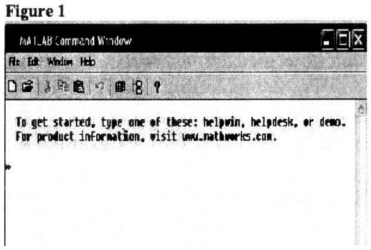


Figure 1



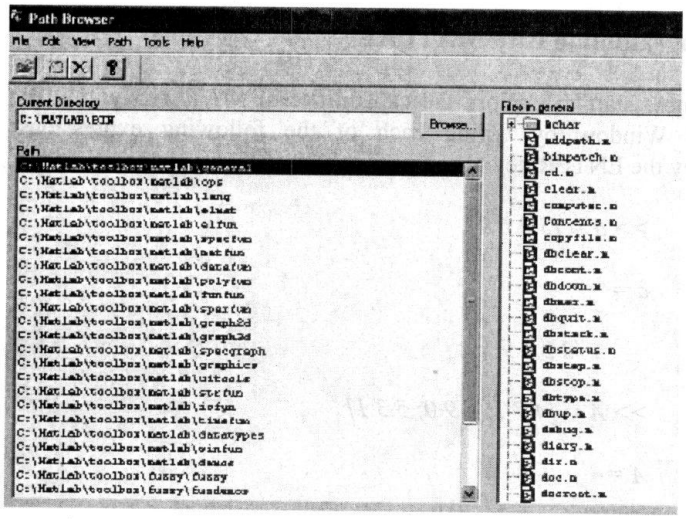
Looking at the two figures above, the programming environment looks so simple, hiding away the underlying programming capabilities.

In the first figure, there are two essential buttons apart from the standard buttons on the toolbar. They are for the following:

- Workspace Browser
- Path Browser.

For example, by clicking the Path Browser button, you have the following window when maximized.

Figure 3

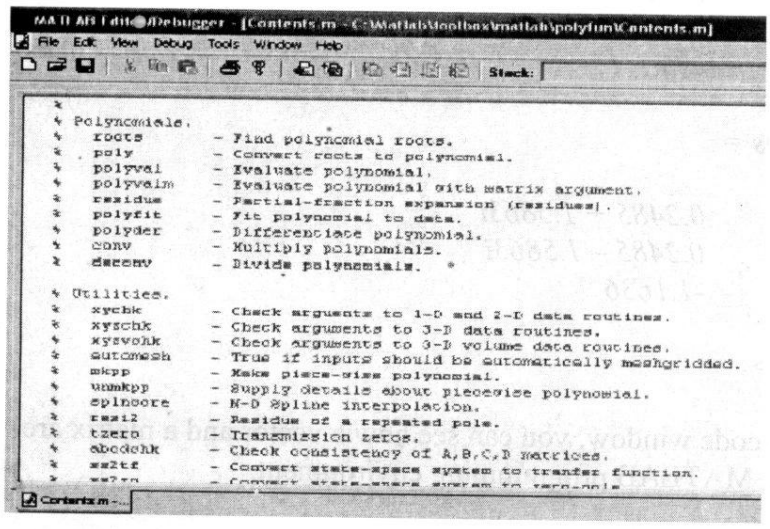


The Path Browser allows you to have access to some of the MATLAB (M) Files. For example, by double-clicking the file ***"contents.m"*** in the path

C:/Matlab \toolbox \matlab\polyfini"

you have the file opened as shown below in the MATLAB Editor.

Figure 4



The file shows some of the function names used by MATLAB to handle interpolation and polynomial functions.

You will now see below some specific examples of using the MATLAB programming features.

3.2 Programming with MATLAB

You will now start to explore the capabilities of MATLAB within its Command Window by typing each of the following expressions followed by the ENTER key to see your results:

```
>> a = [3 2 6 9]

 =
      3269
>> A = [4 71; 5 9 0; 53 1]
A=
                  471
                  590
                  531
>>B =A'
B =
```

```
                    455
                    793
                    101
 >> roots (a)
ans =
        0.2485 + 1.5863i
        0.2485 - 1.5863i
        -1.1636
 >>
```

From the above code window, you can see how a vector and a matrix are typed within the MATLAB programming environment.

Vector a is typed as seen above with a space between each element. When you hit the ENTER key, MATLAB returns

```
a=
        3269
```

However, a matrix is typed with semicolon to separate the rows as seen above. When you hit the ENTER key, MATLAB now arranges the matrix rows and columns as you see above.

The transpose of a matrix is defined by the single quote (B = A'). The moment you hit the ENTER key, the transpose is immediately displayed by MATLAB.

In the above working session, you have seen one of the functions employed by MATLAB. The roots of a polynomial function are obtained by using the built- in MATLAB function "roots".

Remember that you have earlier typed a vector:

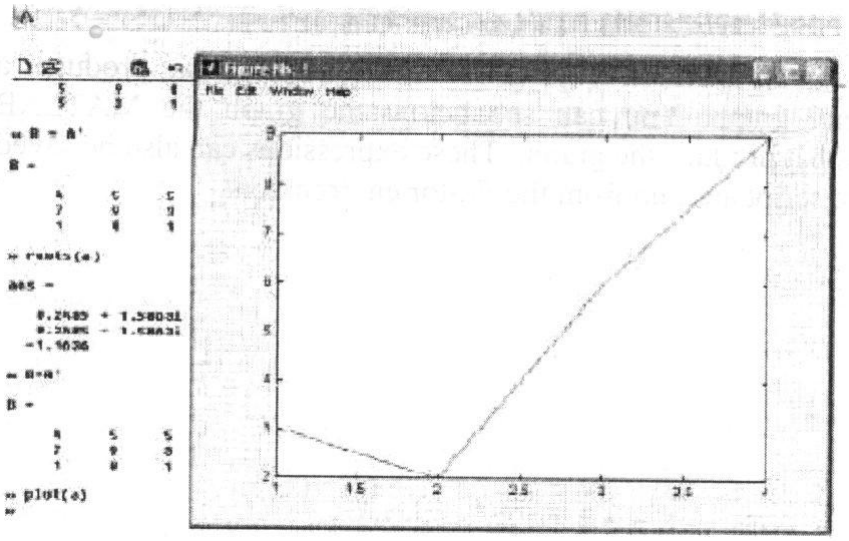
a = [3 2 6 9]

MATLAB takes the above as the coefficients of a polynomial when used with the "roots" function. That means that the polynomial associated with the above vector is as follows:

$$A(x) = 3x3 + 2x2 + 6x + 9$$

Now, look at the following figure:

Figure 5



You have used another MATLAB built- in function: **plot** (a) and the result is the graph seen above.

From the roots of vector a, you can see that MATLAB also handles complex computations.

MATLAB has a way of keeping track in memory of all that you have typed at the command prompt and the answers to your computational processing. You can recall all these from the memory by typing the command;

"whos"

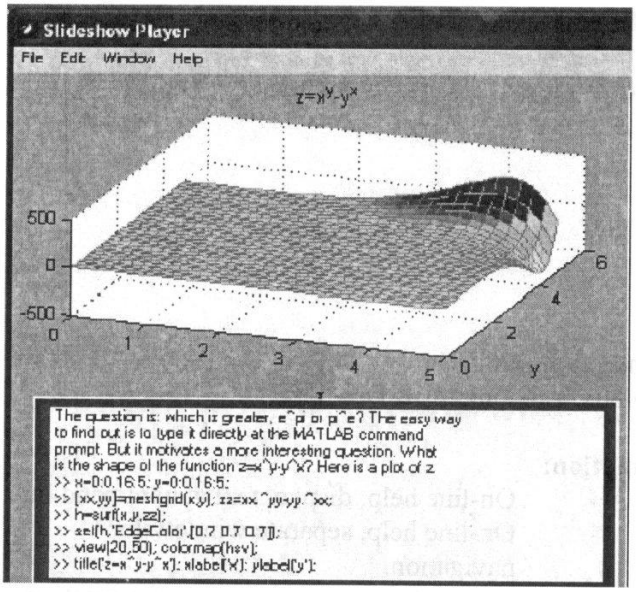
See this below:

Figure 6

```
» B=A'

B =

     4     5     5
     7     9     3
     1     0     1

» plot(a)
» WhOS
  Name        Size              Bytes  Class

  A           3x3                  72  double array
  B           3x3                  72  double array
  a           1x4                  32  double array
  ans         3x1                  48  double array (complex)

Grand total is 25 elements using 224 bytes

» |
```

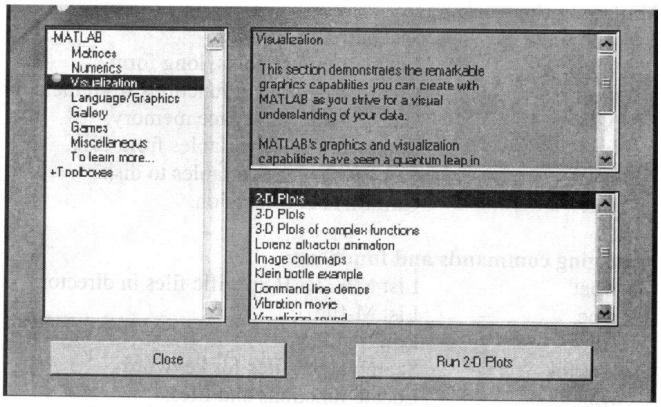
Now, below is an example of MATLAB programming that produces a 3-dimensional graph. You can see below the graph the MATLAB expressions that produce the graph. These expressions can also be saved as an M-File script and run from the Editor environment.

Figure 7



MATLAB has a number of DEMOS that will greatly help you to study various aspects of MATLAB programming capabilities. You can access them from the HELP menu of the command window. Below is a window to show you some of the demos available:

Figure 8



Now, before you round up this unit, see below the list of some commands used by MATLAB. The list is obtained by accessing some of the M-Files in the toolboxes.

Before the list, answer the following question.

SELF ASSESSMENT EXERCISE 1

What command do you use to clear your command window?

Answer

The command is "dc".
Now, see the list:

```
% General purpose commands:
% MATLAB Toolbox Version 5.2 18-Dec-1997
%
% General information:
%     help        -    On-line help, display text at command line.
%     helpwin     -    On-line help, separate window for
                       navigation.
%     helpdesk    -    Comprehensive hypertext documentation
                       and troubleshooting.
%     demo        -    Run demonstrations.
%     ver         -    MATLAB, SIMULINK, and toolbox version
                       information.
%     whatsnew    -    What's new in MATLAB 5.1
%     Readme      -    Display Readme files.
%
% Managing the workspace:
%     who         -    List current variables.
%     whos        -    List current variables, long form.
%     clear       -    Clear variables and functions from memory
%     pack        -    Consolidate workspace memory.
%     load        -    Load workspace variables from disk.
%     save        -    Save workspace variables to disk.
%     quit        -    Quit MATLAB session.
%
% Managing commands and functions:
%     what        -    List MATLAB-specific files in directory
%     type        -    List M-file
%     edit        -    Edit M-file
%     lookfor     -    Search all M-files for keyword
%     which       -    Locate functions and files.
%     pcode       -    Create pre-pared pseudo-code file (p-file).
%     inmem       -    List functions in memory
%     mex         -    Compile MEX-function.
%
```

% **Managing the search path:**

%	path	-	Get/set search path.
%	addpath	-	Add directory to search path.
%	rmpath	-	Remove directory from search path.
%	editpath	-	Modify search path.

%

% **Controlling the command window:**

%	echo	-	Echo commands in M-files
%	more	-	Control paged output in command window.
%	diary	-	Save text of MATLAB session
%	format	-	Set output format.

%

% **Operating system commands:**

%	cd	-	Change current working directory
%	copyfile	-	Copy a file
%	pwd	-	Show (print) current working directory
%	dir	-	List directory
%	delete	-	Delete file
%	getenv	-	Get environment variable
%	mkdir	-	Make directory
%	!	-	Execute operating system command (see PUNCT).
%	dos	-	Execute DOS command and return result.
%	unix	-	Execute UNIX command and return result
%	vms	-	Execute VMS DCL command and return result
%	web	-	Open Web browser on site or files.
%	computer	-	Computer type.

%

% **Debugging M-files:**

%	debug	-	List debugging commands
%	dbstop	-	Set breakpoint
%	dbclear	-	Remove breakpoint
%	dbcont	-	Continue execution
%	dbdown	-	Change local workspace context
%	dbstack	-	Display function call stack
%	dbstatus	-	List all breakpoints
%	dbstep	-	Execute one or more lines
%	dbtype	-	List M-file with line numbers
%	dbup	-	Change local workspace context
%	dbquit	-	Quit debug mode
%	dbmex	-	Debug MEX-files (UNIX only).

%

% **Profilling M -files**:

%	profile	-	Profile M-file execution time.

%

% See also PUNCT.

% **Obsolete functions :**

%	mexdebug	-	Debug MEX-files.

%

% **Others :**

%	binpatch	-	Patch binary file
%	doc	-	A utility for load HTML documentation into a web browser
%	docroot	-	A utility to determine MATLAB help root directory
%	exit	-	Exit from MATLAB
%	helpinfo	-	Information about help options
%	info	-	Information about MATLAB and the MathWorks
%	isstudent	-	True for the student edition of MATLAB
%	isunix	-	True for the UNIX version of MATLAB
%	isvms	-	True for the VMS version of MATLAB
%	isppc	-	True for Macintosh Power PC.
%	isieee	-	True for computers with IEEE arithmetic
%	ls	-	List directory.
%	matlabpath	-	Search path.
%	memory	-	Help for memory limitations
%	notebook	-	Open an m-book in Microsoft word (Windows only).
%	nnload	-	Netscape Navigator load
%	openvar	-	Open a workspace variable for graphical editing.
%	prepender	-	Utility function.
%	profsumm	-	Summarize profile information.
%	subscribe	-	Subscribe to the MathWorks Newsletter.

%

% **GUI Utilities**:

%	editarray	-	Edit an array graphically (Windows only).
%	maeasgn	-	Assign the result of an expression into an array subrange.
%	maesize	-	Print size of a 2-D array.
%	maeresize	-	Change the size of a matrix to be [m n].
%	maedispsubarray	-	Print specified subarray
%	mauifindexe	-	Return the absolute pathname to a MAUI executable.
%	mauifunc	-	Produce short description of a variable
%	mdbstatus	-	DBSTATUS for the Debugger/Editor

```
%     miedit      -   Edit M-file
%     miolereg    -   Register MATLAB as current OLE COM
                      object.
%     miport      -   Get the port which MATLAB is listening on.
%     genpath     -   Generate reasonable path based on toolbox.
%     path2rc     -   Save the current MATLAB path in the
                      pathdef.m file.
%     pathtool    -   Path Browser for Macintosh and PC.
%     regedit     -   Run the registry editor (UNIX only).
%     workspace   -   Workspace Browser for Macintosh and PC.
% Interpolation and polynomials.
%
% Data interpolation:
%     interpl     -   1-D interpolation (table lookup).
%     interplq    -   Quick 1-D linear interpolation
%     interpft    -   1-D interpolation using FFT method.
%     interp2     -   2-D interpolation (table lookup).
%     interp3     -   3-D interpolation (table lookup).
%     intern      -   N-D interpolation (table lookup).
%     griddata    -   Data gridding and surface fitting.
%
% Spline interpolation:
%     spline      -   Cubic spline interpolation
%     ppval       -   Evaluate piecewise polynomial.
%
% Geometric analysis:
%     delaunay    -   Delaunay triangulation.
%     dsearch     -   Search Delaunay triangulation for nearest
                      point.
%     tsearch     -   Closest triangle search.
%     convhull    -   Convex hull
%     voronoi     -   Voronoi diagram
%     inpolygon   -   True for points inside polygonal region
%     rectint     -   Rectangle intersection area.
%     polyarea    -   Area of polygon.
%
% Polynomials:
%     roots       -   Find polynomial roots.
%     poly        -   Convert roots to polynomial
%     polyval     -   Evaluate polynomial
%     polyvalm    -   Evaluate polynomial with matrix argument.
%     residue     -   Partial-fraction expansion (residues).
%     polyfit     -   Fit polynomial to data
%     polyder     -   Differentiate polynomials
%     conv        -   Multiply polynomials
%     deconv      -   Divide polynomials.
```

```
% Utilities:
%     xychk       -   Check arguments to 1-D and 2-D data
                      routines.
%     xyzchk      -   Check arguments to 3-D data routines.
%     xyzvchk     -   Check arguments to 3-D volume data
                      routines.
%     automesh    -   True if inputs should be automatically
                      meshgridded.
%     mkpp        -   Make piece-wise polynomial
%     unmkpp      -   Supply details about piecewise polynomial.
%     splncore    -   N-D Spline interpolation
%     resi2       -   Residue of a repeated pole.
%     tzero       -   Transmission zeros
%     abcdchk     -   Check consistency of A,B,C,D matrices.
%     ss2tf       -   Convert state-space system to transfer
                      function
%     ss2zp       -   Convert state-space system to zero-pole.
%     tf2ss       -   Convert transfer function to state-space
%     tf2zp       -   Convert transfer function to zero-pole.
%     tfchk       -   Check for proper transfer function.
%     zp2ss       -   Convert zero-pole system to state -space.
%     zp2tf       -   Convert zero-pole system to transfer
                      function.
%     mpoles      -   Identify repeated poles and their
                      multiplicities.
%
% Obsolete functions :
%     icubic      -   1-D cubic interpolation
%     interp4     -   2-D bilinear data interpolation.
%     interp5     -   2-D bicubic data interpolation
%     interp6     -   2-D nearest neighbor interpolation
%     table1      -   1-D table lookup
%     table2      -   2-D table lookup.
```

You will now round up this unit. **4.0 CONCLUSION**

This unit has shown you some few tools for advanced programming available in MATLAB. You have seen it demonstrated how **MATLAB** can instantly give you the results of mathematical computations that would normally take some efforts to carry out if you were to employ the conventional programming methods of using other languages.

This unit also is the concluding unit for the whole course. Definitely, you have learnt so much in this unit and the whole course to get you started with programming generally.

5.0 SUMMARY

MATLAB is one of the powerful interactive programming languages that this unit has briefly explored with you. MATLAB stands for

MATrix LABoratory, meaning that it is an application specifically suited for problems you can formulate in vectorial and matrix forms. Some few examples were treated to give yo u a taste of its immense capabilities. It is advisable that you access the Demos available in the Help files of the application so as to learn more about the powerful language.

Moreover, this unit concludes your study in this course, and with the general overview of programming languages you have already, you are now equipped to get started with programming. Good luck.

6.0 TUTOR-MARKED ASSIGNMENTS

1. Identify, among others, five areas of applications MATLAB can be used.
2. Give two functions each available in MATLAB for solving problems related to the following:
   - Polynomials
   - Matrices.

   What is a toolbox? Name 3 toolboxes available in MATLAB.

7.0 REFERENCES/FURTHER READINGS

The MathWorks, Inc., MATLAB Version 5.2.0. 3084, 1998. hap ://www.mathworks. corn