NATIONAL OPEN UNIVERSITY OF NIGERIA

SCHOOL OF SCIENCE AND TECHNOLOGY

COURSE CODE: CIT392

COURSE TITLE: Computer Laboratory II

Course Code	CIT 392
Course Title	COMPUTER LABORATORY II
Course Developer/Writer	Dr. ONASHOGA, S. A. (Mrs.) DEPT. OF COMPUTER SCIENCE UNIVERSITY OF AGRICULTURE, ABEOKUTA, OGUN STATE NIGERIA.
Course Editor	Greg Onwodi NOUN
Program Leader	Prof. Kehinde Obidairo
Course Coordinator	Greg Onwodi NOUN

NATIONAL OPEN UNIVERSITY OF NIGERIA

Table of Contents

Introduction

With the advancement of technology, the knowledge about micro computers operating system, programming languages and their uses became inevitable and an integral part of system and software design. Programming language is concerned with the design of software and computer applications which are a major part of the computer system and they enable us to carry out certain task on the computer system. The operating system introduces us to the proper usage of the system software especially the common ones.
Programming languages can be learnt by just anybody but how well it is learnt depends on the interest of individual and their preferred choice of language. Recently, there are over 50 programming languages but some of the common ones are C, C++ and Java. Similarly, there are various operating systems but its use depends on the users' choice and also on the improving technology in the advanced computer system.

What you will learn in this Course

The course consists of units and a course guide. This course guide tells you briefly what the course is about. What course materials you will be using and how you can work your way with these materials. In addition, it advocates some guidelines for the amount of time you are likely to spend on each unit of the course in order to complete it successfully.

It gives you guidance in respect of your Tutor-Marked Assignment which will be made available in the assignment file. There will be regular tutorial classes that are related to the course. It is advisable for you to attend these tutorial sessions.

Course Aims

This course provides different introductions to laboratory exercises, Module 1 is intended to provide the fundamental concepts in using programming languages, and Module 2 is intended to provide the fundamental concepts in using operating systems. Module 3 aims to cover database design using SQL and Module 4 aims to provide the fundamental concepts in using COBOL and ADA.

Course Objectives

To achieve the aims set out, the course has a set of objectives. Each unit has a specific objective which are included at the beginning of the unit. You should read these objectives before you study the unit. You may wish to refer to them during your study to check on your progress. You should always look at the unit objectives after completion of each unit. By doing so, you would have followed the instruction in the unit.

Below are the comprehensive objectives of the course as a whole. By meeting these objectives, you should have achieved the aims above.

Working through this Course

To complete this course you are required to read each study unit, read the textbooks and read other materials which may be provided by the National Open University of Nigeria.

Each unit contains an assignment which you would be required to submit for assessment purpose. At the end of the course, there is a final examination. The course should take about 17 weeks to complete. Below you will find listed all the components of the course, what you have to do and how you should allocate your time to each unit in order to complete the course on time and successfully.

This course entails that you spend a lot of time to read. I would advice that you avail yourself the opportunity of attending the tutorial sessions where you will have the opportunity of comparing your knowledge with that other people.

The Course Materials

The main components of the course are:

- The Course Guide
- Study Units
- Assignments
- Further Readings

Study Unit

The study units in this course are as follows:

Unit 4 Programming in Java

Module 2 Laboratory exercises using microcomputer operating systems

Unit 1 Ms-XP

Unit 2 Windows 7

Unit 3 Linux

Module 3 Laboratory exercises using SQL

Unit 1 Concepts of SQL

Unit 2 Data Query Language

Unit 3 Data Manipulation Language

Unit 4 Creating Database Objects

Unit 5 Aggregate Function

Module 4 Laboratory exercises using ADA and COBOL

Unit 1 ADA Programming Language

Unit 2 COBOL Programming Language

Assessment

There are two assessment of this course. First is made up of the tutor-marked assignments and the second is the written examination/end of course examination.

You are advised to do the exercises. In tackling the assignments, you are expected to apply information, knowledge and techniques you gathered during the course. The assignments must be submitted to your facilitator for formal assessment in accordance with the deadlines stated in the assignment file. The work you submitted to your tutor for assessment will count for 30% of your total course work. At the end of the course you will need to sit for a final or end of course examination of about three hour duration. This examination will count for 70% of your total course mark.

Tutor-Marked Assignment (TMA)

The TMA is a continuous assessment component of your course. It accounts for 30% of the total score. You will be given some TMAs to answer. These must be answered before you are allowed to sit for the end of course examination. The TMAs would be given to you by your facilitator and returned after you have done the assignment. Assignment question for the units in this course are contained in the assignment file. You will be able to complete your assignment from the information and material contained in your reading, references and study units. However, it is desirable in all degree level of education to demonstrate that you have read and researched more into your references which will give you a wider view point and may provide you with a deeper understanding of the subject.

Make sure that each assignment reaches your facilitator on or before the deadline given in the assignment file. If for any reason you cannot complete your work on time, contact your facilitator before the assignment is due to discuss the possibility of an extension, Extension will not be granted after the due date unless there are exceptional circumstances.

Final Examination and Grading

The duration for the final exam is about 3 hours and it has a value of 70% of the total course work. The examination will consist of practical questions, which will reflect the type of self-testing, practice exercise and tutor-marked assignment problems you have previously encountered. All areas of the course will be assessed.

Make use of the time between finishing the last unit and sitting for the examination to revise the whole course. You might find it useful to review your self-test, TMAs and comment on them before the examination. The end of course examination covers information from all parts of the course.

Course Marking Scheme

Assignment	Marks
Assignment 1-4	Four Assignments, best three marks of the four count at 10% each -30% of course marks.
End of Course Examination	70% of overall course marks.
Total	100% of course materials

Facilitators/Tutors and Tutorials

There are 16 hours of tutorials provided in support of this course. You will be notified of the dates, times and locations of these tutorials as well as the name and phone number of your facilitator so soon as you are allocated a tutorial group.

Your facilitator will mark and comment on your assignment, keep a close watch on your progress and any difficulties you might face and provide assistance to you during the course. You are expected to mail your Tutor Marked Assignment to your facilitator before the schedule date (at least two working days are required). They will be marked by your tutor and returned to you as soon as possible.

Do not delay to contact your facilitator by telephone or e-mail if you need assistance.

The following might be circumstances in which you would find assistance necessary, hence you would have to contact your facilitator if:

- You do not understand any part of the study or the assigned readings.
- You have difficulty with the self-tests.
- You have a question or problem with an assignment or with the grading of an assignment.

You should endeavor to attend the tutorials. This is the only chance to have face-to-face contact with your course facilitator and to ask questions which are answered instantly. You can raise any problem encountered in the course of your study.

Summary

This course intends to provide a background to the knowledge of programming language and the use of some operating systems. Upon completing the course, you will be equipped with the basic knowledge of programming languages and operating systems that are common to all software designers and programmers. In addition, you will be able to answer the following questions:

- ✓ Of what importance is the operating system to computer systems?
- ✓ What applications can we develop/build with programming languages?
- ✓ What is an Operating system?
- ✓ How can we build a computer application that is useful to us and our environment?
- ✓ How can we design our own personal database?

The questions to be answered in this course are not limited to the above-listed. To gain the most from this course, you should check out the texts from the list of references for further studies.

As you go through this class, I wish you success in this course and I hope you will find it both interesting and useful.

Best of Luck!

MODULE 1 – Laboratory Exercises on Programming Languages

UNIT 1: Introduction to Programming

Contents **Pages**

1.0 INTRODUCTION

In any language there are some fundamentals you need to know before you can write even the most elementary programs. These units introduce three such fundamentals: basic program construction, variables, and input/output (I/O). It also touches on a variety of other language features, including comments, arithmetic operators, the increment operator, data conversion, and library functions.

2.0 OBJECTIVES

- Understand what a program is
- How computer can be used to solve problems.
- Distinguish between Procedural, Structured and Object-Oriented Programming

3.0 PROGRAMMING

A digital computer is a useful tool for solving a great variety of **problems**. A solution to a problem is called an **algorithm**; it describes the sequence of steps to be performed for the problem to be solved. A simple example of a problem and an algorithm for it would be:

Problem: Sort a list of names in ascending lexicographic order.
Algorithm: Call the given list, *list1*; create an empty list, *list2*, to hold the sorted list.
Repeatedly find the 'smallest' name in *list1*, remove it from *list1*, and make it the next entry of *list2*, until *list1* is empty.

An algorithm is expressed in abstract terms. To be intelligible to a computer, it needs to be expressed in a language understood by it. The only language really understood by a computer is its own **machine language**. Programs expressed in the machine language are said to be **executable**. A program written in any other language needs to be first translated to the machine language before it can be executed.
A machine language is far too cryptic to be suitable for the direct use of programmers. A further abstraction of this language is the **assembly language** which provides mnemonic names for the instructions and a more intelligible notation for the data. An assembly language program is translated to machine language by a translator called an **assembler**.
Even assembly languages are difficult to work with. High-level languages such as C++ provide a much more convenient notation for implementing algorithms.
They liberate programmers from having to think in very low-level terms, and help them to focus on the algorithm instead. A program written in a high-level language is translated to machine language by a translator called a **compiler**.

Programs

The word *program* is used in two ways: to describe individual instructions (or source code) created by the programmer, and to describe an entire piece of executable software. This distinction can cause enormous confusion, so we will try to distinguish between the source code on one hand, and the executable on the other.
A program can be defined as either a set of written instructions created by a programmer or an executable piece of software.

Source code can be turned into an executable program in two ways: Interpreters translate the source code into computer instructions, and the computer acts on those instructions immediately. Alternatively, compilers translate source code into a program, which you can run at a later time. Although interpreters are easier to work with, most serious programming is done with compilers because compiled code runs much faster. C++ is a compiled language.

3.1 SOLVING PROBLEMS

The problems programmers are asked to solve have been changing. Twenty years ago, programs were created to manage large amounts of raw data. The people writing the code and the people using the program were all computer professionals. Today, computers are in use by far more people, and most know very little about how computers and programs work. Computers are tools used by people who are more interested in solving their business problems than in struggling with the computer.

Ironically, to become easier to use for this new audience, programs have become far more sophisticated. Gone are the days when users typed cryptic commands at esoteric prompts, only to see a stream of raw data. Today's programs use sophisticated "user-friendly interfaces" involving multiple windows, menus, dialog boxes, and the myriad of metaphors with which we have become familiar. The programs written to support this new approach are far more complex than those written only ten years ago.

With the development of the Web, computers have entered a new era of market penetration; more people are using computers than ever before and their expectations are very high. In the last few years, programs have become larger and more complex, and the need for object-oriented programming techniques to manage this complexity has become manifest.

As programming requirements have changed, both languages and the techniques used for writing programs have evolved.

3.2 PROCEDURAL, STRUCTURED, AND OBJECT-ORIENTED PROGRAMMING

Until recently, programs were thought of as a series of procedures that acted upon data. A *procedure*, or function, is a set of specific instructions executed one after the other. The data was quite separate from the procedures, and the trick in programming was to keep track of which functions called which other functions, and what data was changed. To make sense of this potentially confusing situation, structured programming was created.

The principal idea behind structured programming is as simple as the idea of divide and conquer. A computer program can be thought of as consisting of a set of tasks. Any task that is too complex to be described simply would be broken down into a set of smaller component tasks, until the tasks were sufficiently small and self-contained that they were easily understood.

As an example, computing the average salary of every employee of a company is a rather complex task. You can, however, break it down into the following subtasks:

1. Find out what each person earns.
2. Count how many people you have.
3. Total all the salaries.
4. Divide the total by the number of people you have.

Totaling the salaries can be broken down into the following steps:

1. Get each employee's record.

2. Access the salary.
3. Add the salary to the running total.
4. Get the next employee's record.

In turn, obtaining each employee's record can be broken down into the following:

1. Open the file of employees.
2. Go to the correct record.
3. Read the data from disk.

Structured programming remains an enormously successful approach for dealing with complex problems. By the late 1980s, however, some of the deficiencies of structured programming had become all too clear.

First, a natural desire is to think of data (employee records, for example) and what you can do with data (sort, edit, and so on) as a single idea. Procedural programming worked against this, separating data structures from functions that manipulated that data.

Second, programmers found themselves constantly reinventing new solutions to old problems. This is often called "reinventing the wheel," which is the opposite of reusability. The idea behind reusability is to build components that have known properties, and then to be able to plug them into your program as you need them. This is modeled after the hardware world—when an engineer needs a new transistor, she usually does not invent a new one; she goes to the big bin of transistors and finds one that works the way she needs it to, or perhaps modifies it. No similar option existed for a software engineer.

Object-oriented programming attempts to respond to these needs, providing techniques for managing enormous complexity, achieving reuse of software components, and coupling data with the tasks that manipulate that data.

The essence of object-oriented programming is to treat data and the procedures that act upon the data as a single object—a self-contained entity with an identity and certain characteristics of its own.

4.0 CONCLUSION

A digital computer is a useful tool for solving a great variety of **problems**. A computer can do anything you tell it, but it does *exactly* what it's told —nothing more and nothing less.

Computers understand a language variously known as *computer language* or *machine language.* It's possible but extremely difficult for humans to speak machine language. Therefore, a sort of way to speak, using intermediate languages such as C++. Humans can speak C++, and C++ is converted into machine language for the computer to understand.

5.0 SUMMARY

This unit introduces you to programming, explaining the fundamentals involved in writing a program.

6.0 TUTOR MARKED ASSIGNMENT

1. What do you understand by programming?
2. What is a program?
3. What is an Algorithm?
4. Explain what you understand by procedural, structural and Object-Oriented programming?

7.0 FURTHER READING

1. http://www.cplusplus.com/doc/ - tutorial
2. http://www.wcug.wwu.edu/~anton/cpp/
3. http://www.linuxdoc.org/HOWTO/C++Programming-HOWTO.html
4. http://www.frontsource.com.pk/html/vc.html
5. http://msdn.microsoft.com/library/default.asp?URL=/library/devprods/vs6/visualc/vcedit/vcovrvisualcdocumentationmap.htm

MODULE 1 – Laboratory Exercises on Programming Languages

UNIT 2: Programming In C

Contents Page

1.0 Introduction

The **C** programming language was originally developed by Dennis Ritchie of Bell Laboratories, and was designed to run on a PDP-11 with a UNIX operating system. Programming in C is a tremendous asset in those areas where you may want to use Assembly Language, but would rather keep it a simple to write and easy to maintain program. It has been said that a program written in C will pay a premium of a 50% to 100% increase in runtime, because no language is as compact or fast as Assembly Language. However, the time saved in coding can be tremendous, making it the most desirable language for many programming chores. Since C was designed essentially by one person, and not by a committee, it is a very usable language but not too closely Note that you need to get yourself a copy of the development kit (possibly the Turbo C software).Install and launch, You should see a page like this:

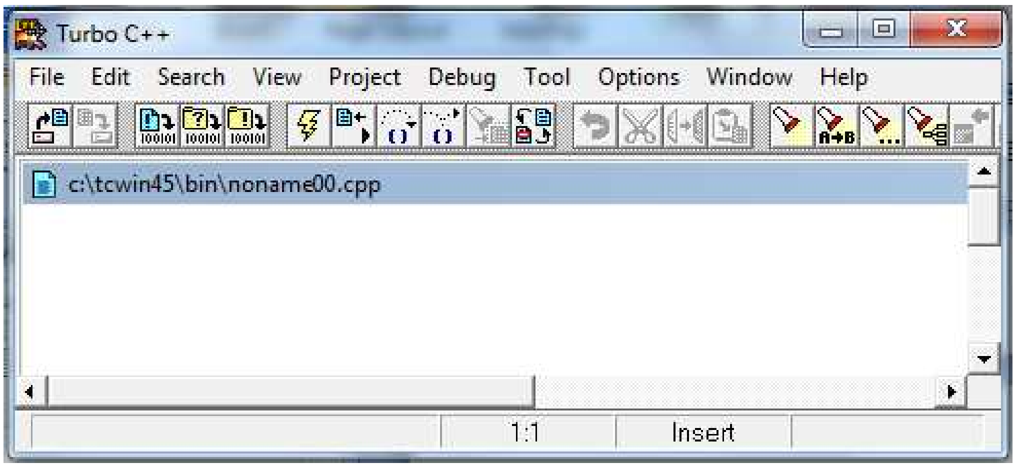


Figure 1.2(a): Turbo C++ Development window.

Note: you would have to save your files with the extension ".c" .

2.0 Objective

On completing this unit, you would be able to:

- Create and write a source code in C
- Understand Control Structure in C
- Understand Functions in C
- Understand Pointer and Arrays in C

3.0 Creating a C source file

To start a C code, we can use a C compiler like Turbo C or Borland C that can also run C. Start with the following steps:

1. Open the C compiler, it looks like the figure below

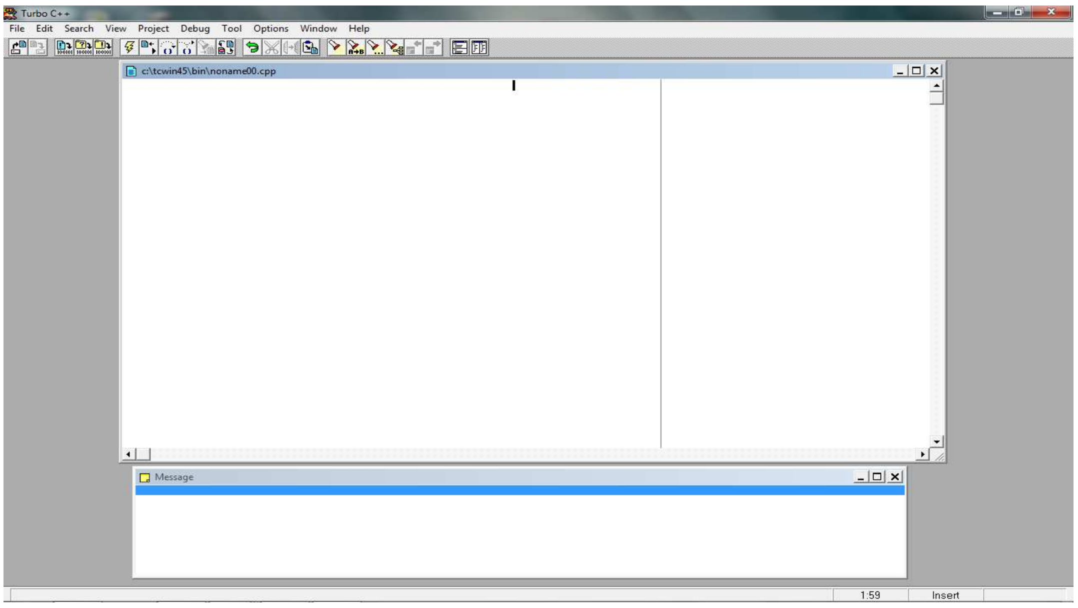

Turbo C++
File Edit Search View Project Debug Tool Options Window Help
c:\tcwin45\bin\noname00.cpp
Message
1:59
Insert

Figure 1.2(b): The Compiler environment

2. Click on File menu from and the select the Save as option to save the file with a file name and an extension. The file name extension of C is “.c”. For example man.c is the file name that indicates the type of file through its file extension as a C file.

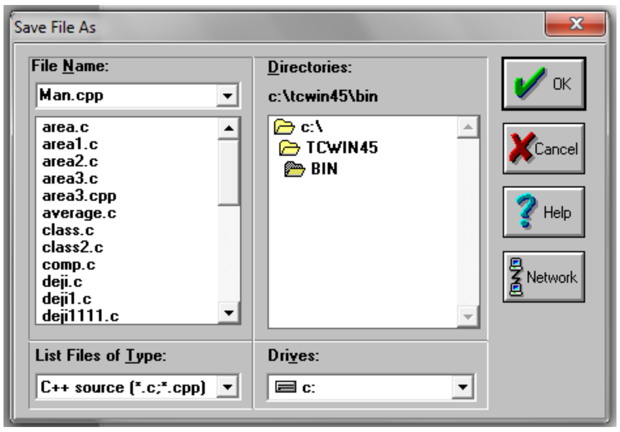


Figure1.2(c): Menu for saving source file

3. The file is then saved. Writing and editing of codes is done in the blank/white space environment of the compiler.

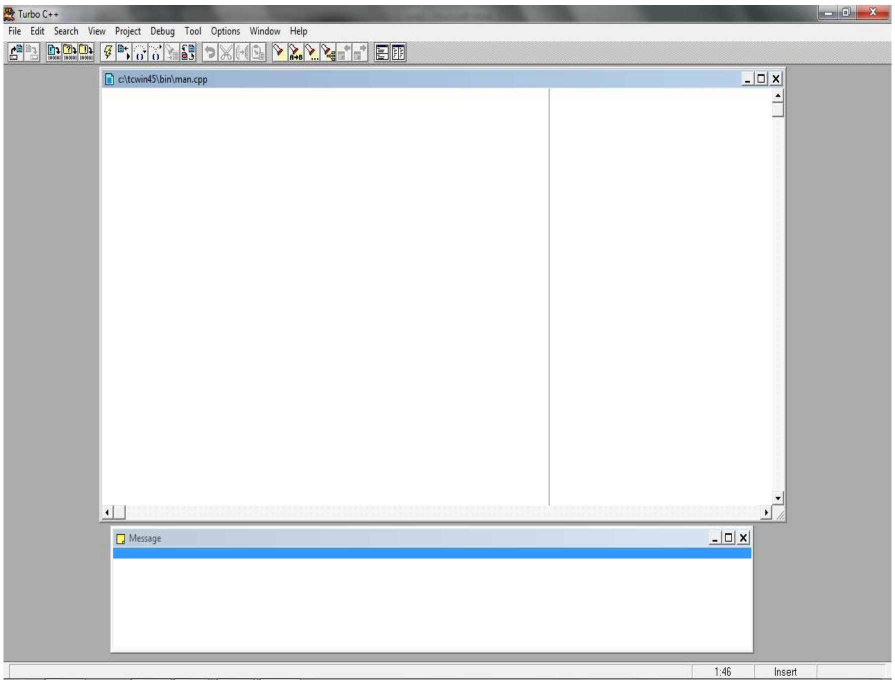

Turbo C++
File Edit Search View Project Debug Tool Options Window Help
c:\tcwin45\bin\man.cpp
Message
1:46
Insert

Figure1.2(d): The saved C file with a blank working space ready for use.

3.1 Elements of C program

The elements of C are like the main parts of the C codes. They can be divided into three parts;

A**. Program Comments**

Program comments are explanatory statements that you can include in the C code that you write to give an over view of what the program is all about. All programming languages allow for some form of comments. C compiler evaluates these lines of codes not as the executable codes. As a result, when we run our executable file, these lines of code are effectively ignored. It could be in this form;

```
/* This program displays "Hello!"*/

// This program displays "Hello!”
```

B. **Include Statements**

This is one the most important line of code in C program, #include <stdio.h>.These are also known as *header files*. They are found in a special directory that is established when you install your compiler with each one providing the programs we write with a different piece of functionality that otherwise we would have to write ourselves. As to the particulars of stdio.h— the letters *io* in stdio.h refer to input-output, which is a computer term that describes how data is input into our programs and how we get information out of them. If we fail to include the stdio.h file in our program, we wouldn’t be able to display the words ‘Hello!’

C. **The main () Function**

The **main()** function is a section of code where the major instructions of a C program are placed. It contains the data that are declared as variables or constants, objects created from classes, namespace declaration, data type declaration and others. There are other functions that can be created in the program but the main() function must be created, without it there is no existence of the code.

```
        int main()
        {
         -    - -
        }
```

The above *int main* shows that the returned value of the **main ()** function will be an *integer* value.

This unit can be read simply as a text however it is intended to be interactive. That is, you should be compiling, modifying and using the programs that are presented herein.

```
#include <stdio.h>
int main ()
{
        Printf("Welcome to my world!");
        return 0;
}
```

NOTE: All lines of code or C statement must end with a semicolon (;) else the line of code will not be executed by the compiler. In order to output text to the monitor, it is put within the function parentheses and bounded by quotation marks. The end result is that whatever is included between the quotation marks will be displayed on the monitor when the program is
run. C uses a semi-colon as a statement terminator, so the semi-colon is required as a signal to the compiler that this line is complete.

D. **Identifiers and Keywords**

ldentzfiers are names that are given to various program elements, such as variables, functions and arrays. Identifiers consist of letters and digits, in any order, except that ***thefirst character must be a letter.*** Both upper- and lowercase letters are permitted, though common usage favors the use of lowercase letters for most types of identifiers. Upper- and lowercase letters are not interchangeable (i.e., an uppercase letter is ***not*** equivalent to the corresponding lowercase letter.) The underscore character (-) can also be included, and is considered to be a letter. There are certain reserved words, called ***keywords,*** that have standard, predefined meanings in C. These keywords can be used only for their intended purpose; they cannot be used as programmer-defined identifiers. The standard keywords are; auto, extern, sizeof, break, float, static, case, for, struct and so on**.**

3.2 Data: Variables and Constants

3.2.1 Constants

There are four basic types of constants in C. They are ***integer constants, floating-point constants, character constants*** and ***string constants***

The following rules apply to all numeric-type constants.

1. Commas and blank spaces cannot be included within the constant.
2. The constant can be preceded by a minus (-) sign if desired. (Actually the minus sign is an ***operator*** that changes the sign of a positive constant, though it can be thought of as a part of the constant itself.)
3. The value of a constant cannot exceed specified minimum and maximum bounds. For each type of constant, these bounds **will** vary from one C compiler to another.

Integer Constants

An ***integer constant*** is an integer-valued number. Thus it consists of a sequence of digits. Integer constants can be written in three different number systems: decimal (base l0), octal (base **8)** and hexadecimal (base 16).

Floating-Point Constants

A floating-point constant is a base- 10 number that contains either a decimal point or an exponent (or both). Several valid floating-point constants are shown below.

```
0. 1  0.2  827.602
50000.  0.000743  12.3 31  5.0066
2 E-8  0.006e-3  1.6667E+8 .12121212e12
```

The following are *not* valid floating-point constants for the reasons stated.

1	Either a decimal point or an exponent must be present.
1,000.0	Illegal character (,).
2E+10.2	The exponent must be an integer quantity (it cannot contain a decimal point).
3E 10	Illegal character (blank space) in the exponent.

Character Constants

A ***character constant*** is a single character, enclosed in apostrophes (i.e., single quotation marks). Several character constants are as follows; **'A' ' X ' ‘3’**

String Constants

A ***string constant*** consists of any number of consecutive characters (including none), enclosed in (double) quotation marks. Several string constants are as follows;

"green"   "Washington, **D.C.** 20005H "

"$19.95"   "THE CORRECT ANSWER **IS:'** '**2*** (**I+3)**/J "

"Line l\nLine 2\nLine **3**"

Note that the string constant "Line 1\nLine 2\nLine 3" extends over three lines, because of the newline characters that are embedded within the string. Thus, this string would be displayed as

Line 1
Line 2
Line 3.

3.2.2 Variables

A *variable* is an identifier that is used to represent some specified type of information within a designated portion of the program. In its simplest form, a variable is an identifier that is used to represent a single data item; i.e., a numerical quantity or a character constant. A given variable can be assigned different data items at various places within the program. Thus, the information represented by the variable can change during the execution of the program. However, the data type associated with the variable cannot change. A C program contains the following lines.

```
i n t a, b, c;
char d;
. . .
a = 3;
b = 5;
c = a + b ;
d = 'a'  ;
a = 4;
b = 2;
c = a - b ;
d = ' W ' ;
```

The first two lines are *type declarations*, which state that a, b and c are integer variables, and that d is a char-type variable. Thus a , b and c will each represent an integer-valued quantity, and d will represent a single character.

3.3 DATA TYPES

C supports several different types of data, each of which may be represented differently within the computer's memory. The basic data types are listed below. Typical memory requirements are also given. (The memory requirements for each data type will determine the permissible range of values for that data type. Note that the memory requirements for each data type may vary from one C compiler to another.)

Table 1.1 Data Types

Int	integer quantity	2 bytes or one word
Char	single character	1 byte
Float	floating-point number (i.e., a number containing a decimal point and or an exponent)	1 word **(4** bytes)
Double	double-precision floating-point number (i.e., more significant figures, and an exponent which may be larger in magnitude)	2 words (8 bytes)

The basic data types can be augmented by the use of the data type ***qualifiers*** short , long, signed and unsigned. For example, integer quantities can be defined as short int , long int or unsigned int.

3.4 Operators on Data: Arithmetic and Logical Operator

The data items that operators act upon are called ***operands.*** Some operators require two operands, while others act upon only one operand. Most operators allow the individual operands to be expressions. **A** few
operators permit only single variables as operands

3.4..1 Arithmetic Operations

There are five *arithmetic operators* in C. They are

Operator	*Purpose*
+	addition
-	subtraction
*	multiplication
/	division
%	remainder after integer division

The % operator is sometimes referred to as the ***modulus operator.*** Example **of** the arithmetic expression

i. (a - b) / (c * d).

ii. 2 * **((i** % **5)** * **(4** + (j - **3)** / **(k** + **2)))**

3.4.2 Logical Operators

There are four ***logical operators*** in C. They are;

Operator	Meaning
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	equal to
!=	not equal to
&&	and
||	or

Exercises

Our first class exercise on how to work with data in a C program;
1. write a program to print numbers using you own identifier.
2. Write a program to display your name on the monitor.
3. Modify the program to display your address and phone number on separate lines by adding two additional "printf" statements.

Sample;

```
main( )
{
int index;
index = 13;
printf("The value of the index is %d\n",index);
index = 27;
printf("The valve of the index = %d\n",index);
index = 10;
printf("The value of the index = %d\n",index);
}
```

3.5 Control Structures
The C programming language has several structures for looping and conditional branching. We will cover them all in this section and we will begin with the while loop.

3.5.1 The While Loop
The *while loop* continues to loop while some condition is true. When the condition becomes false, the looping is discontinued. It therefore does just what it says it does, the name of the loop being very descriptive.
An example of a while loop.

```
/* This is an example of a "while" loop */
main( )
{
int count;
count = 0;
while (count < 6) {
printf("The value of count is %d\n",count);
count = count + 1;
}
}
```

3.5.2 The Do-While Loop
A variation of the while loop is illustrated in the program below which you should load
and display.

```
/* This is an example of a do-while loop */
main( )
{
int i;
i = 0;
do {
printf("the value of i is now %d\n",i);
i = i + 1;
} while (i < 5);
}
```

This program is nearly identical to the last one except that the loop begins with the reserved word "do", followed by a compound statement in braces, then the reserved word "while", and finally an expression in parentheses. The statements in the braces are executed repeatedly as long as the expression in parentheses is true.

3.5.3 The For Loop

The "for" loop is really nothing new, it is simply a new way of describe the "while" loop. An example of a program with a "for" loop.

```
/* This is an example of a for loop */
main( )
{
int index;
for(index = 0;index < 6;index = index + 1)
printf("The value of the index is %d\n",index);
}
```

The "for" loop consists of the reserved word "for" followed by a rather large expression in parentheses. This expression is really composed of three fields separated by semi-colons. The first field contains the expression "index = 0" and is an initializing field. Any expressions in this field are executed prior to the first pass through the loop. There is essentially no limit as to what can go here, but good programming practice would require it to be kept simple.
The second field, in this case containing "index < 6", is the test which is done at the beginning of each loop through the program. It can be any expression which will evaluate to a true or false. The expression contained in the third field is executed each time the loop is executed but it is not executed until after those statements in the main body of the loop are executed.

3.5.4 The If Statement

An example of our first conditional branching statement; the "if".

```
/* This is an example of the if and if-else statements */
main()
{
int data;
for(data = 0;data < 10;data = data + 1) {
if (data == 2)
printf("Data is now equal to %d\n",data);
if (data < 5)
printf("Data is now %d, which is less than 5\n",data);
else
printf("Data is now %d, which is greater than 4\n",data);
} /* end of for loop */
}
```

Consider the first "if" statement. It starts with the keyword "if" followed by an expression in parentheses. If the expression is evaluated and found to be true, the single statement following the "if" is executed. If false, the following statement is skipped. Here too, the single statement can be replaced by a compound statement composed of several statements bounded by braces.

3.5.5 The If-Else

The second "if" is similar to the first, with the addition of a new reserved word, the "else", following the first printf statement. This simply says that, if the expression in the parentheses evaluates as true, the first expression is executed, otherwise the expression following the "else" is executed. Compile and run this program to see if it does what you expect.

3.5.6 The Break And Continue

An example of two new statements; “break” and “continue”

```
main( )
{
int xx;
for(xx = 5;xx < 15;xx = xx + 1){
if (xx == 8)
break;
printf("in the break loop, xx is now %d\n",xx);
}
for(xx = 5;xx < 15;xx = xx + 1){
if (xx == 8)
continue;
printf("In the continue loop, xx is the now %d\n",xx);
}
}
```

Notice that in the first "for" there is an if statement that calls a break if xx equals 8. The break will jump out of the loop you are in and begin executing the statements following the loop, effectively terminating the loop. This is a valuable statement when you need to jump out of a loop depending on the value of some results calculated in the loop. In this case, when xx reaches 8, the loop is terminated and the last value printed will be the previous value, namely 7.

The next "for" loop, contains a continue statement which does not cause termination of the loop but jumps out of the present iteration. When the value of xx reaches 8 in this case, the program will jump to the end of the loop and continue executing the loop, effectively eliminating the printf statement during the pass through the loop when xx is eight. Compile and run the program to see if it does what you expect.

3.5.7 Switch Statement

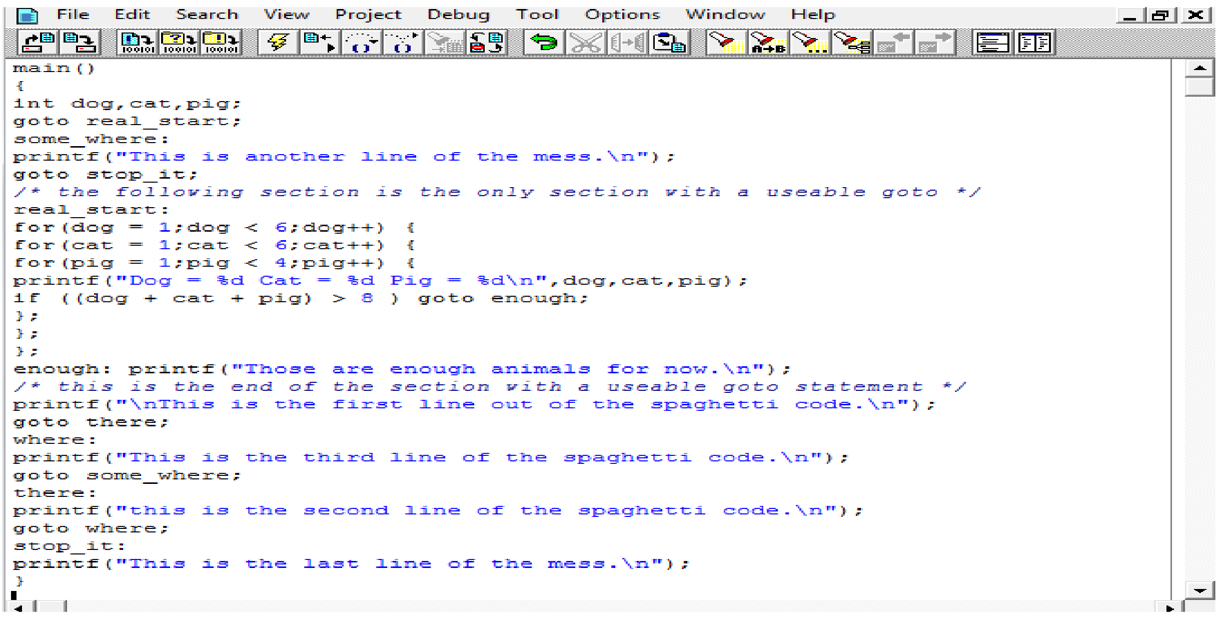
An example of the biggest construct yet in the C language; the switch.

```
main( )
{
int truck;
for (truck = 3;truck < 13;truck = truck + 1) {
switch (truck) {
case 3 : printf("The value is three\n");
break;
case 4 : printf("The value is four\n");
break;
case 5 :
```

```
case 6 :
case 7 :
case 8 : printf("The value is between 5 and 8\n");
break;
case 11 : printf("The value is eleven\n");
break;
default : printf("It is one of the undefined values\n");
break;
} /* end of switch */
} /* end of the loop */
}
```

The switch is not difficult, so don’t let it intimidate you. It begins with the keyword "switch" followed by a variable in parentheses which is the switching variable, in this case "truck". As many cases as desired are then enclosed within a pair of braces. The reserved word "case" is used to begin each case entered followed by the value of the variable, then a colon, and the statements to be executed. Compile and run to see if it does what you expect it to after this discussion.

3.5.8 Goto Statement



```
main()
{
int dog,cat,pig;
goto real_start;
some_where:
printf("This is another line of the mess.\n");
goto stop_it;
/* the following section is the only section with a useable goto */
real_start:
for(dog = 1;dog < 6;dog++) {
for(cat = 1;cat < 6;cat++) {
for(pig = 1;pig < 4;pig++) {
printf("Dog = %d Cat = %d Pig = %d\n",dog,cat,pig);
if ((dog + cat + pig) > 8 ) goto enough;
};
};
};
enough: printf("Those are enough animals for now.\n");
/* this is the end of the section with a useable goto statement */
printf("\nThis is the first line out of the spaghetti code.\n");
goto there;
where:
printf("This is the third line of the spaghetti code.\n");
goto some_where;
there:
printf("this is the second line of the spaghetti code.\n");
goto where;
stop_it:
printf("This is the last line of the mess.\n");
}
```

Figure 1.2(e): An example of a file with some "goto" statements in it.

To use a "goto" statement, you simply use the reserved word "goto", followed by the symbolic name to which you wish to jump. The name is then placed anywhere in the program followed by a colon.

Exercises

1. Write a program that writes your name on the monitor ten times. Write this program three

times, once with each looping method.
2. Write a program that counts from one to ten, prints the values on a separate line for each, and includes a message of your choice when the count is 3 and a different message when the count is 7.

3.6 LIBRARY FUNCTIONS

The C language is accompanied by a number of *library functions* that carry out various commonly used operations or calculations. These library functions are not a part of the language per se, though all implementations of the language include them. Some of these include;

Function	*Type*
abs (i)	int
exp(d)	double
fabs (d)	double
floor (d)	double
pow(d1,d2)	double
printf (...)	int
putchar(c)	int
sqrt(d)	double
scanf(...)	int
tan (d)	double
tolower (c)	int
toupper (c)	int

3.7 FUNCTIONS

A function is a self-contained program segment that carries out some specific, well-defined task. C program can be modularized through the intelligent use of functions. the use of a function avoids the need for redundant (repeated) programming of the same instructions. A function will process information that is passed to it from the calling portion of the program, and return a single value. A good example to describe functions is as follows;

/* convert a lowercase character to uppercase using a programmer-defined function */

```
#include <stdio.h>
char lower-to-upper(char c l ) /* function d e f i n i t i o n */
char c2;
c2 = ( cl >= 'a' && cl <= '2' ) 7 ( 'A' + c l - 'a' ) : c l ;
return(c2);
}
main( )
{
char lower, upper;
printf("P1ease enter a lowercase character: " ) ;
scanf ( *%c" &lower) ;
upper = lower-to-upper( lower) ;
printf(\nThe uppercase equivalent is %c\n\n", upper) ;
}
```

This program consists of two functions-the required main function, preceded by the programmer-defined function lower-to-upper. Compile and run to see what the program does. A function definition has two principal components: the ***first line*** (including the ***argument declarations),*** and the body of the function.

3.8 ARRAYS

An array consists of a set of objects (called its elements), all of which are of the same type and are arranged contiguously in memory. In general, only the array itself has a symbolic name, not its elements. Each element is identified by an index which denotes the position of the element in the array. The number of elements in an array is called its dimension. The general form of an array is *data-type array[expression]* ;

Several typical one-dimensional array definitions are shown below.-

```
int x[lOO];
char text [8 0] ;
float n[12];
```

We have two type of arrays; One Dimensional Array and Multidimensional Array. Their declarations are similar but that of Multidimensional is a little more complex.

data- type array[*expression I]* [*expression* 21 . . . [*expression n]* ;

Several typical multidimensional array definitions are shown below.

```
float table[50][50];
char page[24][80];
```

EXERCISE

1. Describe the array that is defined in each of the following statements.

char name[30];, float c[6];, int params[5] [5];

2. Write an appropriate array definition for each of the following problem situations.

(a) Define a one-dimensional, 12-element integer array called c. Assign the values 1, 4, 7, 10, . . . to the array elements.

(b) Define a one-dimensional character array called point. Assign the string "NORTH" to the array elements. End the string with the null character.

3.9 POINTERS

Simply stated, a pointer is an address. Instead of being a variable, it is a pointer to a variable stored somewhere in the address space of the program. It is always best to use an example to display a pointer.

Example of a program with some pointers in it.

```
main( ) /* illustratrion of pointer use */
{
int index,*pt1,*pt2;
index = 39; /* any numerical value */
pt1 = &index; /* the address of index */
pt2 = pt1;
printf("The value is %d %d %d\n",index,*pt1,*pt2);
*pt1 = 13; /* this changes the value of index */
printf("The value is %d %d %d\n",index,*pt1,*pt2);
```

```
        }
```

The following two rules are very important when using pointers and must be thoroughly understood.
1. A variable name with an ampersand in front of it defines the address of the variable and therefore points to the variable. You can therefore read line six as "pt1 is assigned the value of the address of "index".
2. A pointer with a "star" in front of it refers to the value of the variable pointed to by the pointer. Line nine of the program can be read as "The stored (starred) value to which the pointer "pt1" points is assigned the value 13". Now you can see why it is convenient to think of the asterisk as a star, it sort of sounds like the word store.

Shown below is a simple program that illustrates the relationship between two integer variables, their corresponding addresses and their associated pointers.

```
#include <stdio.h>
main( )
{
int u = 3;
int v;
int*pu;         /* pointer t o an integer */
int*pv;         /* pointer t o an integer */
pu = &u;        / * assign address of u t o pu */
v =*pu;         / * assign value of u t o v */
pv = &v;        /* assign address of v t o pv */
printf("\nu=%d &u=%x pu=%x *pu=%d', u, &u, pu, *pu);
printf("\n\nv=%d &v=%x pv=%x *pv=%d", v, &v, pv, *pv);
}
```

Note that pu is a pointer to u, and pv is a pointer to v. Therefore pu represents the address of u, and pv represents the address of v.Pointers are often passed to a function as arguments. This allows data items within the calling portion of the program to be accessed by the function, altered within the function, and then returned to the calling portion of the program in altered form. Pointers can work with arrays and even functions because you can return a function using a pointer to the calling function.

4.0 Conclusion
A good consistent practice will help you improve and diversify your knowledge and skill on C. Also be encouraged to read further and deeper beyond this material.

5.0 Summary
In this unit you have been given the introductory concept on programming with C. Also you have learnt on how to create a C source file, the concept of arrays and the introduction into some control structures in C.

6.0 Tutor Marked Assignment
1.0 What do you understand by Control Structure, Define the standard format of four different Control Structures in C.
2. Write an appropriate array definition for each of the following problem situations.

(a) Define a one-dimensional, 12-element integer array called c. Assign the values 1, 4, 7, 10, . . . to the array elements.
(b) Define a one-dimensional character array called point. Assign the string "NORTH" to the array elements. End the string with the null character.

7.0 Further Reading

1. Schuam Outlines Theory And Problems Of Programming With C, Second Edition by *Byron S. Gottfried, Ph.D.* ISBN 0-07-024035-3 Copyright © 1996, 1990 The McGraw-Hill Companies, Inc. 541 pages.
2. C programming Tutorial-http://www.iu.hio.no/~mark/CTutorial/CTutorial.html
3. C tutorial- http://einstein.drexel.edu/courses/Comp_Phys/General/C_basics/

MODULE 1 – Laboratory Exercises on Programming Languages

UNIT 3: Programming in C++

Contents Page

1.0 Introduction

C++ among many programming language is one that is flexible. C++ was built on an already existing programming language C, having close similarity in many aspects including its way of declaration, data type and control structure but the input and output conventions differs. C++ is a general purpose programming language that

– is a better C,
– supports data abstraction,
– supports object-oriented programming, and
– supports generic programming

C++ is case sensitive. In other words, uppercase and lowercase letters are considered to be different. A variable named age is different from Age, which is different from AGE. Some words are reserved by C++, and you may not use them as variable names. Another name for keywords is *reserved words*. These are keywords used by the compiler to control your program. Keywords include if, while, for, and main.
Note that you need to get yourself a copy of the development kit(possibly the Turbo C software). Install and launch,You should see a page like this:

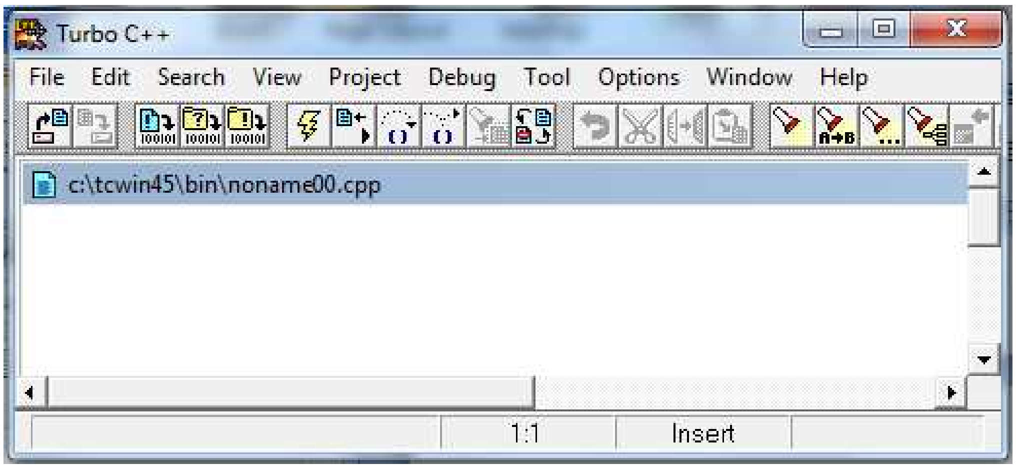

Turbo C++
File Edit Search View Project Debug Tool Options Window Help
c:\tcwin45\bin\noname00.cpp
1:1
Insert

Figure 1.3(a): The Development window for C++

2.0 Objectives

On completing this unit, you would be able to:

- Create and write a C++ source code
- Understand Control Structure in C++
- Understand Functions in C++
- Create Classes and Objects
- Understand Pointer and Arrays in C++

3.0 Creating a C++ source file

To start a C++ code, we can use a C++ compiler like Turbo C++ or Borland C++. Start with the following steps:

1. Open the C++ compiler, it looks like the figure below

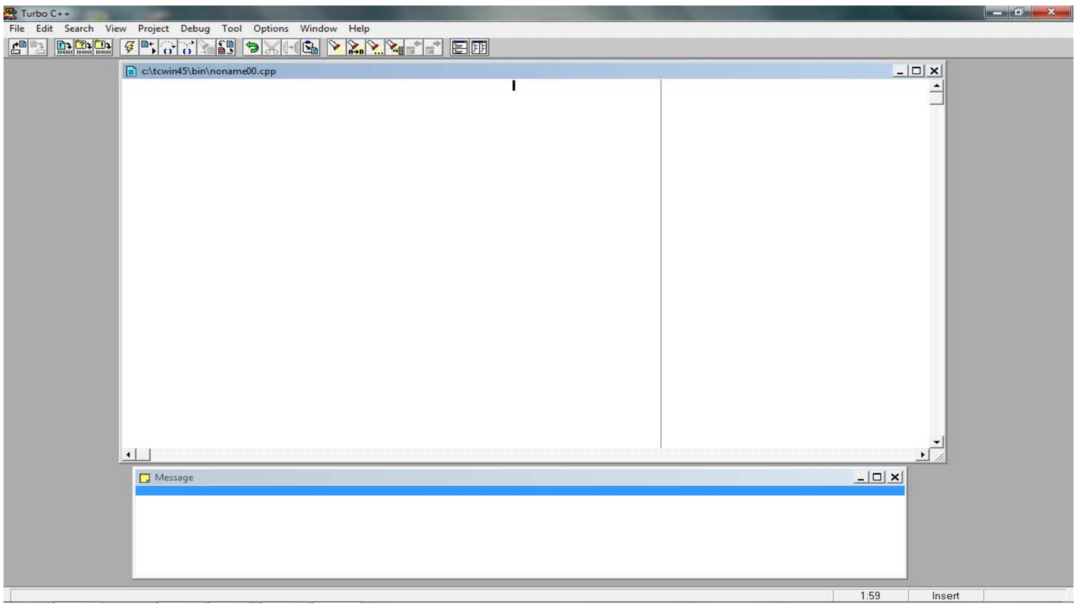
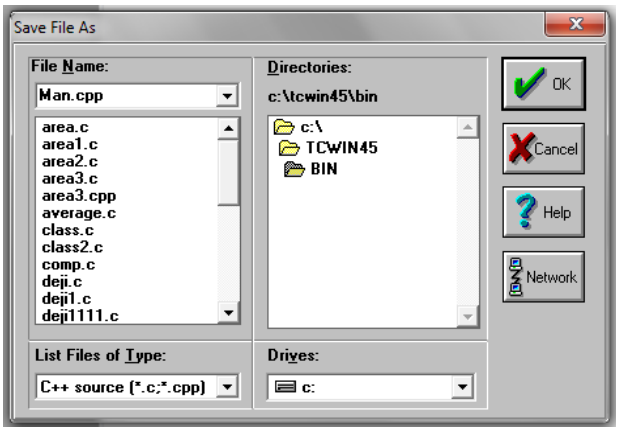

Turbo C++
File Edit Search View Project Debug Tool Options Window Help
c:\tcwin45\bin\noname00.cpp
Message
1:59
Insert

Figure 1.3(b): Turbo C++ compiler environment

2. Click on File menu from and the select the Save as option to save the file with a file name and an extension. The file name extension of C++ is ".cpp". For example man.cpp is the file name that indicates the type of file through its file extension as a C++ file.



File Name: Man.cpp

area.c
area1.c
area2.c
area3.c
area3.cpp
average.c
class.c
class2.c
comp.c
deji.c
deji1.c
deji1111.c

Directories: c:\tcwin45\bin

c:\
TCWIN45
BIN

List Files of Type: C++ source (*.c;*.cpp)

Drives: c:

Figure1.3(c): Menu for saving source file

3. The file is then saved. Writing and editing of codes is done in the blank/white space environment of the compiler.

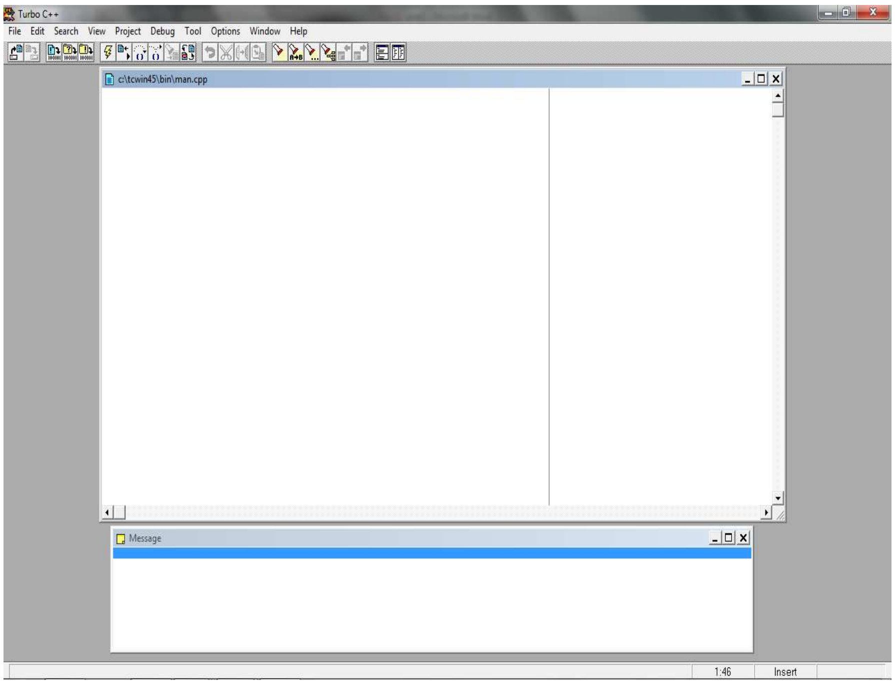

Turbo C++
File Edit Search View Project Debug Tool Options Window Help
c:\tcwin45\bin\man.cpp
Message
1:46
Insert

Figure1.3(d): The saved C++ file with a blank working space ready for use.

3.1 Elements of C++ program

The elements of C++ are like the main parts of the C++ codes. They can be divided into three parts;

A. Program Comments

Program comments are explanatory statements that you can include in the C++ code that you write to give an over view of what the program is all about. All programming languages allow for some form of comments. C++ compiler evaluates these lines of codes not as the executable codes. As a result, when we run our executable file, these lines of code are effectively ignored. It could be in this forms;

```
/* This program displays "Hello!"*/

// This program displays "Hello!”
```

B. Include Statements

This is one the most important line of code in C++ program, #include <iostream>.These are also known as *header files*. They are found in a special directory that is established when you install your compiler with each one providing the programs we write with a different piece of functionality that otherwise we would have to write ourselves. As to the particulars of iostream—the letters *io* in iostream refer to input-output, which is a computer term that describes how data is input into our programs and how we get information out of them. If we fail to include the iostream file in our program, we wouldn't be able to display the words 'Hello!'

C. **The main () Function**

The **main()** function is a section of code where the major instructions of a C++ program are placed. It contains the data that are declared as variables or constants, objects created from classes, namespace declaration, data type declaration and others. There are other functions that can be created in the program but the main() function must be created, without it there is no existence of the code.

```
        int main()
        {
         ---------
        }
```

The above *int main* shows that the returned value of the **main ()** function will be an *integer* value.

```
#include <iostream>
int main ()
{
        using namespace std;
        cout << "Welcome to my world!";
        return 0;
}
```

NOTE: All line of code or C++ statement must end with a semicolon (;) else the line of code will not be executed by the compiler.

3.2 Data: Constants and Variables

Data is input into a computer program to be processed into meaningful output.

Variables

Values—numbers and characters, for example—are stored in variables while the program is running. As the name implies, the values of the variables can change at any time. The types of variable include; *local variable-* which is a variable declared in a function, *global variables-* which are variables declared outside of a function and the variable can be accessed—updated and retrieved—from any function in the program, *class variables-* technically called *member variables*, which are variables declared in a C++ class and *static variables-*we can declare a data type for a variable and assign values to it. Example of *local variable*s and *global variable*;

```
1.int number1;
  number1 = 12;
```

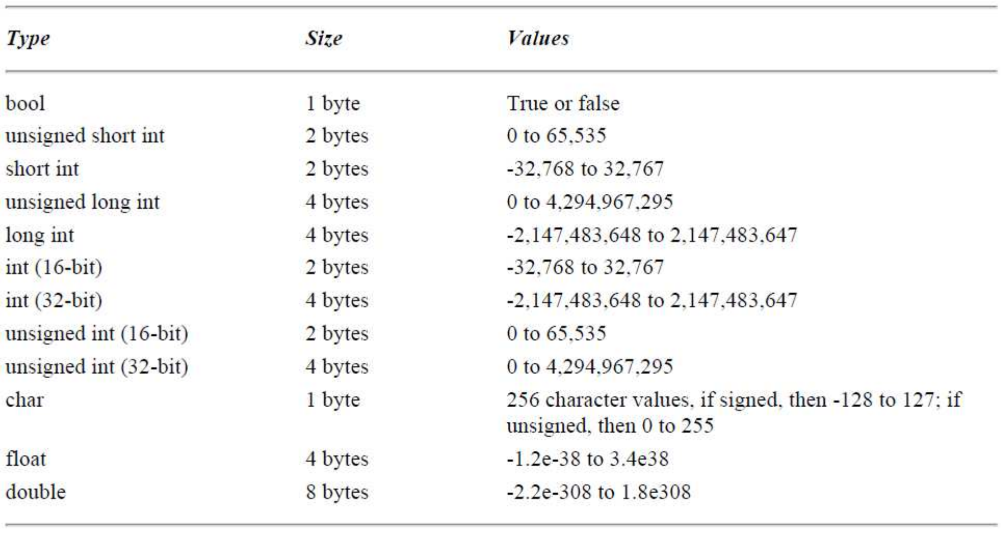
```
2. void smiley()
{
using namespace std;
int number = 44;
cout <<"smiley has executed and \n";
cout <<"the value of number is " << number << "\n";
}
```

In C++, variable names must start with either a letter or an underscore and may not contain spaces. Assigning an initial value to a variable is called *initialization*. Several other variable types are built into C++. They can be conveniently divided into integer variables (the type discussed so far), floating-point variables, and character variables.

Floating-point variables have values that can be expressed as fractions—that is, they are real numbers. Character variables hold a single byte and are used for holding the 256 characters and symbols of the ASCII and extended ASCII character sets. The ASCII character set is the set of characters standardized for use on computers. ASCII is an acronym for *American Standard Code for Information Interchange*. Nearly every computer operating system supports ASCII, although many support other international character sets as well.

The types of variables used in C++ programs are described in Table below. This table shows the variable type how much assumes it takes in memory, and what kinds of values can be stored in these variables. The values that can be stored are determined by the size of the variable types.

Table 1.3(a): Variable data types



Type	*Size*	*Values*
bool	1 byte	True or false
unsigned short int	2 bytes	0 to 65,535
short int	2 bytes	-32,768 to 32,767
unsigned long int	4 bytes	0 to 4,294,967,295
long int	4 bytes	-2,147,483,648 to 2,147,483,647
int (16-bit)	2 bytes	-32,768 to 32,767
int (32-bit)	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned int (16-bit)	2 bytes	0 to 65,535
unsigned int (32-bit)	4 bytes	0 to 4,294,967,295
char	1 byte	256 character values, if signed, then -128 to 127; if unsigned, then 0 to 255
float	4 bytes	-1.2e-38 to 3.4e38
double	8 bytes	-2.2e-308 to 1.8e308

Constants

A constant is similar to a variable in that it is given an easy-to-remember name and holds a value. Unlike a variable, however, once you assign a value to a constant, its value can never be changed. You declare a constant in a manner similar to a variable, you use the keyword *const* to designate a constant, like this-

```
const int BOOSTER = 100;
const float PI = 3.142;
```

Constants are named using all capital letters, like BOOSTER in the example above. C++ has two types of constants: literal and symbolic.

I. Literal Constants
A literal constant is a value typed directly into your program wherever it is needed, for example:

```
int myAge = 39;
```

myAge is a variable of type int; 39 is a literal constant. You are not allowed to assign a value to 39, and its value can't be changed. It is an *rvalue* because it can only appear on the right side of an assignment statement.

II. Symbolic Constants
A symbolic constant is a constant that is represented by a name, just as a variable is represented. Unlike a variable, however, after a constant is initialized, its value can't be changed. If your program has one integer variable named students and another named class, you could compute how many students you have, given a known number of classes, if you knew each class consisted of 15 students:

```
students = classes * 15;
```

In this example, 15 is a literal constant. Your code would be easier to read, and easier to maintain, if you substituted a symbolic constant for this value:

```
students = classes * studentsPerClass
```

If you later decided to change the number of students in each class, you could do so where you define the constant studentsPerClass without having to make a change every place you used that value. Two ways exist to declare a symbolic constant in C++. The old, traditional, and now obsolete way is with a preprocessor directive, #define and the const keyword.

3.3 Data Type: Numeric and Non-Numeric

C++ has a set of fundamental data types corresponding to the most common basic storage units of a Computer. Each C++ data type has unique memory requirements, along with capabilities and operations that you can perform on them. Declaring a data type that is not appropriate for the data you wish to store in it is a common beginner's error and can result in a range of problems- from your program not compiling at all, to it compiling and running but giving incorrect results. The most common data types are : Boolean type (*bool*), Character types (*char*), Integer types (*int*), Floating point types (*double*), Pointer types (*int**), and Array types (*char*[]). We can categorize them into Numeric and Non-Numeric data types.

3.4 Numeric Data Type

C++ numeric data types are assigned to variables to store a number that will later be used in a mathematical calculation. In C++, there are two categories of numeric data types: *integers*- which are whole numbers, such as 1 or 2, and *floating-point numbers*- which are numbers with a fractional part, such as 1.2 or 2.4. Within the integer category you have a choice of a data type that supports negative and positive numbers or positive numbers only. Always choose short, unsigned short, int, unsigned int, long, and unsigned long data types to store whole numbers (called integers) such as 23, 45, and 34470. short, int, and long allow for both positive and negative numbers. Choose float or double data types to store numbers with fractions such as

3.1416, 23.12 and 644.67. Select a data type appropriate for the values you wish to store in the variable. If the data type has a range much larger than you will need, you will waste valuable computer memory. When assigning a numeric value to a variable, don't include commas in the number. For example, 5,028 should be coded as 5028.

3.4.1 Non-numeric Data Type

This set of data type does not involve any arithmetic operation on it and hence it includes logical (boolean) data type, string and character data types.

- **Boolean Type**

A Boolean (*bool*) can have only two possible values: True and False. A Boolean is used to express the results of logical operations.

```
#include <iostream>
        int main()
{
        bool married = true;
        bool single = false;
        cout << "The value of married is " << married << endl;
        cout << "The value of single is " << single ;
        return 0;
    }
```

- **Character Types**

A variable of type *cha r* can hold a character of the implementation's character set. For example: *char ch* ='a'; A *char* has 8 bits so that it can hold one of 256 different values.

```
#include <iostream>
        int main()
{
        char character1 = 'a';
        cout << "The value of character1 is " << character1;
        return 0;
}
```

- **String Types**

To store more than one character in a variable, you will need to declare a String data type instead. String is not a fundamental data type. In C++, a String is an object, but a String variable is declared just like any other data type. A reference to the C++ 'string' library is the particular library that contains the code that allows us to use the string object. Some C++ compilers will complain that they can't find the string object if you don't include a reference to the library.

```
#include <iostream>
#include <string> // include for C++ standard string class
int main()
{
        string string1 = "John Smiley";
        cout << "The value of string1 " << string1;
        return 0;
}
```

3.4.2 Operations on Data: Arithmetic, Logical Operation

Operations on that data vary based on the data type. There several choices as to what to do with this result. You can choose to ignore it, discard it, assign it to a variable, or use it in an expression of some kind.

Determining the Order of Operations

All operators perform some defined function. In addition, every operator has a *precedence* — a specified place in the order in which the expressions are evaluated. Consider, for example, how precedence affects solving the following problem:

```
int var = 2 * 3 + 1;
```

If the addition is performed before the multiplication, the value of the expression is 2 times 4 or 8. If the multiplication is performed first, the value is 6 + 1 or 7.

The precedence of the operators determines who goes first.

Table 3-1 shows that multiplication has higher precedence than addition, so the result is 7. (The concept of precedence is also present in arithmetic. C++ adheres to the common arithmetic precedence.)

So what happens when we use two operators of the same precedence in the same expression? Well, it looks like this:

```
int var = 8 / 4 / 2;
```

But is this 8 divided by 2 or 4, or is it 2 divided by 2 or 1? When operators of the same precedence appear in the same expression, they are evaluated from left to right (the same rule applied in arithmetic). Thus, the answer is 8 divided by 4, which is 2 divided by 2 (which is 1).

The expression

```
x / 100 + 32
```

divides x by 100 before adding 32. But what if the programmer wanted to divide x by *100 plus 32?* The programmer can change the precedence by bundling expressions together in parentheses (shades of algebra!), as follows:

```
x/(100 + 32)
```

NOTE: In a given expression, C++ normally performs multiplication and division *before* addition or subtraction. Multiplication and division have higher precedence than addition and subtraction.

3.4.3 Arithmetic Operations

Arithmetic operations are performed on data stored in numeric variables or numeric constants. Arithmetic operations cannot be performed on any other kind of data, if you try, you'll either get a compiler error or a runtime error. This operation involves the following operators;

- **Addition Operator**

The addition operation (+) adds two operands, operands can be a number, a variable, a constant, or any expression that results in a number.

```
number3 = number1 + number2;
#include <iostream>
int main()
{
        int number1 = 12;
        int number2 = 23 ;
        cout << "The answer is " << (number1 + number2);
        return 0;
}
```

- **Subtraction Operator**

The subtraction operator (–) works by subtracting one operand from another and returning a result. In actuality, it subtracts the operand on its right side from the operand on its left.

```
#include <iostream>
int main()
{
        int number1 = 44;
        int number2 = 33;
        int result = 0;
        result = number1 - number2;
        cout << "The answer is " << result;
        return 0;
}
```

- **Multiplication Operator**

The multiplication operator (*) multiplies two operands,”

```
#include <iostream>
int main()
{
        int number1 = 4;
        int number2 = 3;
        int result = 0;
        result = number1 * number2;
        cout << "The answer is " << result;
        return 0;
}
```

- **Division Operator**

The division operator (/) works by dividing one operand by another and returning a result. In actuality, it divides the operand on the left side of the division operator by the operand on the right but the result must be assigned a data type that may not be an integer.

```
#include <iostream>
int main()
{
        int number1 = 5;
        int number2 = 2;
        int result = 0;
        result = number1 / number2;
        cout << "The answer is " << result;
        return 0;
}
```

- **Remainder Operator**

The remainder operation-sometimes called the *modulus operation*-deals with remainders. The result of the remainder operation is the remainder of a division operation. For instance, 7 divided by 2 is 3, with a remainder of 1.

```
#include <iostream>
int main()
{
	int number1 = 5;
	int number2 = 2;
	int result = 0;
	result = number1 % number2;
	cout << "The remainder is " << result;
	return 0;
}
```

- **Increment Operator**

This operator adds 1 to its value, usually performed on variables.

```
#include <iostream>
int main()
{
	int number1 = 5;
	number1++;
	cout << "The answer is " << number1;
	return 0;
}
```

- **Decrement Operator**

This is the opposite of the increment operator. It subtracts 1 from its value.

```
#include <iostream>
int main()
{
	int number1 = 5;
	number1--;
	cout << "The answer is " << number1;
	return 0;
}
```

Table 1.3(b): Table of Arithmetic Operators in C++

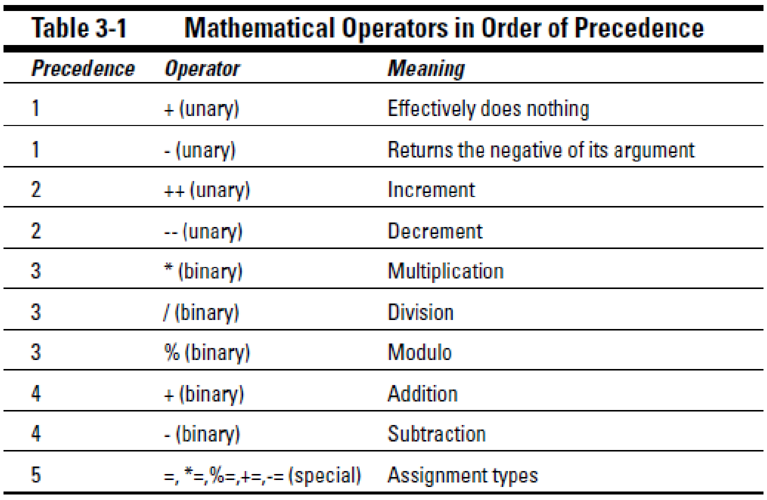


Table 3-1 Mathematical Operators in Order of Precedence

Precedence	*Operator*	*Meaning*
1	+ (unary)	Effectively does nothing
1	- (unary)	Returns the negative of its argument
2	++ (unary)	Increment
2	-- (unary)	Decrement
3	* (binary)	Multiplication
3	/ (binary)	Division
3	% (binary)	Modulo
4	+ (binary)	Addition
4	- (binary)	Subtraction
5	=, *=,%=,+=,-= (special)	Assignment types

3.4.4 Logical Operations

There is a whole other class of operators known as the *logical operators*. C++ programs have to make decisions. A program that can't make decisions is of limited use. The temperature-conversion program laid out in Chapter 1 is about as complex you can get without *some* type of decision-making. Invariably a computer program gets to the point where it has to figure out situations such as "Do *this* if the *a* variable is less than some value, do that *other* thing if it's not." That's what makes a computer appear to be intelligent — that it can make decisions.

Simple Logical Operator

The simple logical operators are show in the table below.

Table 1.3(c) : Table of Simple Logical Operators

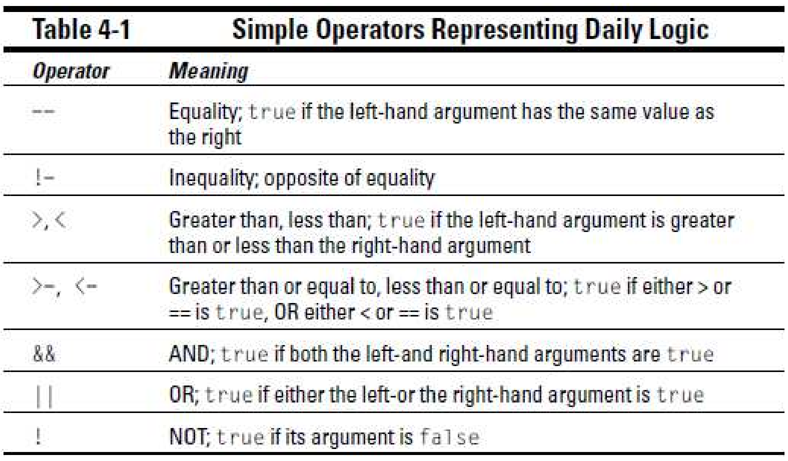


Table 4-1 Simple Operators Representing Daily Logic

Operator	*Meaning*
`==`	Equality; `true` if the left-hand argument has the same value as the right
`!=`	Inequality; opposite of equality
`>, <`	Greater than, less than; `true` if the left-hand argument is greater than or less than the right-hand argument
`>=, <=`	Greater than or equal to, less than or equal to; `true` if either > or == is `true`, OR either < or == is `true`
`&&`	AND; `true` if both the left-and right-hand arguments are `true`
`||`	OR; `true` if either the left-or the right-hand argument is `true`
`!`	NOT; `true` if its argument is `false`

The first six entries in the figure above are *comparison* operators. The equality operator is used to compare two numbers. For example, the following is true if the value of n is 0, and is false otherwise:

```
        n == 0;
```

Looks can be deceiving. Don't confuse the equality operator (==) with the assignment operator (=). Not only is this a common mistake, but it's a mistake that the C++ compiler generally cannot catch — that makes it more than twice as bad.

```
        n = 0; // programmer meant to say n == 0
```

The greater-than (>) and less-than (<) operators are similarly common in everyday life

3.4.5 Bitwise Logical Operations

All C++ numbers can be expressed in binary form. Binary numbers use only the digits 1 and 0 to represent a value. The following table defines the set of operations that work on numbers *one bit at a time,* hence the term *bitwise* operators.

Table 1.3(d): Table of Bitwise Logical Operators

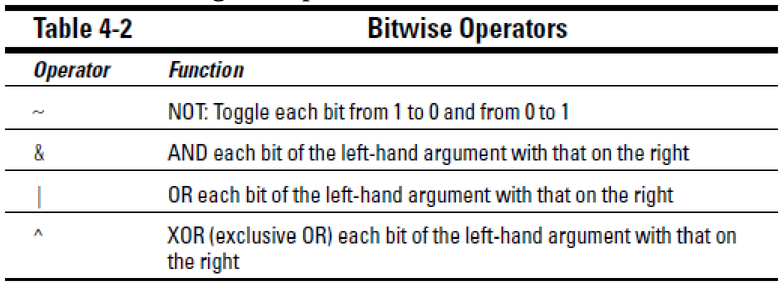


Table 4-2 Bitwise Operators

Operator	*Function*
~	NOT: Toggle each bit from 1 to 0 and from 0 to 1
&	AND each bit of the left-hand argument with that on the right
|	OR each bit of the left-hand argument with that on the right
^	XOR (exclusive OR) each bit of the left-hand argument with that on the right

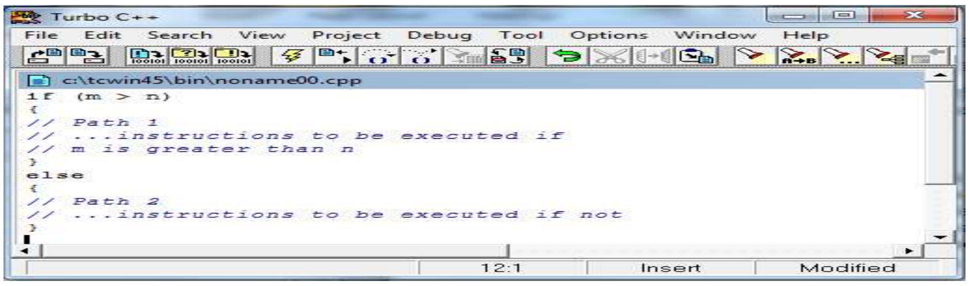
3.5 CONTROL STRUCTURE

Computer programs are all about making decisions. If the user presses a key, the computer responds to the command. This section focuses on controlling program flow. Flow-control commands allow the program to decide what action to take based on the results of the C++ logical operations performed. There are basically three types of flow-control statements: the branch, the loop, and the switch.

1. Branch Statement

The simplest form of flow control is the *branch statement.* This instruction allows the program to decide which of two paths to take through C++ instructions, based on the results of a logical expression.

Figure 1.3(e) illustrates the branch statement implemented using the ***if statement***:


Turbo C++
File Edit Search View Project Debug Tool Options Window Help
c:\tcwin45\bin\noname00.cpp
12:1
Insert
Modified

```
if  (m > n)
{
// Path 1
// ...instructions to be executed if
// m is greater than n
}
else
{
// Path 2
// ...instructions to be executed if not
}
```

Figure 1.3(e): The branch Statement

First, the logical expression m > n is evaluated. If the result of the expression is true, control passes down the path marked Path 1 in the previous snippet. If the expression is false, control passes to Path 2. The else clause is optional. If it is not present, C++ acts as if it is present but empty. Actually, the braces are optional (sort of) if there's only one statement to execute as part of the if. If you lose the braces, however, it's embarrassingly easy to make a mistake that the C++ compiler can't catsch.

2. Loops

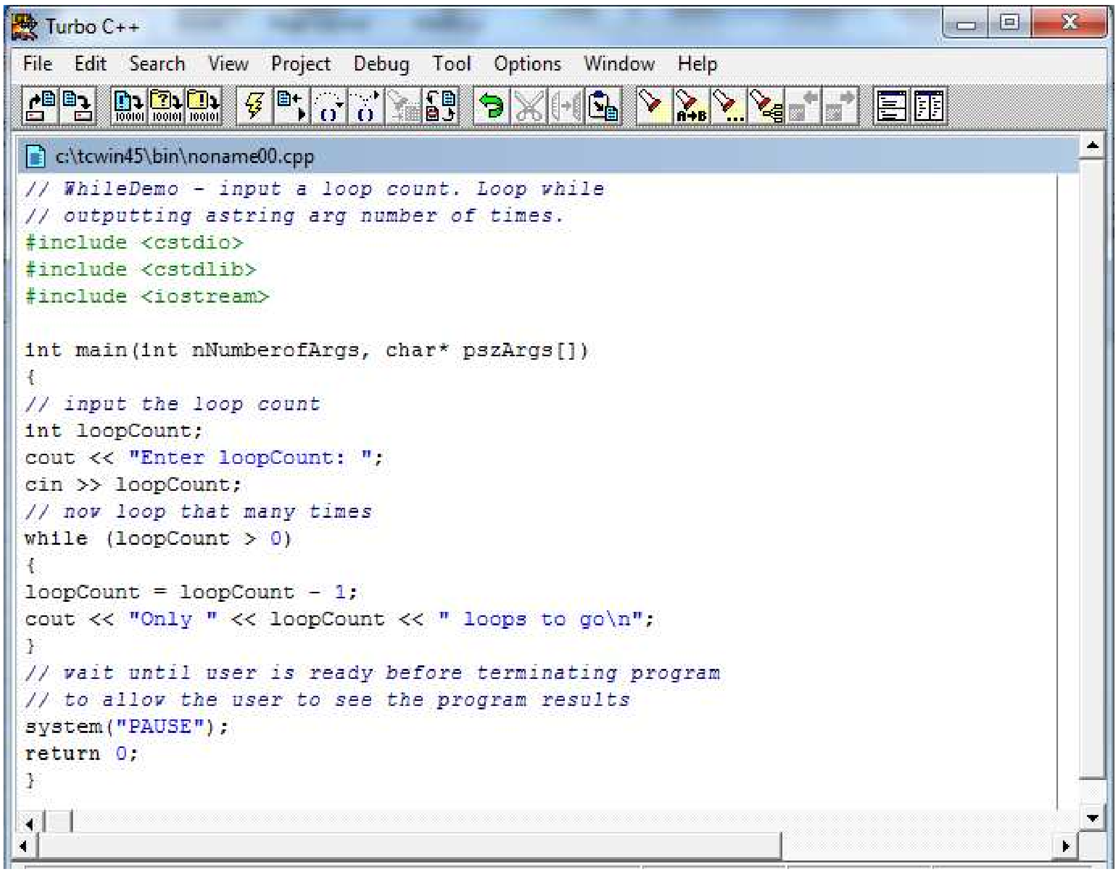
Branch statements allow you to control the flow of a program's execution from one path of a program or another. This is a big improvement, but still not enough to write full-strength programs. Executing the same command multiple times requires some type of looping statements.

While Loop

The simplest form of looping statement is the while loop. Here's what the while loop looks like:

```
while(condition)
{
        // ... repeatedly executed as long as condition is true
}
```

The *condition* is tested. This condition could be if var > 10 or if var1 == var2 or anything else you might think of. If it is true, the statements within the braces are executed. Upon encountering the closed brace, C++ returns control to the beginning, and the process starts over. The effect is that the C++ code *within the braces (i.e. code block) is executed repeatedly* as long as the condition is true. If the condition were true the first time, what would make it be false in the future? Consider the following example program:


Turbo C++
File Edit Search View Project Debug Tool Options Window Help
c:\tcwin45\bin\noname00.cpp

```
// WhileDemo - input a loop count. Loop while
// outputting astring arg number of times.
#include <cstdio>
#include <cstdlib>
#include <iostream>

int main(int nNumberofArgs, char* pszArgs[])
{
// input the loop count
int loopCount;
cout << "Enter loopCount: ";
cin >> loopCount;
// now loop that many times
while (loopCount > 0)
{
loopCount = loopCount - 1;
cout << "Only " << loopCount << " loops to go\n";
}
// wait until user is ready before terminating program
// to allow the user to see the program results
system("PAUSE");
return 0;
}
```

Figure 1.3(f): The while loop.

WhileDemo begins by retrieving a loop count from the user, which it stores in the variable loopCount. The program then executes a while loop. The while first tests loopCount. If loopCount is greater than zero, the program enters the body of the loop (the *body* is the code between the braces) where it decrements loopCount by 1 and outputs the result to the display. The program then returns to the top of the loop to test whether loopCount is still positive.
When executed, the program WhileDemo outputs the results shown in this next snippet. Here I entered a loop count of 5. The result is that the program loops 5 times, each time outputting a countdown.

```
Enter loopCount: 5
Only 4 loops to go
Only 3 loops to go
Only 2 loops to go
Only 1 loops to go
Only 0 loops to go
Press any key to continue . . .
```

If the user enters a negative loop count, the program skips the loop entirely. That's because the specified condition is never true, so control never enters the loop. A separate, less frequently used version of the while loop known as the do . . . while:

```
do
{
        // ...the inside of the loop
} while (condition);
```

The condition is only checked at the beginning of the while loop or at the end of the do . . . while loop. Even if the condition ceases to be true at some time during the execution of the loop, control does not exit the loop until the condition is retested.

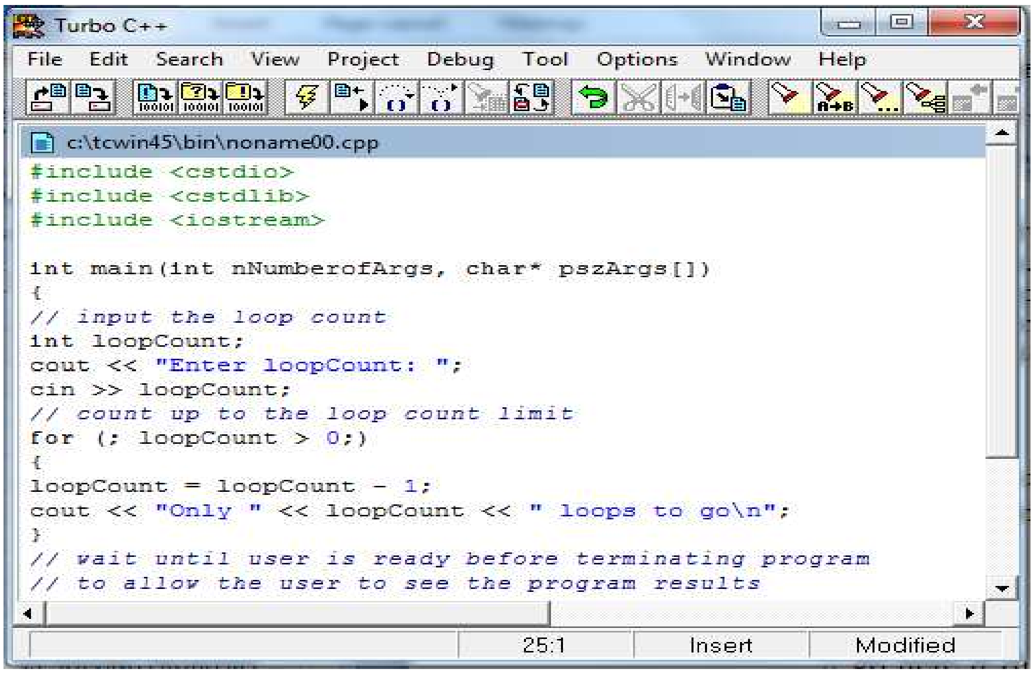
for loop

The most common form of loop is the *for* loop. The *for* loop is preferred over the more basic while loop because it's generally easier to read. The *for* loop has the following format:

```
for (initialization; conditional; increment)
{
        // ...body of the loop
}
```

Execution of the *for* loop begins with the *initialization clause,* which got its name because it's normally where counting variables are initialized. The initialization clause is only executed once when the *for* loop is first encountered. Execution continues with the *conditional clause.* This clause works a lot like the while loop: as long as the conditional clause is true, the *for* loop continues to execute.

The for loop is best understood by example. The following ForDemo program is nothing more than the WhileDemo converted to use the for loop construct:


Turbo C++
File Edit Search View Project Debug Tool Options Window Help
c:\tcwin45\bin\noname00.cpp
25:1
Insert
Modified

```
#include <cstdio>
#include <cstdlib>
#include <iostream>

int main(int nNumberofArgs, char* pszArgs[])
{
// input the loop count
int loopCount;
cout << "Enter loopCount: ";
cin >> loopCount;
// count up to the loop count limit
for (; loopCount > 0;)
{
loopCount = loopCount - 1;
cout << "Only " << loopCount << " loops to go\n";
}
// wait until user is ready before terminating program
// to allow the user to see the program results
```

Figure 1.3(g): The For Loop.

The program reads a value from the keyboard into the variable loopCount. The for starts out comparing loopCount to zero. Control passes into the for loop if loopCount is greater than zero. Once inside the for loop, the program decrements loopCount and displays the result. That done, the program returns to the for loop control. Control skips to the next line after the for loop as soon as loopCount has been decremented to zero. All three sections of a for loop may be empty.

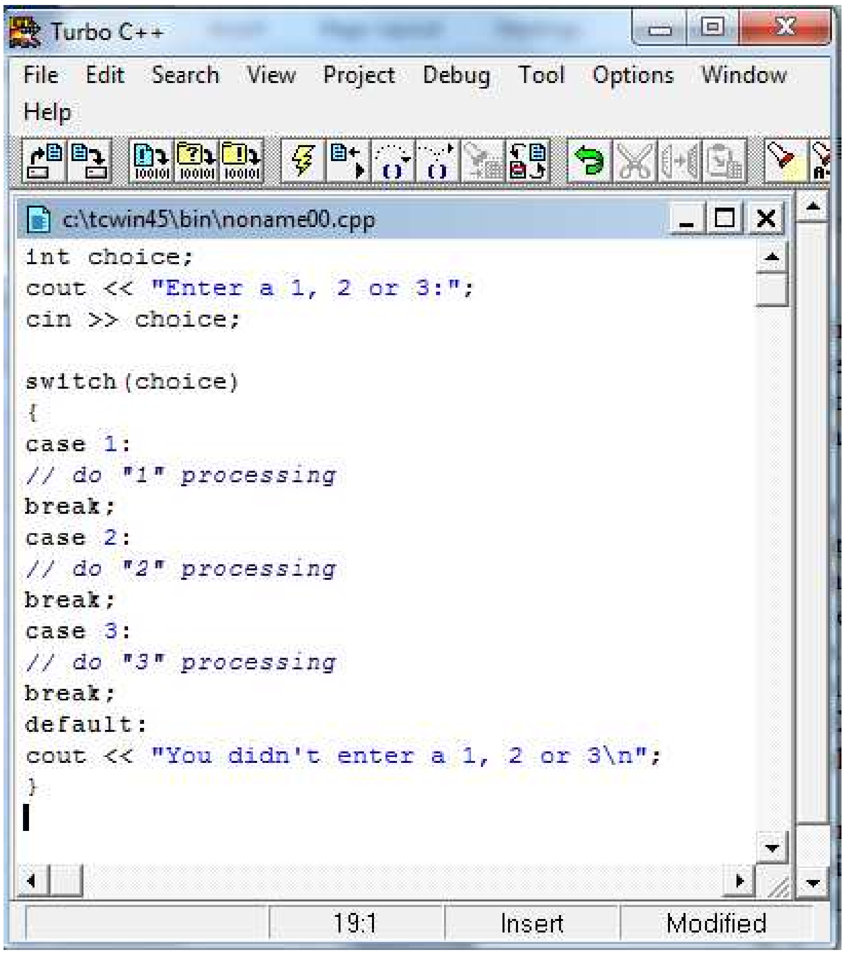
3. Switch Statement

One last control statement is useful in a limited number of cases. The switch statement resembles a compound if statement by including a number of different possibilities rather than a single test:

```
switch(expression)
{
        case c1:
        // go here if the expression == c1
                break;
        case c2:
        // go here if expression == c2
                break;
        default:
        // go here if there is no match
}
```

The value of expression must be an integer (int, long, or char). The case values c1, c2, and c3 must be constants. When the switch statement is encountered, the expression is evaluated and

compared to the various case constants. Control branches to the case that matches. If none of the cases match, control passes to the default clause.
Consider the following example code snippet:


Turbo C++
File Edit Search View Project Debug Tool Options Window Help
c:\tcwin45\bin\noname00.cpp
19:1
Insert
Modified

```
int choice;
cout << "Enter a 1, 2 or 3:";
cin >> choice;

switch(choice)
{
case 1:
// do "1" processing
break;
case 2:
// do "2" processing
break;
case 3:
// do "3" processing
break;
default:
cout << "You didn't enter a 1, 2 or 3\n";
}
```

Figure 1.3(h): The Swith Case Statement.

Once again, the switch statement has an equivalent, in this case multiple if statements; however, when there are more than two or three cases, the switch structure is easier to understand.
The break statements are necessary to exit the switch command. Without the break statements, control falls through from one case to the next.

3.6 Functions

Although main() is a function, it is an unusual one. To be useful, a function must be called, or invoked, during the course of your program. main() is invoked by the operating system.
A program is executed line by line in the order it appears in your source code until a function is reached. Then the program branches off to execute the function. When the function finishes, it returns control to the line of code immediately following the call to the function.

Example 1.3(a): Demonstrating a Call to a Function

```
1: #include <iostream.h>
2:
3: // function Demonstration Function
4: // prints out a useful message
5: void DemonstrationFunction()
6: {
7: cout << “In Demonstration Function\n”;
8: }
9:
10: // function main - prints out a message, then
11: // calls DemonstrationFunction, then prints out
12: // a second message.
13: int main()
14: {
15: cout << “In main\n” ;
16: DemonstrationFunction();
17: cout << “Back in main\n”;
18: return 0;
19: }
```

The function DemonstrationFunction() is defined on lines 5 to 8. When it is called, it prints a message to the screen and then returns. Line 13 is the beginning of the actual program. On line 15, main() prints out a message saying it is in main(). After printing the message, line 16 calls DemonstrationFunction(). This call causes the commands in DemonstrationFunction() to execute. In this case, the entire function consists of the code on line 7, which prints another message. When DemonstrationFunction() completes (line 8), it returns back to where it was called from. In this case, the program returns to line 17, where main() prints its final line.

3.6.1 Using Functions

Functions either return a value or they return void, meaning they return nothing. A function that adds two integers might return the sum, and thus would be defined to return an integer value. A function that just prints a message has nothing to return and would be declared to return void.
Functions consist of a header and a body. The header consists of the return type, the function name, and the parameters to that function. The parameters to a function enable values to be passed into the function. Thus, if the function were to add two numbers, the numbers would be the parameters to the function. A typical function header looks like:

```
int sum(int a, int b)
```

A parameter is a declaration of what type of value will be passed in; the actual value passed in by the calling function is called the argument. Many programmers use these two terms, parameters and arguments, as synonyms. Others are careful about the technical distinction. This lessons use the terms interchangeably.
The body of a function consists of an opening brace, zero or more statements, and a closing brace. The statements constitute the work of the function. A function may return a value, using a return statement. This statement will also cause the function to exit. If you do not put a return

statement into your function, it will automatically return void (no value) at the end of the function. The value returned must be of the type declared in the function header.

3.7 Array

An **array** consists of a set of objects (called its **elements**), all of which are of the same type and are arranged contiguously in memory. In general, only the array itself has a symbolic name, not its elements. Each element is identified by an **index** which denotes the position of the element in the array. The number of elements in an array is called its **dimension**. The dimension of an array is fixed and predetermined; it cannot be changed during program execution. Arrays are suitable for representing composite data which consist of many similar, individual items. Examples include: a list of names, a table of world cities and their current temperatures, or the monthly transactions for a bank account.

An array variable is defined by specifying its dimension and the type of its elements. For example, an array representing 10 height measurements (each being an integer quantity) may be defined as:

```
int heights[10];
```

The individual elements of the array are accessed by indexing the array. The first array element always has the index 0. Therefore, heights[0] and heights[9] denote, respectively, the first and last element of heights. Each of heights elements can be treated as an integer variable. So, for example, to set the third element to 177, we may write:

```
heights[2] = 177;
```

Attempting to access a nonexistent array element (e.g., heights[-1] or heights[10]) leads to a serious runtime error (called ‘index out of bounds’ error).

Processing of an array usually involves a loop which goes through the array element by element. Below is an illustration using a function which takes an array of integers and returns the average of its elements.

Example 1.3(b): function that returns the average of an array

```
const int size = 3;
double Average (int nums[size])
{
        double average = 0;
        for (register i = 0; i < size; ++i)
                average += nums[i];
return average/size;
}
```

Like other variables, an array may have an initializer. Braces are used to specify a list of comma-separated initial values for array elements. For example,

```
int nums[3] = {5, 10, 15};
```

initializes the three elements of nums to 5, 10, and 15, respectively. When the number of values in the initializer is less than the number of elements, the remaining elements are initialized to zero:

```
int nums[3] = {5, 10}; // nums[2] initializes to 0
```

When a complete initializer is used, the array dimension becomes redundant, because the number of elements is implicit in the initializer. The first definition of nums can therefore be equivalently written as:

```
int nums[] = {5, 10, 15}; // no dimension needed
```

Another situation in which the dimension can be omitted is for an array function parameter. For example, the Average function above can be improved by rewriting it so that the dimension of nums is not fixed to a constant, but specified by an additional parameter. Example 1.3 (c) illustrates this.

Example 1.3(c): The dimension of nums is not fixed.

```
double Average (int nums[], int size)
{
    double average = 0;
    for (register i = 0; i < size; ++i)
        average += nums[i];
    return average/size;
}
```

A C++ string is simply an array of characters. For example,

```
char str[] = "HELLO";
```

defines str to be an array of six characters: five letters and a null character. The terminating null character is inserted by the compiler. By contrast,

```
char str[] = {'H', 'E', 'L', 'L', 'O'};
```

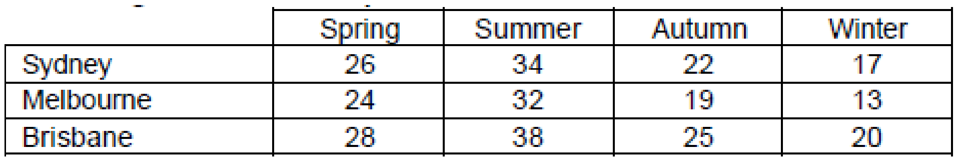
defines str to be an array of five characters. It is easy to calculate the dimension of an array using the sizeof operator. For example, given an array ar whose element type is Type, the dimension of ar is:

```
sizeof(ar) / sizeof(Type)
```

Multidimensional Arrays

An array may have more than one dimension (i.e., two, three, or higher). The organization of the array in memory is still the same (a contiguous sequence of elements), but the programmer's perceived organization of the elements is different. For example, suppose we wish to represent the average seasonal temperature for three major Australian capital cities.

Table 1.3(e): Average seasonal temperature.



	Spring	Summer	Autumn	Winter
Sydney	26	34	22	17
Melbourne	24	32	19	13
Brisbane	28	38	25	20

This may be represented by a two-dimensional array of integers:

```
int seasonTemp[3][4];
```

As before, elements are accessed by indexing the array. A separate index is needed for each dimension. For example, Sydney's average summer temperature (first row, second column) is given by seasonTemp[0][1].

The array may be initialized using a nested initializer:

```
int seasonTemp[3][4] = {
        {26, 34, 22, 17},
        {24, 32, 19, 13},
        {28, 38, 25, 20}
};
```

Because this is mapped to a one-dimensional array of 12 elements in memory, it is equivalent to:

```
int seasonTemp[3][4] = {
        26, 34, 22, 17, 24, 32, 19, 13, 28, 38, 25, 20
};
```

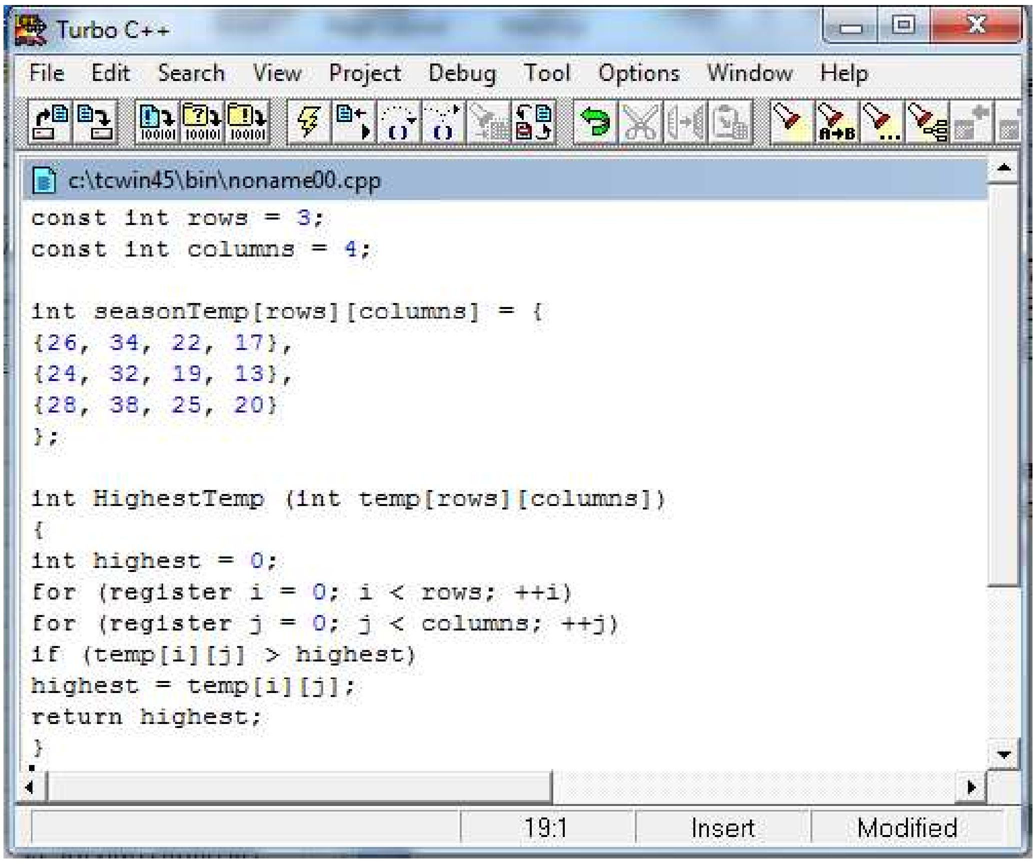
The nested initializer is preferred because as well as being more informative, it is more versatile. For example, it makes it possible to initialize only the first element of each row and have the rest default to zero:

```
int seasonTemp[3][4] = {{26}, {24}, {28}};
```

We can also omit the first dimension (but not subsequent dimensions) and let it be derived from the initializer:

```
int seasonTemp[][4] = {
        {26, 34, 22, 17},
        {24, 32, 19, 13},
        {28, 38, 25, 20}
};
```

Processing a multidimensional array is similar to a one-dimensional array, but uses nested loops instead of a single loop. Figure 1.3(i) illustrates this by showing a function for finding the highest temperature in seasonTemp.


Turbo C++
File Edit Search View Project Debug Tool Options Window Help
c:\tcwin45\bin\noname00.cpp
19:1
Insert
Modified

```
const int rows = 3;
const int columns = 4;

int seasonTemp[rows][columns] = {
{26, 34, 22, 17},
{24, 32, 19, 13},
{28, 38, 25, 20}
};

int HighestTemp (int temp[rows][columns])
{
int highest = 0;
for (register i = 0; i < rows; ++i)
for (register j = 0; j < columns; ++j)
if (temp[i][j] > highest)
highest = temp[i][j];
return highest;
}
```

Figure 1.3(i): Processing multidimensional array using nested for loops.

3.8 Pointers

A pointer is simply the *address* of a memory location and provides an indirect way of accessing data in memory. A pointer variable is defined to 'point to' data of a specific type. For example:

```
int *ptr1; // pointer to an int
char *ptr2; // pointer to a char
```

The *value* of a pointer variable is the address to which it points. For example, given the definitions

```
int num;
```

we can write:

```
ptr1 = &num;
```

The symbol & is the **address** operator; it takes a variable as argument and returns the memory address of that variable. The effect of the above assignment is that the address of num is assigned to ptr1. Therefore, we say that ptr1 points to num.

A simple integer pointer

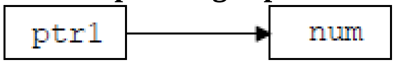


Figure 1.3(j): Assignment of the address of num to pointer ptr.

Given that ptr1 points to num, the expression *ptr1 dereferences ptr1 to get to what it points to, and is therefore equivalent to num. The symbol * is the **dereference** operator; it takes a pointer as argument and returns the contents of the location to which it points. In general, the type of a pointer must match the type of the data it is set to point to. A pointer of type void*, however, will match any type. This is useful for defining pointers which may point to data of different types, or whose type is originally unknown.

A pointer may be **cast** (type converted) to another type. For example,

ptr2 = (char*) ptr1; converts ptr1 to char pointer before assigning it to ptr2. Regardless of its type, a pointer may be assigned the value 0 (called the **null** pointer).

The null pointer is used for initializing pointers, and for marking the end of pointer-based data structures (e.g., linked lists).

Dynamic Memory

In addition to the program stack (which is used for storing global variables and stack frames for function calls), another memory area, called the **heap**, is provided. The heap is used for dynamically allocating memory blocks during program execution. As a result, it is also called **dynamic memory**. Similarly, the program stack is also called **static memory**. Two operators are used for allocating and de-allocating memory blocks on the heap. The new operator takes a type as argument and allocated a memory block for an object of that type. It returns a pointer to the allocated block. For example,

```
int *ptr = new int;
char *str = new char[10];
```

allocate, respectively, a block for storing a single integer and a block large enough for storing an array of 10 characters. Memory allocated from the heap does not obey the same scope rules as normal variables. For example, in

```
void Foo (void)
{
char *str = new char[10];
//...
}
```

when Foo returns, the local variable str is destroyed, but the memory block pointed to by str is *not*. The latter remains allocated until explicitly released by the programmer.

The delete operator is used for releasing memory blocks allocated by new. It takes a pointer as argument and releases the memory block to which it points. For example:

```
delete ptr; // delete an object
delete [] str; // delete an array of objects
```

Note that when the block to be deleted is an array, an additional [] should be included to indicate this.

Should delete be applied to a pointer which points to anything but a dynamically-allocated object (e.g., a variable on the stack)? a serious runtime error may occur. It is harmless to apply delete to the 0 pointer.

Dynamic objects are useful for creating data which last beyond the function call which creates them. Example 1.3(c) illustrates this using a function which takes a string parameter and returns a *copy* of the string.
Example 1.3(c):

```
1: #include <string.h>
2: char* CopyOf (const char *str)
3: {
4:        char *copy = new char[strlen(str) + 1];
5:        strcpy(copy, str);
6:        return copy;
7:        }
```

1. This is the standard string header file which declares a variety of functions for manipulating strings.
4. The strlen function (declared in string.h) counts the characters in its string argument up to (but excluding) the final null character. Because the null character is not included in the count, we add 1 to the total and allocate an array of characters of that size.
5. The strcpy function (declared in string.h) copies its second argument to its first, character by character, including the final null character.

Because of the limited memory resources, there is always the possibility that dynamic memory may be exhausted during program execution, especially when many large blocks are allocated and none released. Should new be unable to allocate a block of the requested size, it will return 0 instead. It is the responsibility of the programmer to deal with such possibilities. The exception handling mechanism of C++ provides a practical method of dealing with such problems.

Pointer Arithmetic

In C++ one can add an integer quantity to or subtract an integer quantity from a pointer. This is frequently used by programmers and is called pointer arithmetic. Pointer arithmetic is *not* the same as integer arithmetic, because the outcome depends on the size of the object pointed to. For example, suppose that an int is represented by 4 bytes. Now, given

```
char *str = "HELLO";
int nums[] = {10, 20, 30, 40};
int *ptr = &nums[0]; // pointer to first element
```

str++ advances str by one char (i.e., one byte) so that it points to the second character of "HELLO", whereas ptr++ advances ptr by one int (i.e., four bytes) so that it points to the second element of nums. Figure 1.3(k) illustrates this diagrammatically.

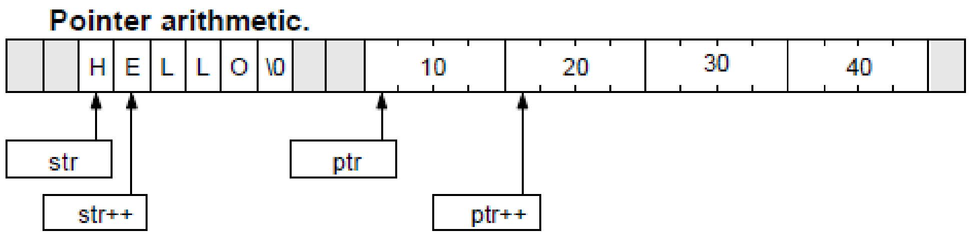


Figure 1.3(k): Result of pointer arithmetic.

It follows, therefore, that the elements of "HELLO" can be referred to as *str, *(str + 1), *(str + 2), etc. Similarly, the elements of nums can be referred to as *ptr, *(ptr + 1), *(ptr + 2), and *(ptr + 3).
Another form of pointer arithmetic allowed in C++ involves subtracting two pointers of the same type. For example:

```
int *ptr1 = &nums[1];
int *ptr2 = &nums[3];
int n = ptr2 - ptr1; // n becomes 2
```

Pointer arithmetic is very handy when processing the elements of an array.

Example 1.3(d) shows as an example a string copying function similar to strcpy.

Example 1.3(d): String copying function using pointer arithmetic.

```
1: void CopyString (char *dest, char *src)
2:{
3:      while (*dest++ = *src++);
4:}
```

Line 3 the condition of this loop assigns the contents of src to the contents of dest and then increments both pointers. This condition becomes 0 when the final null character of src is copied to dest.
In turns out that an array variable (such as nums) is itself the address of the first element of the array it represents. Hence the elements of nums can also be referred to using pointer arithmetic on nums, that is, nums[i] is equivalent to
*(nums + i).
The difference between nums and ptr is that nums is a constant, so it cannot be made to point to anything else, whereas ptr is a variable and can be made to point to any other integer. Example 1.3(e) shows how the HighestTemp function can be improved using pointer arithmetic.

Example 1.3(e): Using pointer arithmetic.

```
1: int HighestTemp (const int *temp, const int rows, const int columns)
2: {
3:      int highest = 0;
4:      for (register i = 0; i < rows; ++i)
5:              for (register j = 0; j < columns; ++j)
6:                      if (*(temp + i * columns + j) > highest)
7:                              highest = *(temp + i * columns + j);
8:	return highest;
9: }
```

1. Instead of passing an array to the function, we pass an int pointer and two additional parameters which specify the dimensions of the array. In this way, the function is not restricted to a specific array size.
6. The expression *(temp + i * columns + j) is equivalent to temp[i][j] in the previous version of this function.
HighestTemp can be simplified even further by treating temp as a one-dimensional array of row * column integers. This is shown in Example 1.3(f) below

Example 1.3(f):

```
int HighestTemp (const int *temp, const int rows, const int columns)
{
	int highest = 0;
	for (register i = 0; i < rows * columns; ++i)
		if (*(temp + i) > highest)
			highest = *(temp + i);
	return highest;
}
```

Function Pointers

It is possible to take the address of a function and store it in a function pointer. The pointer can then be used to indirectly call the function. For example,

```
int (*Compare)(const char*, const char*);
```

defines a function pointer named Compare which can hold the address of any function that takes two constant character pointers as arguments and returns an integer. The string comparison library function strcmp, for example, is such.Therefore:

```
Compare = &strcmp; // Compare points to strcmp function
```

The & operator is not necessary and can be omitted:

```
Compare = strcmp; // Compare points to strcmp function
```

Alternatively, the pointer can be defined and initialized at once:

```
int (*Compare)(const char*, const char*) = strcmp;
```

When a function address is assigned to a function pointer, the two types must match. The above definition is valid because strcmp has a matching function prototype:

```
int strcmp(const char*, const char*);
```

Given the above definition of Compare, strcmp can be either called directly or indirectly via Compare. The following three calls are equivalent:

```
strcmp("Tom", "Tim"); // direct call
(*Compare)("Tom", "Tim"); // indirect call
Compare("Tom", "Tim"); // indirect call (abbreviated)
```

A common use of a function pointer is to pass it as an argument to another function; typically because the latter requires different versions of the former in different circumstances. A good example is a binary search function for searching through a sorted array of strings. This function may use a comparison function (such as strcmp) for comparing the search string against the array strings. This might not be appropriate for all cases. For example, strcmp is case-sensitive. If we wanted to do the search in a non-case-sensitive manner then a different comparison function would be needed.
As shown in Example 1.3(g) below, by making the comparison function a parameter of the search function, we can make the latter independent of the former.

Example 1.3(g):

```
1: int BinSearch (char *item, char *table[], int n,
2: int (*Compare)(const char*, const char*))
3: {
4:      int bot = 0;
5:      int top = n - 1;
6:      int mid, cmp;
7:      while (bot <= top) {
8:          mid = (bot + top) / 2;
9:          if ((cmp = Compare(item,table[mid])) == 0)
10:         return mid; // return item index
11:         else if (cmp < 0)
12:                 top = mid - 1; // restrict search to lower half
13:         else
14:                 bot = mid + 1; // restrict search to upper half
15:     }
16:     return -1; // not found
17: }
```

1. Binary search is a well-known algorithm for searching through a *sorted* list of items. The search list is denoted by table which is an array of strings of dimension n. The search item is denoted by item.
2. Compare is the function pointer to be used for comparing item against the array elements.
7. Each time round this loop, the search span is reduced by half. This is repeated until the two ends of the search span (denoted by bot and top) collide, or until a match is found.
9. The item is compared against the middle item of the array.
10. If item matches the middle item, the latter's index is returned.
11. If item is less than the middle item, then the search is restricted to the lower half of the array.
14. If item is greater than the middle item, then the search is restricted to the upper half of the array.
16. Returns -1 to indicate that there was no matching item. The following example shows how BinSearch may be called with strcmp passed as the comparison function:

```
char *cities[] = {"Boston", "London", "Sydney", "Tokyo"};
cout << BinSearch("Sydney", cities, 4, strcmp) << '\n';
```

This will output 2 as expected.

4.0 Conclusion

A C++ program comprises the code written using correct syntax, comments, keywords, identifiers, fundamental data types, user-written data types and header files. Keywords are the reserved words that are part of the C++ language. Fundamental data types are built-in data types and can be used to develop more complex user-written data types. Identifier names are chosen by the programmer and must not be C++ keywords. Both identifiers and functions must be declared ahead of their use in a program.

A variable definition specifies the type of a variable, gives it a name, and sometimes can give it an initial value. Every variable declare must have a data type. The order in which operators are evaluated in an expression is significant and is determined by precedence rules. These rules divide the C++ operators into a number of precedence levels. Operators in higher levels take precedence over operators in lower levels.

Statements represent the lowest-level building blocks of a program. Arrays are suitable for representing composite data which consist of many similar, individual items. A **pointer** is simply the address of an object in memory and is useful for creating **dynamic** objects during program execution.

5.0 Summary

The difficulty in learning a complex subject, such as programming, is that so much of what you learn depends on everything else there is to learn. This unit introduced the numeric variables which are either integral (char, int, short int, and long int) or they are floating point (float and double). Numeric variables can also be signed or unsigned. You must declare a variable before it can be used, and then you must store the type of data that you've declared as correct for that variable. while loops check a condition, and if it is true, execute the statements in the body of the loop. do...while loops execute the body of the loop and then test the condition. for loops initialize a value and then test an expression. If an expression is true, the body of the loop is executed. This unit explains the use of the array data type in C++. How to declare and use array. Array could be a one-dimensional array or n-dimensional array, where n=2, 3…

In addition the concept of pointer was introduced. Pointer is a very powerful feature of C++

6.0 Tutor Marked Assignment

1. Modify the sample code (Helloworld.cpp) to print the message "**Welcome to CPP**"
2. What does #include do?
3. What is a variable and Why do you declare variable in program
4. How do you declare variable, give example?
5. Define variables to represent the following entities:
   i. Age of a person.
   ii. Income of an employee.
   iii. Number of words in a dictionary.
   iv. A letter of the alphabet.
   v. A greeting message.
6. Which of the following represent valid variable definitions?
   int n = -100;
   unsigned int i = -100;

```
signed int = 2.9;
long m = 2, p = 4;
int 2k;
double x = 2 * m;
float y = y * 2;
unsigned double z = 0.0;
double d = 0.67F;
float f = 0.52L;
signed char = -1786;
char c = '$' + 2;
sign char h = '\111';
char *name = "Peter Pan";
unsigned char *num = "276811";
```

7. Explain operator precedence

8. What are logical operators, write a small program to explain them.

9. What will be the value of each of the following variables after its initialization:

   i. `double d = 2 * int(3.14);`

   ii. `long k = 3.14 - 3;`

10. Write a program which inputs an integer value, checks that it is positive, and outputs its factorial, using the formulas:

$$factorial(0) = 1$$
$$factorial(n) = n \times factorial(n\text{-}1)$$

11. The following table specifies the major contents of four brands of breakfast cereals. Define a two-dimensional array to capture this data:

	Fiber	Sugar	Fat	Salt
Top Flake	12g	25g	16g	0.4g
Cornabix	22g	4g	8g	0.3g
Oatabix	28g	5g	9g	0.5g
Ultrabran	32g	7g	2g	0.2g

Write a function which outputs this table element by element.

12. Rewrite the following function using pointer arithmetic:

```
char* ReverseString (char *str)

{
        int len = strlen(str);
        char *result = new char[len + 1];
        for (register i = 0; i < len; ++i)
        result[i] = str[len - i - 1];
        result[len] = '\0';
        return result;
```

```
                        }
```
7.0 Further Reading

1. Learn to Program with C++ by John Smiley ISBN: 0-07-223040-1 Osborne/McGraw-Hill © 2003, 589 pages.

2. *The C++ Programming Language, Third Edition* by Bjarne Stroustrup. Copyright ©1997 by AT&T. Published by Addison Wesley Longman, Inc. ISBN 0201889544.

MODULE 1 – Laboratory Exercises on Programming Languages

UNIT 4: Programming in Java

Content Page

1.0 INTRODUCTION

Java is an object-oriented program used to create Web pages with dynamic and interactive content, develop large-scale enterprise applications, enhance the functionality of World Wide Web servers (the computers that provide the content we see in our Web browsers), provide applications for consumer devices (such as cell phones, pagers and personal digital assistants) and for many other purposes. Java programs consist of pieces called *classes*. Classes consist of pieces called *methods* that perform tasks and return information when they complete their tasks. The class libraries are also known as the *Java APIs (Application Programming Interfaces)*. The Java compiler, **javac**, is not a traditional compiler in that it does not convert a Java program from source code into native machine code for a particular computer platform. Instead, the Java compiler translates source code into byte codes. Byte codes are the language of the Java Virtual Machine—a program that simulates the operation of a computer and executes its own machine language.
Note: you would need to download and install the Java 2 Standard edition kit(jdk) and NETBEANS IDE for this course.
After installation the NETBEANS IDE looks like this:

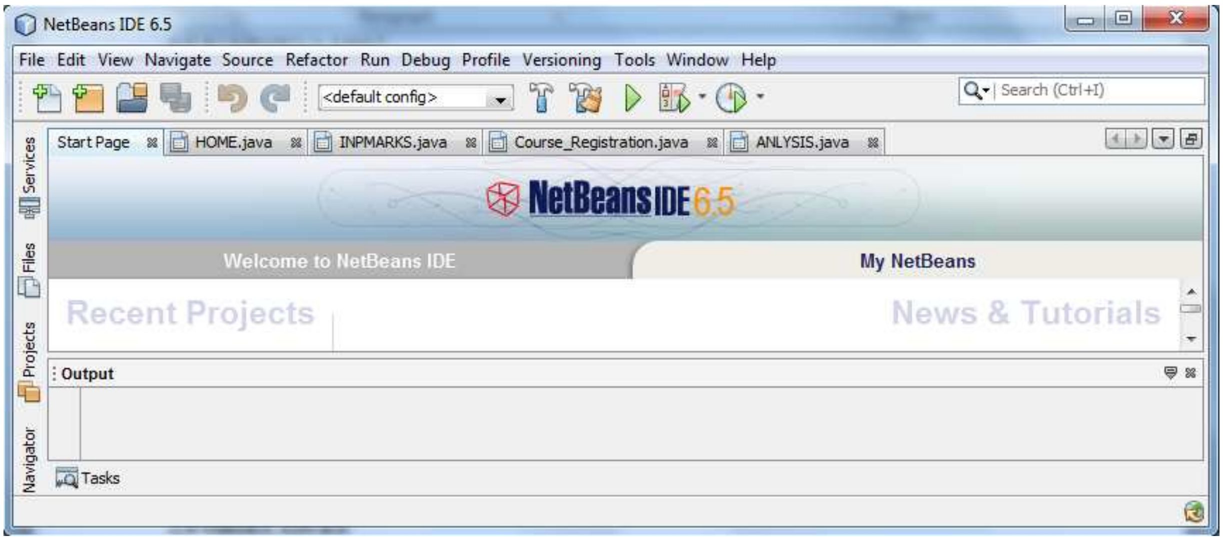


Figure 1.4(a): The NETBEANS IDE Start Up page.

2.0 OBJECTIVES

On completing this unit, you would be able to:

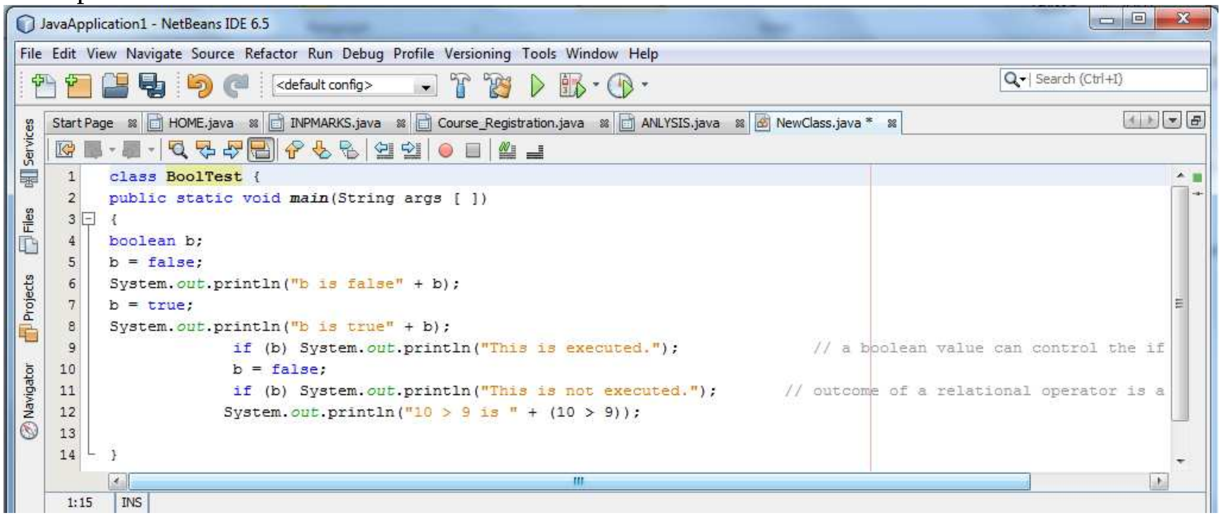
- Create and write a Java source code
- Understand Control Structure in Java
- Understand Methods in Java
- Create Methods and Classes
- Understand Arrays in Java

3.0 DATA TYPES

The data types in Java are the building blocks for the java program. Data types are portable across all computer platforms that support Java. This and many other portability features of Java enable programmers to write programs once, without knowing which computer platform will execute the program. Each data type has its size in bits. Data types includes; Boolean, Character, Floating point, and so on.

BOOLEAN TYPE

A Boolean can have only two possible values: True and False. A Boolean is used to express the results of logical operations. Boolean is also the type *required* by the conditional expressions that govern the control statements such as **if** and **for**.
Example 1.0// Demonstrate boolean values.



```java
class BoolTest {
public static void main(String args [ ])
{
boolean b;
b = false;
System.out.println("b is false" + b);
b = true;
System.out.println("b is true" + b);
             if (b) System.out.println("This is executed.");          // a boolean value can control the if
             b = false;
             if (b) System.out.println("This is not executed.");       // outcome of a relational operator is a
            System.out.println("10 > 9 is " + (10 > 9));

}
```

Figure 1.4(b): The use of Boolean Data type.

Output Generated:

```
b is false
b is true
This is executed.
10 > 9 is true
```

INTEGERS DATA TYPE

Java defines four integer types: ***byte***, ***short***, ***int***, and ***long***. All of these are signed, positive and negative values. Java does not support unsigned, positive-only integers.

- **byte**

The smallest integer type is ***byte***. This is a signed 8-bit type that has a range from –128 to 127. Variables of type **byte** are especially useful when you're working with a stream of data from a network or file.

```
byte b, c;
```

- **short**

short is a signed 16-bit type. It has a range from –32,768 to 32,767. It is probably the least-used Java type.

```
short s;
short t;
```

- **int**

The most commonly used integer type is ***int***. It is a signed 32-bit type.

```
int lightspeed;
```

- **long**

long is a signed 64-bit type and is useful for those occasions where an ***int*** type is not large enough to hold the desired value. The range of a ***long*** is quite large.

```
long days;
```

FLOATING-POINT DATA TYPES

Floating-point numbers, also known as *real* numbers, are used when evaluating expressions that require fractional precision. There are two kinds of floating-point types, ***float*** and ***double***, which represent single- precision (that uses 32 bits of storage.) and double-precision(uses 64 bits to store a value.) numbers, respectively.

```
float hightemp, lowtemp;
double pi;
```

CHARACTERS

In Java, the data type used to store characters is **char**. Unlike C++, in Java **char** is a 16-bit type. Java uses Unicode to represent characters; which defines a fully international character set.

```
char ch1, ch2;
ch1 = 88;
ch2 =’Y’;
```

Though **char**s are not integers, in many cases you can operate on them as if they were integers.

Example 1.4(a): // char variables behave like integers.

```
class CharDemo2 {
public static void main(String args[])
{
char ch1;
ch1 = 'X';
System.out.println("ch1 contains " + ch1);
ch1++;          // increment ch1
System.out.println("ch1 is now " + ch1);
        }
}
```

Output generated is:

```
ch1 contains X
ch1 is now Y
```

In the program, **ch1** has the value *X* in ASCII, and then **ch1** is incremented. **ch1**contains *Y;* which is the next character in the ASCII.

ASSIGNMENT OPERATIONS ON DATA

There are several arithmetic assignment operators that we can assign to data, variables and expressions in Java to carry out arithmetic operations and they include;

Assume *int* a=3; b=6; c=9; f=16; g=12.

i. Addition Assignment Operator: denoted as "+=". For example: a += 7→ a = a + 7→ 10 to a.
ii. Subtraction Assignment Operator: denoted as "-=". For example: b -= 4→ b = b – 4→ 2 to b.
iii. Multiplication Assignment Operator: denoted as "*=". For example: c *= 5→ c = c * 5 →45 to c.
iv. Division Assignment Operator: denoted as "/=". For example: f /= 4→ f = f / 4→ 2 to f.
v. Remainder Assignment Operator: denoted as "%=". For example: g %= 9 →g = g % 9→ 3 to g.

Even though assignment operators are like normal binary operators- +, /, - ,* and %; they do not function in the same way. Other operators that are essential to the operations on data are; the **unary *increment operator*** (++***)***-rather than the expression **c = c + 1** or **c += 1** instead **++c**, and the **unary *decrement operator*** (--***)***-rather than the expression **c = c -1** or **c -= 1** instead **--c**. Apart from arithmetic operators we also have relational operators that express simple conditional statements; > (greater than), < (less than), >= (greater than or equal to) and <= (less than or equal to) and ==(equal to) and != (not equal).

LOGICAL OPERATORS

Java provides ***logical operators*** to enable programmers to form more complex conditions by combining simple conditions. The logical operators are **&&** (*logical AND*), **&** (*boolean logical AND*), **||** (*logical OR*), **|** (*boolean logical inclusive OR*), **^** (*boolean logical exclusive OR*) and **!** (*logical NOT*, also called *logical negation*).

Example 1.4(b): gender == 1 && age >= 65
semesterAverage >= 90 || finalExam >= 90

VARIABLES

Variables are the basic units of storage in a Java program, their values change but the data types do not change. They are defined by the combination of an identifier, a type, and an optional initializer. All variables must be declared before they can be used. The basic form of a variable declaration is shown below:

type identifier [= *value*][, *identifier* [= *value*] ...] ;

The *identifier* is the name of the variable. Initializing the variable is by specifying an equal sign and a value.

```
int cadence = 7;
int speed = 0;
int gear = 1;
```

The Java programming language defines the following kinds of variables:

1. **Instance Variables (Non-Static Variables):** Objects actually store their individual state in "non-static variables", that is, variables declared without the static keyword. Non-static variables are also known as *instance variables* because their values are unique to each *instance* of a class (to each object, in other words).

2. **Class Variables (Static Variables):** A *class variable* is any variable declared with the static modifier; this tells the compiler that there is exactly one copy of this variable in existence; regardless of how many times the class has been instantiated.

3. **Local Variables:** Similar to how an object stores its state in variables; a method will often store its temporary state in *local variables*. The syntax for declaring a local variable is similar to declaring a variable (for example, int count = 0;). There is no special keyword designating a variable as local; that determination comes entirely from the location in which the variable is declared - which is between the opening and closing braces of a method. As such, local variables are only visible to the methods in which they are declared; they are not accessible from the rest of the class.

4. **Parameters:** Recall that the signature for the main method is public static void main(String[] args). Here, the args variable is the parameter to this method. You may also occasionally see the term "member" used as well. A type's variables, methods, and nested types are collectively called its *members*.

Class Exercise: Try this: write a program on your own that has to do with calculating the sum of integers from number 1 to 10

Answer: // Calculate the sum of the integers from 1 to 10

```
     public class Calculate {
     public static void main( String args[] )
     {
     int sum, x;
     x = 1;
     sum = 0;
     while ( x <= 10 ) {
     sum += x;
     ++x;
   }
     System.out.println( "The sum is: " + sum );
 }
}
```

3.1 CONTROL STRUCTURE

Statements in a program are executed one after the other in the order in which they are written. Control structures in Java are usually used to write complex programs require continuous execution till a particular output is achieved. Two groups of control strictures are; i. Selection Control Structure and, ii. Repetition Control Structure

3.1.2 SELECTION CONTROL STRUCTURE

A selection structure is used to choose among alternative courses of action in a program. There are two type of selection structure; the *IF* structure and the *IF/ELSE* structure.

3.1.3 IF SELECTION STRUCTURE

The ***if*** structure is a single-entry/single-exit structure. The ***if*** selection structure performs an indicated action only when the given condition evaluates to true; otherwise, the action is skipped.

```
if ( studentGrade >= 60 )
System.out.println( "Passed" );
```

3.1.4 IF/ELSE SELECTION STRUCTURE

The ***if/else*** selection structure allows the programmer to specify that a different action is to be performed when the condition is true rather than when the condition is false.

```
if ( studentGrade >= 60 )
System.out.println( "Passed" );
else
System.out.println( "Failed" );
```

3.1.5 REPETITION CONTROL STRUCTURE

A *repetition structure* allows the programmer to specify that an action is to be repeated while some condition remains true. There are three types of repetition structure; the *while* structure, the *for* structure and the *do/while* structure.

3.1.6 WHILE REPETITION STRUCTURE

```
int product = 2;
while ( product <= 1000 )
product = 2 * product;
```

When the ***while*** structure is entered, **product** is 2. Variable **product** is repeatedly multiplied by 2, taking on the values 4, 8, 16, 32, 64, 128, 256, 512 and 1024 successively. When **product** becomes 1024, the condition **product <= 1000** in the **while** structure becomes **false**. This result terminates the repetition, with 1024 as **product**'s final value.

3.1.7 FOR REPETITION STRUCTURE

The ***for*** repetition structure handles all of the details of counter-controlled repetition. For example

1. The control variable from **1** to **100** in increments of **1**.

```
for (int i = 1; i <= 100; i++)
```

2. The control variable from **100** to **1** in increments of **-1** (i.e., decrements of **1**).

```
for (int i = 100; i >= 1; i--)
```

3.1.8 DO/WHILE REPETITION STRUCTURE

The **do**/**while** repetition structure is similar to the **while** structure. The **do**/**while** structure tests the loop-continuation condition *after* performing the body of the loop; When a **do**/**while** structure terminates, execution continues with the statement after the **while** clause.

```
int counter = 1,value;
do {
        value+=110 – (counter * 10);
         ++counter;
 } while ( counter <= 10 );
```

3.2 ARRAYS

A*rray* is a group of like data type variables that are referred to by a common name. Arrays of any type can be created and may have one or more dimensions. Arrays are fixed-length entities they remain the same length once they are created. A specific element in an array is accessed by its index. The elements of an array can be either primitive types or reference types (including arrays).

DECLARING AND CREATING ARRAYS

```
int c[] = new int[ 12 ];
```

In the declaration, the square brackets following the variable name *c* indicate that *c* is a variable that will refer to an array (i.e., the variable will store an array reference). In the assignment statement, the array variable *c* receives the reference to a new array of 12 *int* elements.

3.2.1 ONE-DIMENSIONAL ARRAY

A *one-dimensional array* is, essentially, a list of like-typed variables. The general form of a one-dimensional array declaration is *type var-name*[ ];

```
int month_days[];
```

Example 1.4(c); // Average an array of values.

```
fclass Average {
public static void main(String args[]) {
        double nums[] = {10.1, 11.2, 12.3, 13.4, 14.5};
        double result = 0;
        int i;
        for(i=0; i<5; i++)
        result = result + nums[i];
        System.out.println("Average is " + result / 5);
  }
}
```

3.2.2 MULTIDIMENSIONAL ARRAYS

Multidimensional arrays are actually arrays of arrays in Java, in this type of array; the rows are allowed to vary in length. For example, declares a two-dimensional array;

```
int twinD[ ] [ ] = new int[2][3];
```

This allocates a 2 by 3 array and assigns it to *twinD*. Internally this matrix is implemented as an *array* of *arrays* of **int**.

Example:

```
class MultiDimArrayDemo {
public static void main(String[] args) {
String[][] names = {{"Mr. ", "Mrs. ", "Ms. "},
{"Smith", "Jones"}};
System.out.println(names[0][0] + names[1][0]); //Mr. Smith
System.out.println(names[0][2] + names[1][1]); //Ms. Jones
}
}
```

The output from this program is:

```
Mr. Smith
Ms. Jones
```

3.3 JAVA PACKAGES

A *package* is a set of classes that are stored in a directory, which has the same name as the package name. Packages enable you to organize the class files provided by Java. Conceptually you can think of packages as being similar to different folders on your computer. You might keep HTML pages in one folder, images in another, and scripts or applications in yet another. Because software written in the Java programming language can be composed of hundreds or *thousands* of individual classes, it makes sense to keep things organized by placing related classes and interfaces into packages.

The Java packages are classified into the following two categories:

I. Java Application Programming Interface (API) packages: The Java API consists of various packages, such as java.lang, java.util, java.io, java.awt, java.net, and java.applet.

II. Java user-defined packages: The packages that a user creates are called user defined packages. The user-defined packages can be imported in any Java program.

A real life example of packages- Consider a library with a large, unordered collection of books. Accessing and searching books from such a library is a time consuming activity. To avoid this problem, the books in the library should be well categorized and arranged in sections. Java provides classes same as a library provides books.

3.3.1 JAVA APPLICATION PROGRAMMING INTERFACE (API) PACKAGES

The Java API contains classes that are grouped in different packages in accordance to their functions, such as java.lang package provides various fundamental classes of Java. The java.util package provides various utility classes, such as Date, Calendar, and Dictionary. The java.applet package provides the basic functions for creating applets.

The following table lists the various in-built packages of Java:

1. **java.lang:** Provides various classes, such as Object, System, and Class.

2. **java.util:** Provides various classes that support collection or groups of objects, such as hash tables, String parsing, and system properties.
3. **java.io:** Defines two streams, InputStream and OutputStream that determine the flow of bytes from a source to destination.
4. **java.awt:** Provides classes to implement graphical user interface, such as creating buttons, check boxes, text boxes, menus, and list boxes.
5. **java.net:** Provides classes that support network programming, such as Socket, ServerSocket, and DatagramSocket.
6. **java.applet:** Provides the Applet class that provides methods to display images, play audio files, and obtain information about the applet environment. Some of these methods are play(), getImage(), getAppletInfo(), and getAudioClip().

THE import **STATEMENT FOR JAVA API PACKAGE**

The import statement is used to include a Java package in a program. The following syntax shows how to import a package in a Java program:

```
import<package_name>.*;
import<package_name>.<class_name>;
```

In the preceding syntax, the name of a package is followed by an asterisk (*) or by the name of a class in the package, which you need to import in the program. The dot operator (.) separates the elements in the syntax statement, such as package_name, and class name.
For example, the following syntax shows how to import the AWT package in a Java program:

```
import java.awt. *; // Imports all the classes of the awt package
import java.awt.Button; // Imports only the Button class of the awt
package
```

In the preceding syntax, an asterisk (*) at the end of the import statement indicates that all the classes of the AWT package have to be imported. The second statement indicates that only the Button class from the AWT package has to be imported.

3.3.2 USER-DEFINED PACKAGES

When you write a Java program, you create many classes. You can organize these classes by creating your own packages. The packages that you create are called user-defined packages, and contain one or more classes that can be imported in a Java program. Java provides the feature of creating a package to organize the related classes. You create a user-defined package by using the keyword, package. The package declaration must be at the beginning of the source file. You can make only one package declaration in a source file. The following syntax shows how to create a user-defined package:

```
package <package_name>
// Class definition
public class <classname1>
{
// Body of the class.
}
public class <classname2>
{
// Body of the class.
}
```

The file containing the package is saved as **.java** file. After compiling the source code, the **.class** file is created that is stored in the directory having the same name as the package name. To create a user-defined package, perform the following steps:

1. Create a source file containing the package definition.
2. Create a folder having the same name as package name and save the source file within the folder.
3. Compile the source file.

You can use the following syntax to create a user-defined package to include the Student class (Containing student information in a department):

```
package app.stdDetails;
public class Student
{
// Body of the class.
}
```

THE import **STATEMENT FOR USER-DEFINED PACKAGES**

The following syntax shows how to include the user-defined package, stdDetails from the app directory in a program:

```
import app. stdDetails. Student
```

In the preceding syntax, the Studentclass is imported from the stdDetails package. You can use the following syntax to implement the user-defined package, stdDetails:

```
import app.stdDetails. Student;
public class Director extends Student
{
// Body of the class.
}
```

When you compile the file containing the package statement, the .class file is saved in a directory with the name of the package. For example, the Student.class file will exist in the following directory:

```
<DIR>\app\stdDetails
```

DIR is the path to the directory where the file is saved. When you compile the program using the –d option, the compiler creates the package directory and moves the compiled class file to it using the following syntax to compile the program: javac –d .Employee.java

DIR is the current working directory. You can use the following code to create a user-defined package, BankDet, and implement it:

```
package BankDet;
//Defining class
public class SamplePack
{
String custname;
int acct_no;
int cust_id;
public SamplePack(String n1,int r1,int r2)
{
custname=n1;
acct_no=r1;
cust_id=r2;
}
public void display()
{
System.out.println();
System.out.println("**********CUSTOMER
INFORMATION**********");
System.out.println();
System.out.println();
System.out.println(" "+"Name of the Customer: "+custname);
System.out.println(" "+"Account Number :"+acct_no);
System.out.println(" "+"Customer ID:" +cust_id);
 }
}
```

Compile the preceding code snippet using the following syntax:

javac SamplePack.java

You can use the following code to implement the user-defined package, BankDet in a program:

```
import BankDet.SamplePack;
public class Bank
{
public static void main(String args[])
{
//Creating instance of the class 'SamplePack'.
SamplePack myobj=new SamplePack("John Smith",1001,1002);
myobj.display();
  }
}
```

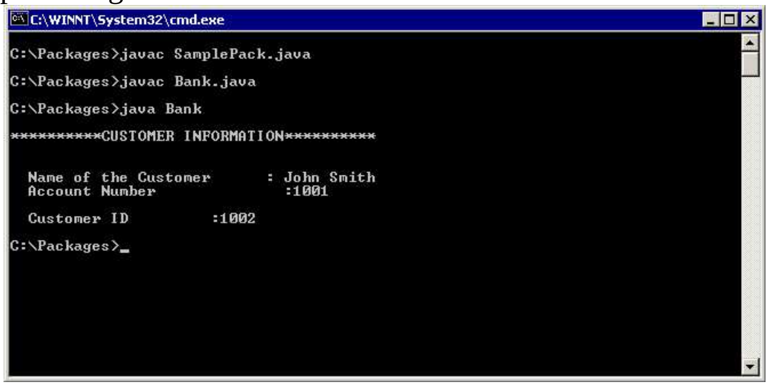
Compile the preceding code using the following syntax:

javac Bank.java

Execute the preceding code using the following syntax:

java Bank

The output of the preceding code is:



```
C:\Packages>javac SamplePack.java

C:\Packages>javac Bank.java

C:\Packages>java Bank
**********CUSTOMER INFORMATION**********


  Name of the Customer      : John Smith
  Account Number              :1001

  Customer ID         :1002

C:\Packages>_
```

Figure 1.4(c): Implementing User-Defined Package

Exercises:

1. Identify the correct order of steps to be taken to create a user-defined package:

1. Create a source file containing the package definition.
2. Compile the source file.
3. Create a folder having the same name as package name and save the source file within the folder.

Ans: a. 1, 2, 3
b. 1, 3, 2
c. 2, 3, 1
d. 3, 2, 1

2. Which package defines classes that implement GUI, such as buttons, checkboxes, and combo boxes?

Ans: a. java.applet
b. java.io
c. java.awt
d. java.net

3. An institute assigns roll numbers of 5 digits to its students belonging to different semesters.

The first two digits of the roll number specifies the semester to which the student belongs. Write a program to determine the semester of a student using the roll number.

Solution

The steps to create the program are:

1. Write the code.
2. Compile the code.
3. Execute the code.

1. Write the code

You can use the following code to determine the semester of a student.

```
class SubStrings
{
public static void main(String args[])
{
String str = new String("02262");
String substr = str.substring(0,2);
System.out.println("The student belongs to " + substr + "
semester");
}
}
```

2. Compile the code

Type the following command to compile the **SubStrings.java** file from your DOS prompt:

```
javac SubStrings.java
```

3. Execute the code

Type the following command to execute the **SubStrings.java** file: **java SubStrings**

3.4 JAVA BINARY (I/O) STREAM CLASSES

File class to perform input/output (I/O) operations for file handling in Java. Input/Output (I/O) operations consist of two parts, data that is entered into a program through an input source, such as a keyboard and the processed data that is returned through an output source, such as a monitor. Thus we can implement file classes, random access files and use streams in Java.

3.4.1 IMPLEMENTING THE FILE CLASS

Accessing Files by Using the File Class

The File class in the java.io package provides various methods to access the properties of a file or a directory, such as file permissions, date and time of creation, and path of a directory. Constructors to create an instance of the File class are:

- File(File dirObj, String filename): Creates a new instance of the File class. The dirObj argument is a File object that specifies a directory. The filename argument specifies the name of the file.
- File(String directoryPath): Creates a new instance of the File class. The directoryPath argument specifies the path of the file.
- File(String directoryPath, String filename): Creates a new instance of the File class. The argument directoryPath specifies the path of the file, and the filename argument specifies the name of the file.

Methods of the File class:

- String getName():Retrieves the name of the specified file.
- String getParent():Retrieves the name of the parent directory.
- String getPath():Retrieves the path of the specified file.
- String[] list():Displays the list of files and directories inside the specified directory.

Accessing File Properties:

- boolean delete():The delete() method is used to delete a specified file. The delete() method returns true, if it successfully deletes the file otherwise false.
- boolean renameTo(File newName):The renameTo() method is used to rename a specified file. The renameTo() method returns true, if it successfully renames the file otherwise false.

3.4.2 IMPLEMENTING RANDOM FILES IN JAVA

Random access files enable you to access the contents of a file in a non-sequential manner. These files enable you to read and write text and bytes to any location in a file. These files implement the DataInput and DataOutput interfaces, which define the basic I/O operations.

Accessing Random Files

You access the random files in Java by using the RandomAccessFile class. The constructor throws FileNotFoundException, if the RandomAccessFile class is unable to retrieve the name of the file to be created. Constructors to create an instance of the RandomAccessFile class include:

• RandomAccessFile(File fileObj, String mode): Creates an instance of the random access file.
• RandomAccessFile(String name, String mode): Creates an instance of the random access file.

Methods of the RandomAccessFile class
• void close() throws IOException: Closes the random access file and releases the system resources, such as streams and file pointers associated with the file.
• long getFilePointer() throws IOException: Retrieves the current position of the file pointer in the specified file.
• int skipBytes(int n) throws IOException: Ignores the number of bytes from a file specified by the n argument.
• long length() throws IOException: Retrieves the length of the specified file
• void seek(long position) throws IOException: Sets the current location of the file pointer at the specified position.

3.4.3 USING STREAMS IN JAVA

Accessing Files Using the InputStream Class

Input streams are the byte streams that read data in the form of bytes. The InputStream class is an abstract class that enables its subclasses to read data from different sources, such as a file and keyboard and displays the data on the monitor.

Methods of the InputStream class
• int read(): Reads the next byte of data from an input stream. It returns –1 if it encounters the end of a stream.
• int read(byte b[]): Reads the number of bytes from the array specified by the b[] argument. The read() method returns –1 when the end of the file is encountered.
• int read(byte b[], int offset, int length) throws IOException: Reads the number of bytes from the array specified by the b[] argument. The argument offset specifies the starting position for the read operation, and length specifies the number of bytes to be read.
• available(): Returns the total number of bytes available for reading in a stream.
• long skip(long n) throws IOException: Ignores the specified number of bytes from an input stream.
• mark(int nbyte): Places a mark at the current point in the input stream and this mark remains until the specified data bytes are read.
• public final int readInt(): Retrieves an integer from the input stream.

THE FileInputStream CLASS

The FileInputStream class performs file input operations, such as reading data from a file. The object of the FileInputStream class is used to read bytes of data from files. You can use the following constructors to create an instance of the FileInputStream class:
1. FileInputStream(File f): Creates a file input stream that connects to an existing file to be used as data source. The file object f specifies the name of the file to which an input stream is connected.
2. FileInputStream(String s): Creates a file input stream that connects to an existing file to be used as data source. The path of the file in file system is given by the string argument.

3. FileInputStream(FileDescriptor fdobj): Creates a file input stream that connects to an existing file to be used as data source. The fdObj file descriptor argument represents an existing connection to the specified file.

Methods of the FileInputStream class:
• public int read(): Reads a byte of data from the input stream.
• public int read(byte b[], int offset, int length): Reads the specified number of bytes of data from the specified byte array. The offset argument specifies the starting position for reading the data from a file.
• public long skip(long n) throws IOException: Ignores the specified number of bytes of data from an input stream.

THE BufferedInputStream CLASS

The BufferedInputStream class accepts data from other input streams and stores it in a memory buffer. The BufferedInputStream class reads data specified in any data format. A buffer stores the bytes of data and enables you to perform I/O operations on various data bytes simultaneously. The BufferedInputStream class supports read(), skip(), mark(), and reset() methods. You can use the following constructors to create an instance of the BufferedInputStream class:
1. BufferedInputStream(InputStream is): Creates an input stream and adds a buffer to it. The default size of the buffer is 512 bytes.
2. BufferedInputStream(InputStream is, int size): Creates an input stream and adds a buffer of size specified by the size argument.

SYSTEM.IN OBJECT

The System class in the java.lang package has a static member variable, in that refers to the keyboard. The in variable is an instance of the InputStream class and is used to read data from the keyboard.

Accessing Files Using the OutputStream Class

Output streams are byte streams that write data in the form of bytes. The OutputStream class is an abstract class that enables its subclasses to write data to a file or screen.

Methods in the OutputStream class:
• write(): Writes the output to a file.
• write(byte b[]): Writes an array of bytes specified by the b argument to a file.
• write(byte b[], int offset, int length): Writes an array of bytes specified by the b argument to a file. The offset argument determines the starting position for the byte array. The length argument specifies the number of bytes to be written.
• close(): Closes the byte output stream.
• flush(): Clears the buffers by removing any buffered output written on the disk.

SYSTEM.OUT OBJECT

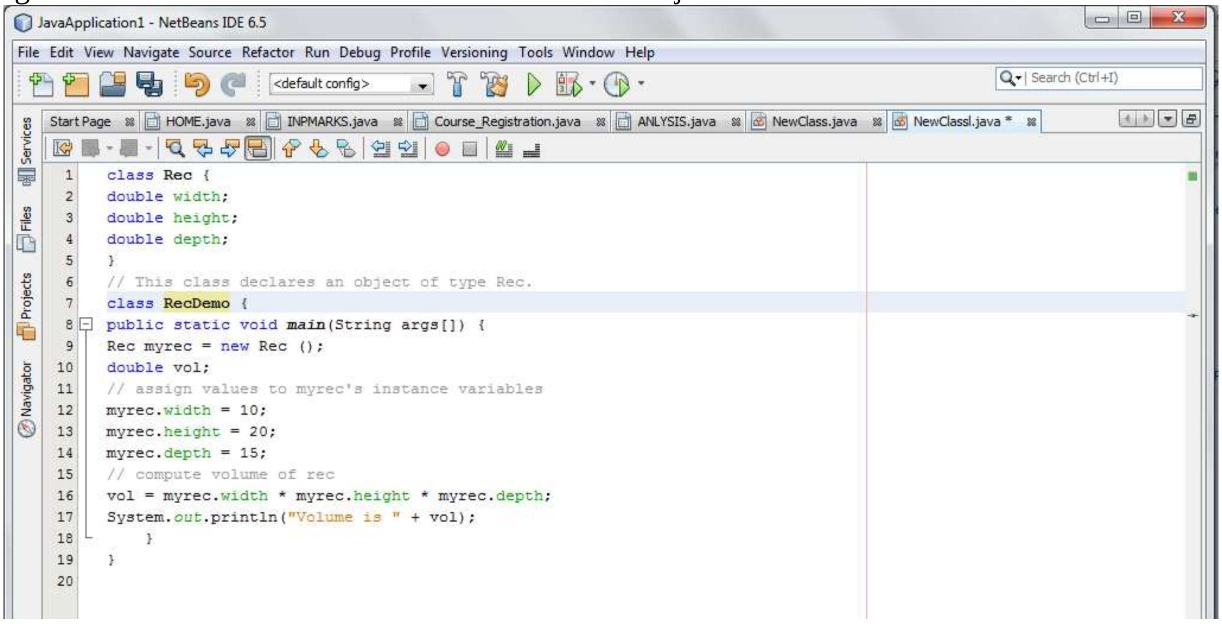
The System class in the java.lang package has a static member variable, out that represents the computer monitor. The out variable is an instance of the PrintStream class.

3.5 CLASSES

Class is at the core of Java. It is the logical construct upon which the entire Java language is built. It defines the shape and nature of an object. As such, the class forms the basis for object-oriented programming in Java. Any concept you wish to implement in a Java program must be encapsulated within a class. A class defines a new data type that can be used to create objects (an object is an *instance* of a class) of that type. A class is declared by use of the keyword ***class***. The general form of a **class** definition is shown here:

```
class classname {
        type instance-variable1;
        type instance-variable2;
        // ...
        type instance-variableN;
        …
    }
```

/* A program that uses the Rec class. Call this file RecDemo.java */



```
class Rec {
double width;
double height;
double depth;
}
// This class declares an object of type Rec.
class RecDemo {
public static void main(String args[]) {
Rec myrec = new Rec ();
double vol;
// assign values to myrec's instance variables
myrec.width = 10;
myrec.height = 20;
myrec.depth = 15;
// compute volume of rec
vol = myrec.width * myrec.height * myrec.depth;
System.out.println("Volume is " + vol);
    }
}
```

Figure 1.4(d): Here is a complete program that uses the Rec class

The data, or variables, defined within a **class** are called *instance variables.* The code is contained within *methods.* The methods and variables defined within a class are called *members* of the class. The instance variables are acted on and accessed by the methods defined for that class. The **main()** is a section of code where the major instructions of a Java program are placed. It contains the data that are declared as variables or constants, objects created from classes, instance variables and methods.

3.6 METHODS

Methods (called **functions** or **procedures** in other programming languages) allow the programmer to modularize a program by separating its tasks into self-contained units. These methods are sometimes referred to as **programmer-declared methods.** Methods are used for performing common mathematical calculations, string manipulations, character manipulations, input/output operations, database operations, networking operations, file processing, error checking and many other useful operations. The general form of a method is

```
type name(parameter-list) {
// body of method
}
```

Above, *type* specifies the data type returned by the method. This can be any valid type, including class types created. If the method does not return a value, its return type will be ***void***. The name of the method is specified by *name.* This can be any legal identifier. The *parameter-list* is a sequence of type and identifier pairs separated by commas. Parameters are essentially variables that receive the value of the *arguments* passed to the method when it is called. If the method has no parameters, then the parameter list will be empty. Methods that have a return type other than **void** return a value to the calling routine using the following form of the **return** statement:

```
return value;
```

Method works with classes to produce a full Java program. A good example of Method is

```
public static void main( String args[] )
```

It is left void so that the Java compiler can invoke the *main* without creating a class object. Also we can declare a method as *static*;To declare a method as *static*, place the keyword *static* before the return type in the method's declaration. You can call any *static* method by specifying the name of the class in which the method is declared, followed by a dot (.) and the method name, as in

```
ClassName.methodName( arguments )
```

For example: *Math* Class method provides a collection of methods that enable you to perform common mathematical calculations. Example 9.1; You can calculate the square root of 900.0 with the static method call

```
 Math.sqrt( 900.0 )
System.out.println( Math.sqrt( 900.0 ) );
```

The preceding expression evaluates to 30.0. Method sqrt takes an argument of type double and returns a result of type double.

3.6.1 METHODS TAKING PARAMETERS

Some methods don't need parameters but most do. Parameters allow a method to be generalized. A parameterized method can operate on a variety of data and/or be used in a number of slightly different situations.

Example 1.4(d):

```
 int square(int i)
{
return i * i;
}
```

From the above, **square( )** will return the square of whatever value it is called with thus making **square( )** a general-purpose method that can compute the square of any integer value.

Table 1.4(a): Summary of the Maths Class Method in Java

Math class methods		
Method	**Description**	**Example**
abs(x)	absolute value of x	abs(23.7) is 23.7 abs(-23.7) is 23.7
ceil(x)	rounds x to the smallest integer not less than x	ceil(9.2) is 10.0 ceil(-9.8) is -9.0
cos(x)	trigonometric cosine of x (x in radians)	cos(0.0) is 1.0
exp(x)	exponential method e^x	exp(1.0) is 2.71828 exp(2.0) is 7.38906
floor(x)	rounds x to the largest integer not greater than x	floor(9.2) is 9.0 floor(-9.8) is -10.0
log(x)	natural logarithm of x (base e)	log(Math.E) is 1.0 log(Math.E * Math.E) is 2.0
max(x, y)	larger value of x and y	max(2.3, 12.7) is 12.7 max(-2.3, -12.7) is -2.3
min(x, y)	smaller value of x and y	min(2.3, 12.7) is 2.3 min(-2.3, -12.7) is -12.7
pow(x, y)	x raised to the power y (i.e., x^y)	pow(2.0, 7.0) is 128.0
sin(x)	trigonometric sine of x (x in radians)	sin(0.0) is 0.0
sqrt(x)	square root of x	sqrt(900.0) is 30.0
tan(x)	trigonometric tangent of x (x in radians)	tan(0.0) is 0.0

3.6.2 THE this KEYWORD

A method will need to refer to the object that invoked it sometimes-Java defines the ***this*** keyword. ***this*** can be used inside any method to refer to the *current* object. **this** can be used anywhere to reference an object of the current class' type.

Example 1.4(e):

```
// A redundant use of this.
Rec(double w, double h, double d) {
        this.width = w;
        this.height = h;
        this.depth = d;
}
```

3.7 JAVA APPLICATIONS

Computer programmers create such applications by writing computer programs. For example, E-mail application helps one in sending and receiving e-mail, and the Web browser lets us view Web pages from Web sites around the world. A Java **application** is a computer program that executes when you use the **java command** to launch the Java Virtual Machine (JVM). Let us consider a simple application

Example 1.4(f): //Welcome1.java

```
// Text-printing program.
 public class Welcome1
 {
    // main method begins execution of Java application
      public static void main( String args[] )
     {
        System.out.println( "Welcome to Java Programming!" );
     } // end method main
} // end class Welcome1
```

Output Generated:

```
Welcome to Java Programming!
```

Our next application reads (or inputs) two integers typed by a user at the keyboard, computes the sum of the values and displays the result. This program must keep track of the numbers supplied by the user for the calculation later in the program.

Example 1.4(g): // Addition program that displays the sum of two numbers.

```
          import java.util.Scanner; // program uses class Scanner
      public class Addition
      {
           // main method begins execution of Java application
              public static void main( String args[] )
  {
  // create Scanner to obtain input from command window
   Scanner input = new Scanner( System.in );
```

```
        int number1; // first number to add
        int number2; // second number to add
        int sum; // sum of number1 and number2
        System.out.print( "Enter first integer: " ); // prompt
        number1 = input.nextInt(); // read first number from user
        System.out.print( "Enter second integer: " ); // prompt
        number2 = input.nextInt(); // read second number from user
        sum = number1 + number2; // add numbers
        System.out.printf( "Sum is %d\n", sum ); // display sum
    } // end method main
} // end class Addition
```

Output generated;

```
Enter first integer: 45
Enter second integer: 72
Sum is 117
```

4.0 CONCLUSION

Java has gained grounds in the Universities lately and has become one of the most used development tool especially in web tool development which students can also focus on but constant practice makes you better in its use.

5.0 SUMMARY

You have learnt the major data types and their operations on them. You also learnt the use of control structures and arrays in Java and the respective classes and their methods. Further practice will improve your knowledge on Java.

6.0 TUTOR MARKED ASSIGNMENT

1. Explain at least 4 data types in Java.
2. Write a Java program to print “my name is Sarah and I am beautiful”.
3. Write a program in Java to solve five factorial (5!) using any control structure.
4. What are arrays? State the types with definite Java syntax.
5. What are Java Packages?
6. Briefly give 10 examples of Maths Methods in Java and state the function of each.
7. What are Java Streams?

7.0 FURTHER READING

1. Java 2: The Complete Reference by Patrick Naughton and Herbert Schildt. ISBN: 0072119764 Osborne/McGraw-Hill © 1999, 1108 pages
2. *Thinking in Java, 2nd Edition*, (Prentice Hall, 2000), http://www.BruceEckel.com.
3. Learning the Java Language (The Java™ Tutorials); Copyright 1995-2006 Sun Microsystems, Inc.

MODULE 2 – USING OPERATING SYSTEMS

UNIT 1: Microsoft Operating System- Windows Xp

Content Page

1.0 INTRODUCTION

All computers run software called the *operating system*. The operating system allows the computer to run other software and to perform basic tasks, such as communicating between the computer and the user, keeping track of files and folders, and controlling peripheral devices. When a computer is turned on, the operating system software is automatically loaded into the computer's memory from the hard disk in a process called *booting*. Examples of operating systems include Windows XP.

In 2001 Microsoft released a new operating system known as Windows XP. This new version was widely praised in computer magazines. It shipped in two distinct editions, "Home" and "Professional", the former lacking many of the superior security and networking features of the Professional edition. The name "XP" is short for "eXPerience." Additionally, the first "Media Center" edition was released in 2002 with an emphasis on support for DVD and TV functionality including program recording and a remote control. Mainstream support for Windows XP ended on April 14, 2009.

2.0 OBJECTIVES

On completing this unit, you would be able to:

- Understand the Windows XP environment
- Work and Organize Files and Folders in XP
- Use applications easily in XP

3.0 WINDOWS XP ENVIRONMENT AND FEATURES

Windows XP has a graphical user interface or GUI (pronounced "gooey") that displays open applications and documents in areas on the screen called *windows*. Placing each open application into its own window allows Windows XP to multitask. *Multitasking* is an operating system feature that allows more than one application to run at a time. When the Windows XP operating system is running, the computer screen is referred to as the *Desktop*:

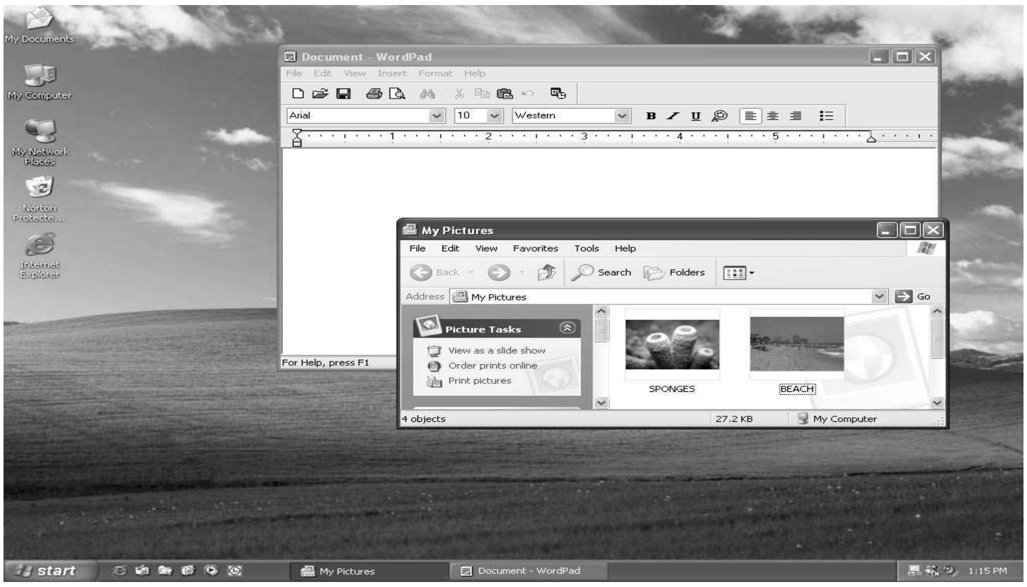
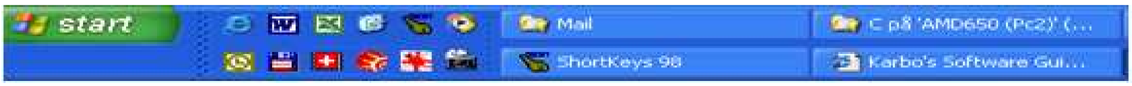


Figure 2.1(a): The Windows XP desktop environment

In the Figure 1.0, there are certain features we can observe;

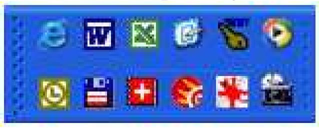
• **Icons** represent the applications and files available on the computer. Double-clicking an icon opens that file or application.

• The **Taskbar** displays buttons that represent each open file or application. In the example above, there are two open applications, My Pictures and WordPad. Clicking a button either displays or minimizes the window containing the file or application. The Taskbar also contains the start menu, Quick Launch toolbar, and the notification area.



• The start **menu** is used to run applications.

• The **Quick Launch toolbar** is used to start frequently used applications with just one click.



• The **notification area** contains a clock and other icons that display the status of certain activities such as printing.

Note: The Desktop's properties can be changed by right-clicking the Desktop and selecting Properties from the menu, which displays the Display Properties dialog box. This dialog box contains options that can be used to change the Desktop theme, background, and screen saver. Also, for **The Taskbar,** If too many buttons are displayed on the Taskbar, Windows XP will group files from the same application under a single button. Clicking that button on the Taskbar displays a menu where the user can select the appropriate file.

3.1 COMMANDS AND BUTTONS IN WINDOWS XP

Certain commands are required to carry out some task in Windows XP and certain button on the XP environment send commands to the operating system so that actions can be carried out on applications.

3.1.1 WINDOWS XP COMMANDS

Commands in XP can be found in the menu of applications. When menus are clicked on, a drop-down bar will emerge and these command options can be seen and chosen based on the task at hand. They include the following;

- Copy **command -**Copies the contents of an external storage device. Found in the menu displayed by right-clicking the content to be copied.
- Copy this file/folder **command -**Copies a selected file/folder to a new location. Found in the File and Folder Tasks section of the My Computer window.
- Delete this file/folder **command -**Moves a selected file/folder to the Recycle Bin. Found in the File and Folder Tasks section **of the My Computer window.**
- Format **command -**Prepares an external storage devices to receive data. Found in the menu displayed by right-clicking the icon of the device.
- Make a new folder **command -**Creates a new folder. Found in the File and Folder Tasks section of the My
  Computer window.
- Move this file/folder **command -**Moves a file/folder to a new location. Found in the File and Folder Tasks section of the My Computer window.
- Rename this file/folder **command -**Selects the existing file/folder name so that a new name can be typed. Found in the File and Folder Tasks section **of the My Computer window.**
- Restore **command -**Restores a file in the Recycle Bin to its previous location.
- Search for files/folders **command -**Used to find files and folders. Found in the System Tasks section of the My Computer window.

3.2 APPLICATIONS IN WINDOWS XP

Applications, also called software, applications software, or programs, are written by professional programmers to perform specific tasks. An application can manipulate text, numbers, graphics, or a combination of these elements. Some application packages offer considerable computing power by focusing on a single task, such as word processing; others, called integrated software, offer somewhat less power. A *Windows application* runs under the

Windows operating system.

In Windows XP, click start on the Taskbar to display a menu:

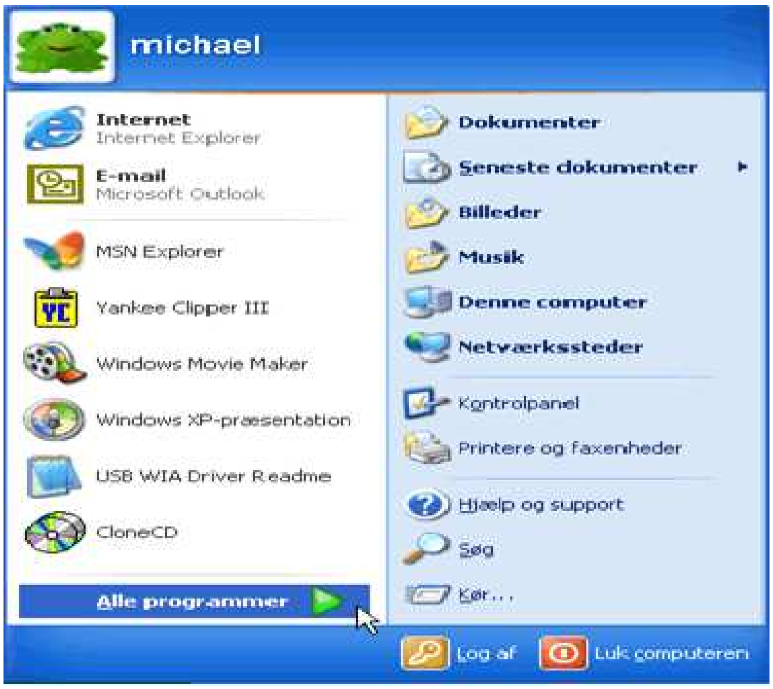


Figure 2.1(b): The Start menu opens when you press the Start button.

In the start menu, some items are commands that perform specific tasks, such as starting an application. For example, clicking Microsoft Word starts the Microsoft Word application. Clicking a menu item with an arrow, such as **All Programs** and **My Recent Documents** displays a submenu of commands. The appearance of the start menu may vary because it can be customized and because it displays the most commonly used applications. Most Windows applications have a similar interface. The *interface* of an application is the way it looks on the screen and the way in which a user provides input to the application.

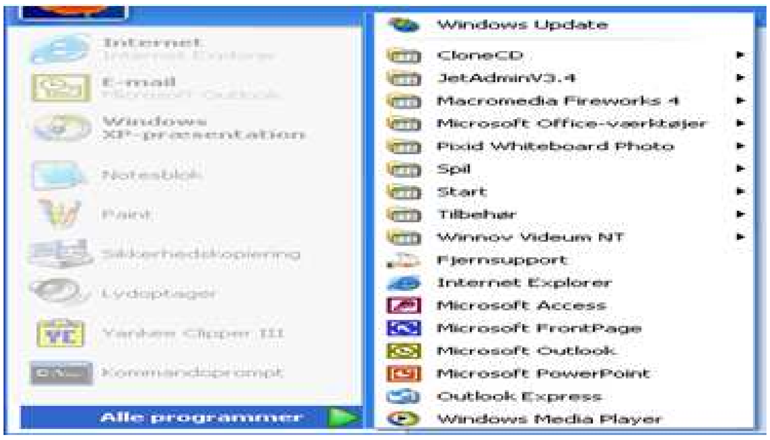


Figure 2.1(c): The list with All programs is seen to the right.

An example of Windows Application is the Paint Application for drawing;

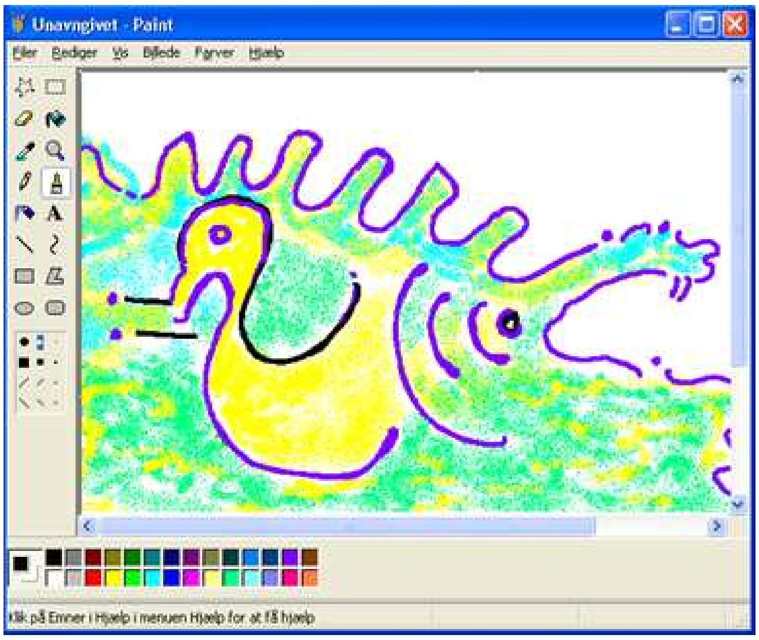


Figure 2.1(d): Paint is a small drawing program, which comes with Windows XP.

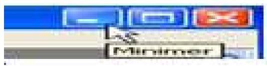
Let us begin:

1. Activate the Start button and open All Programs. Then open the program Paint, which is in the sub menu Accessories.

2. Now you can draw some by holding the left mouse button down and drag inside the white work area. Choose the Brush tool and draw a couple of curley cues.

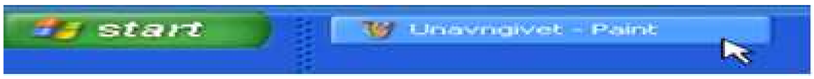
Now you have a program running: Paint is open. You can see that in a *window* on your screen.

1. Look in the upper right corner of the Paint window; there are three small buttons that control the window.

2. Click once on the left button 

3. Paint is *minimized*. This means that the program disappears from the screen.

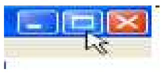
4. Notice the task bar in the bottom of the screen. Suddenly a button was added to the right of the Start button:



5. The button is named "unnamed - Paint". It tells that the drawing is not yet saved under a file name, and gives the name of the current program.

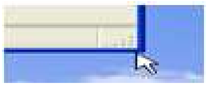
6. The window with Paint was thus *minimized*. Click on the button in the task bar. Then Paint appears again.

7. Now press on the middle of the three buttons in the upper right corner:

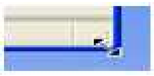


8. Then the program window changes from a reduced size to fill the whole screen (becomes *maximized*). Click on the same button a couple of times. Finish by having the "smaller" program window on the screen.

9. When a program window is not maximized, you can change its size. Try that. Place the mouse over the lower right corner of the window, where there us kind of a handle:



10. When you touch the corner, the mouse cursor changes to a double arrow. That is a signal that you can drag in the window frame, to increase or decrease the window size. Try that!



But you can drag in all corners and edges of the window. This window gymnastics was available in all versions of Windows; however many pc users still do not know and understand the system. That is a shame, since it can be very convenient to adjust the window size when you as an example work with more than one program open.

3.2.1 USING TOOLS BAR AND MENU BAR

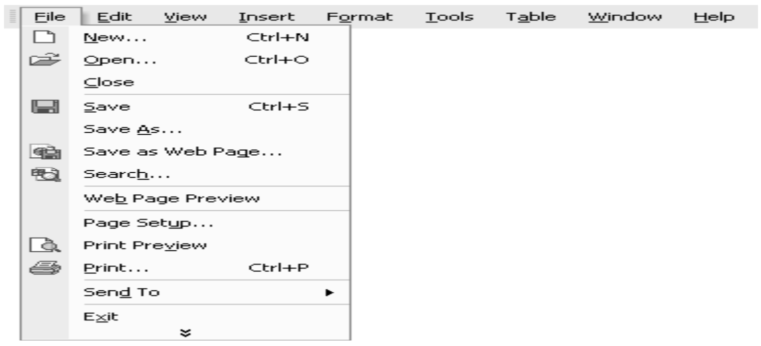
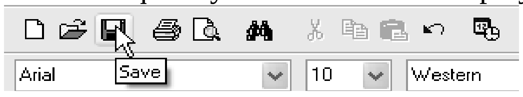


Figure 2.1(e): File Menu displays from the Menu Bar

An application window contains menus and toolbars. Each word on menu name can be clicked to display a menu of commands. Pointing to a command on the menu and then clicking selects the command. Clicking outside a displayed menu or pressing the Esc key removes the menu from the screen. The arrows ( ) at the bottom of a menu indicate that there are more commands available. Pointing to the arrow expands the menu to display more commands. The commands in an unexpanded menu are the most commonly used commands.

In the above figure, *keyboard accelerator* is a sequence of keys that select a command or menu. For example, the **New** command can be executed by pressing and holding the Ctrl key while pressing the N key, as shown in the menu, others are Ctrl + O (Open), Ctrl + S (Save) and Ctrl + P (Print).

Toolbars contain buttons that represent different actions, such as saving a document. Click a button to perform an action. Point to a button to display a *ScreenTip*, which is the action that the button will perform:
If a toolbar displays a button at the far right, clicking this button displays more buttons. Toolbars may vary in appearance since the most frequently used buttons are displayed.



3.2.2 THE DIALOGUE BOX

A dialog box is how a user communicates with an application. Dialog boxes can contain different elements. Windows XP gives (like the other Windows versions) the user great possibilities to make personal adjustments of screen colors, screen saver, printing, Saving Documents etc.

There is one central dialog box, which is used for certain adjustments, and it is called "*Display Properties*". You can activate it in different ways, such as selecting Control Panel --> Appearance and Themes. But there is another method, which I often use:

1. You need to minimize all program windows (try to use the keyboard shortcut Windows + d, then you have only the empty desktop in front of you.

2. Point with the mouse somewhere on the desktop, and make a *right click*.

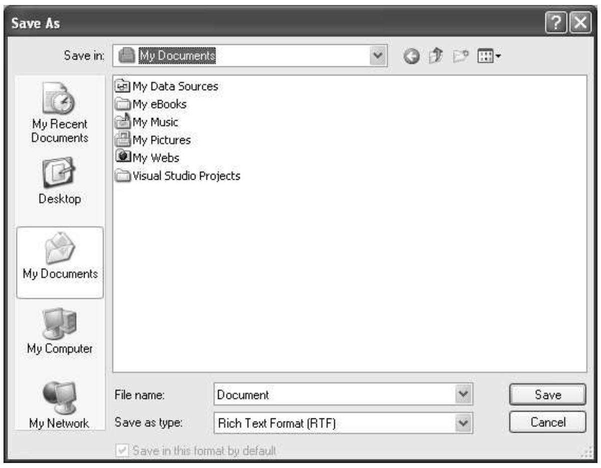
3. Then in the bottom of the menu select Properties:

4. That opens a dialog box with five tabs, where Themes is selected.

5. Try to quickly orient yourself about the possibilities by clicking on each tab.

6. Try also to move from tab to tab by pressing Control + Tab. See how the dialog box contents are changed for each new tab. You shift in "reverse" by pressing Control +Shift + Tab.

Another example of a dialogue box is the dialogue box that displays the save as properties;



Save in: My Documents

My Data Sources
My eBooks
My Music
My Pictures
My Webs
Visual Studio Projects

File name: Document

Save as type: Rich Text Format (RTF)

Save in this format by default

Figure 2.1(f): Save As dialog box

File name display a list of items to choose from to name your file. Click the arrow in the file name list to display items. Icons on the left side of the dialogue box perform actions as shortcuts to folders. In the dialog box above the shortcuts change a folder location. Once you are ready to save the file in the chosen folder, click the save button and the close the dialogue box to end that task.

3.2.3 MY DOCUMENT FOLDER

This part of Windows XP must be properly understood before we can start working with files and folders of different application.

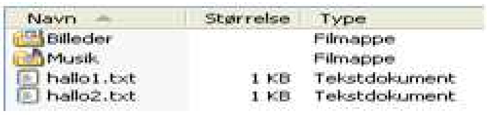
WHAT ARE DOCUMENTS?

When you work on your pc, you will create documents; that is really hard to avoid. A significant function in word processing, spread sheets and graphics programs is that your data are stored as **files**. Files, which are created with regular user program are called *documents*. The documents are saved on the pc's hard disk. But it is not insignificant where we place them – the documents must be available for retrieval and they need to be kept on back-up copies. Windows XP helps us with a built-in system folder called My Documents. Let us just open it again.

1. Open the folder My Documents from the start menu.

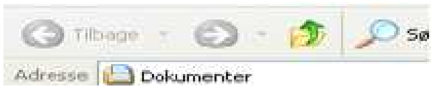
2. In the right window you see the *root level* of the folder My Documents. You might say that you are in the "top floor", since there can be many levels of sub folders in My Documents folder.

3. You now see at least two folders and two files. The files are shown with their full name (both file name and suffix).



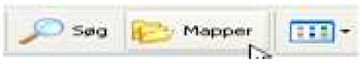
Navn	Størrelse	Type
Billeder		Filmappe
Musik		Filmappe
hallo1.txt	1 KB	Tekstdokument
hallo2.txt	1 KB	Tekstdokument

4. Notice the Address field, just below the tool bar. It reads My Documents:

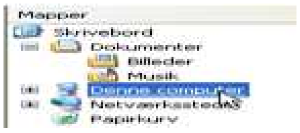


5. In this context address means placement. And since the folder My Documents is a system folder, the address is simply My Documents.

6. But the system folder My Documents also has a more "physical" place on the hard disk. Take a look at that. Click on the button Folders in the tool bar:



7. Find the icon My Computer in the left window and select it



8. It opens the sub division of drives and folders in the pc. Here you can again find the folder My Documents, which you find further down in the "tree".

Windows XP is basically designed for multiple users. Each user has their own edition of My Documents folder, just as he/she can have other personal settings for Windows XP. There are four users on my pc, therefore I have four copies of the folder My Documents; of course I identify my personal folder, and you should do that also:

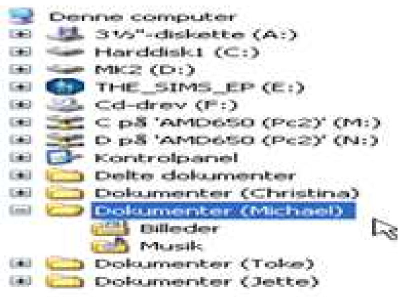


Figure 2.1 (g): Here one of the four personal folders is selected.

THE PERSONAL DOCUMENTS

In Figure 2.1(g), there are four users of the pc (Christina, Michael, Toke and Jette). You can see each folder in the tree structure under My Computer. When the folder My Documents is selected in this way, you can see its *path* in the address field. This address (path) is somewhat peculiar, so it requires an explanation:

The system folder My Documents is created by Windows XP and has automatically been assigned a not too user friendly placement on the hard disk. Here you see the path to my folder:

D:\Documents and Settings\michael.AMD900\Dokumenter

On your computer the path will look different, but the construction will still be similar to mine. Notice that each folder level is separated by a *backslash*.

In the bottom of the folder hierarchy (to the far right in the path) in the personal folder is the “physical” sub-folder My Documents:

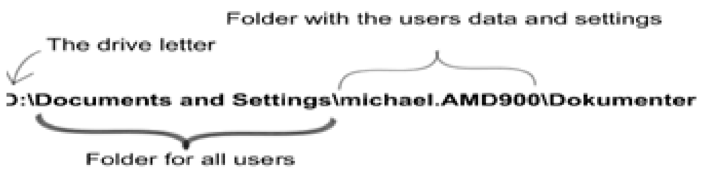

Folder with the users data and settings
The drive letter
D:\Documents and Settings\michael.AMD900\Dokumenter
Folder for all users

Figure 2.1(h): The construction of a path of folders and files in Windows XP

3.3 WORKING AND STORING FILES AND FOLDERS IN WINDOWS XP

A pc contains an incredible amount of data. When you work with user programs like Word, Excel and Photoshop, you create even more data. This whole mass of data must be maintained, and much of the work consists of *file management*.

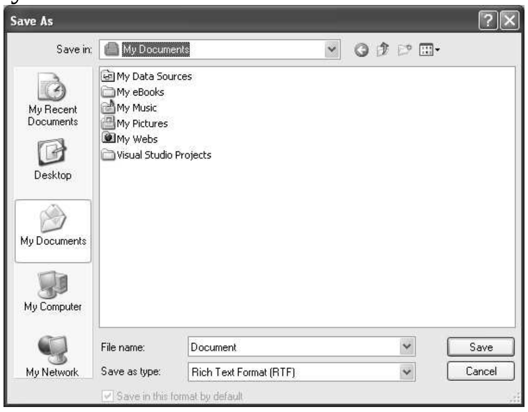
WHAT ARE DATA?

Data can be programs, documents or something else, but they are all saved in *files*. A *file* is a collection of related data stored on a lasting medium such as a hard disk, a CD, or a diskette. A *file name* is a unique name for a file stored on disk. All these files, of which there are thousands, are saved on the hard disk *drive* in multiple *folders*. The folders are structures that you create and maintain with the Explorer program.

Table 2.1(a): Data are saved in files, which are in folders on a drive.

Drives	Whole hard disks or parts of them (*partitions*), plus CD-ROMs etc. A drive typically contains many folders.
Folders	Containers for files that you can create yourself. A folder often contains many files.
Files	The smallest data units that we have access to. A file can be a whole document or a part of one.

The way users work with files to perform common tasks such as saving and printing is the same in most Windows applications. We will use the WordPad application to illustrate this concept. To save a project, select Save from the File menu or click the Save button on the toolbar. The Save As dialog box is displayed the first time a document is saved:



Save As

Save in: My Documents

My Data Sources
My eBooks
My Music
My Pictures
My Webs
Visual Studio Projects

File name: Document
Save as type: Rich Text Format (RTF)

Save in this format by default

Figure2.1(i): Save As dialog

The Save in list displays a location. The folders at that location are displayed in the contents box below the Save in list. Double-click a folder in the contents box to place that folder name in the

Save in list and display that folder's contents. When the correct folder is displayed in the Save in list, type a descriptive file name in the File name box and select Save. The shortcuts can also be used to change the folder location.

Apart from saving file, we can also use some of the commands from Section 4.1 to work on some file in certain application. When finished working on a document, it should be saved and then closed. *Closing a document* means that its window is removed from the screen and the file is no longer in the computer's memory. To close a document, select **Close** from the File menu or click the **Close** button in the upper-right corner of the document window. A warning dialog box is displayed if the document has been modified since it was last saved. *Opening a file* transfers a copy of the file contents to the computer's memory and then displays it in a window. To open a file, select **Open** from the File menu or click the **Open** button on the toolbar. The Open dialog box is similar to that of Save as but the title differs.

PRINTING

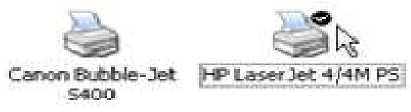
It is quite easy to print a document, but it requires that you have a printer connected to your pc, and that it is turned on and ready with paper. If you are using an application like Notepad;

1. Maximize Notepad.

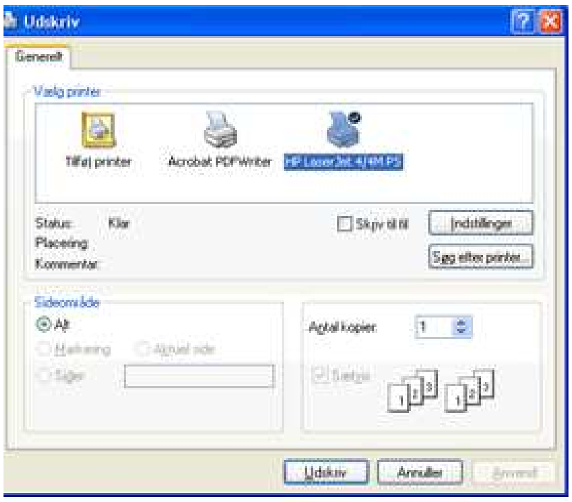
2. Then select menu items File --> Print…, or use the keyboard shortcut Ctrl + P; it works in all Windows programs.

3. You will probably get the Print dialog box.

4. If you have installed additonal printers, you can select between them in the same dialog box, since each printer appears with its own icon. One of the printers will always be the default printer. You can see that by the small checkmark in a black circle by the icon:



5. When you have selected a printer, just click OK, and the printing starts.



Status: Klar
Placering:
Kommentar:
☐ Skriv til fil

Sideområde
[✓ Alt]
Markering
Aktuel side
Sider

Antal kopier: 1
✓ Sortér

Figure 2.1(j): The Print dialog box is used to select the printer, number of copies, etc.

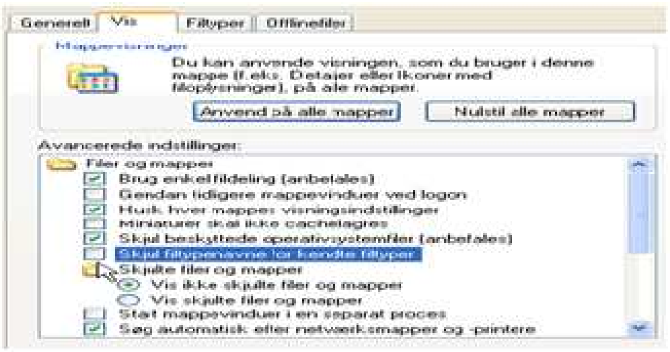
WINDOWS EXPLORER

Now let us look closer at the computer's folder and file contents. The file names are actually in two parts (a *first name* and a *suffix*), but ordinarily only the first names are shown. Now change the view, so the suffix (or rather the *file type name*) is shown in Windows Explorer.

1. Open Windows Explorer with the keyboard shortcut Windows + e. That opens Windows Explorer directly, where the system folder My Computer is selected in the folder window.

2. Select menu items Tools

3. Click on the View tab. In the Advanced settings you need to *remove* the checkmark by "Hide extensions for known file types" like here



Du kan anvende visningen, som du bruger i denne mappe (f.eks. Detaljer eller Ikoner med filoplysninger), på alle mapper.

Avancerede indstillinger:

Filer og mapper
- ✓ Brug enkel fildeling (anbefales)
- ☐ Gendan tidligere mappevinduer ved logon
- ✓ Husk hver mappes visningsindstillinger
- ☐ Miniaturer skal ikke cachelagres
- ✓ Skjul beskyttede operativsystemfiler (anbefales)
- ☐ Skjul filtypenavne for kendte filtyper
- Skjulte filer og mapper
  - [✓ Vis ikke skjulte filer og mapper]
  - Vis skjulte filer og mapper
- ☐ Start mappevinduer i en separat proces
- ✓ Søg automatisk efter netværksmapper og -printere

Figure 2.1(k): The display of the list of the View tab

4. Click on OK. Now the complete file names are shown for all files in all folders. Let us see an example. Click on the C-drive in the folder window:

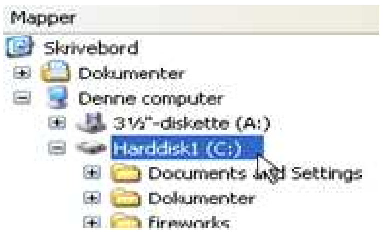


Figure 2.1(l): The C: folder

5. Then you see the contents of the C-drive's *root*. That is the "top" folder level on the hard disk. You might first get a warning not to make any changes. Then just click on "Show the contents of this folder":

6. Then click on the Windows folder in the left window. That is also a system folder (you might have to click again on Show the contents of this folder). It contains all the data that are in Windows XP.

There are countless numbers of file types, and you will never get to know them all. But – depending on which programs you work with – it is a good idea to know the file types that you produce and use. Here are some examples:

Table 2.1(b): Each file type has its own properties

Type	**Icon**	**Description**
htm, html		**HTML-documents**. Contain typically home pages. Can be viewed in a browser like Internet Explorer. Can be edited in text editors like Notepad.
Doc		**Word processing document,** which is handled in Microsoft Word.
Xls		**Spread sheet.** **Can be opened in programs like Excel.**

Type	Icon	Description
Bmp, png gif, jpg		**Graphics files.** **Can be opened with Windows Picture- and faxviewer, with Photoshop, Fireworks and many other graphics programs.**
mp3, wav wma		**Sound recordings.** **Can be played with among others Windows Media Player**
Avi, mpg wmv		**Movies (video recordings).** **Can be created with programs like Movie Maker and be played with Windows Media Player.**

3.4.1 SEARCHES ON THE HARD DISK

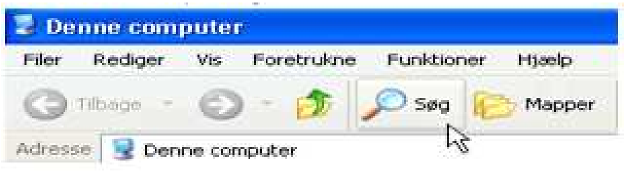
You can get the need to search for files. Maybe you have retrieved some files from the Internet, and now you cannot find them, or maybe some document got lost on the disk. Then you just need to use the search function.

THE SEARCH FUNCTION

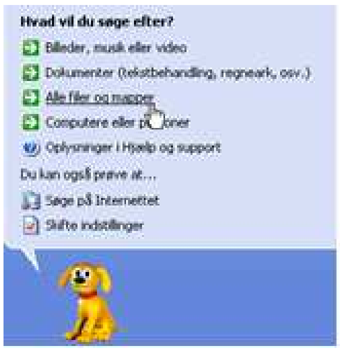
You can find the Search dialog box in different ways. You find the menu item Search in the Start menu. But in this exercise we will do it in a different way:

1. Open Windows Explorer and select My Computer.

2. Now you can either press F3.



3. Then the Search panel opens in Explorer's left window. There is a box with different options. Click on "All files and folders" like here:


Hvad vil du søge efter?

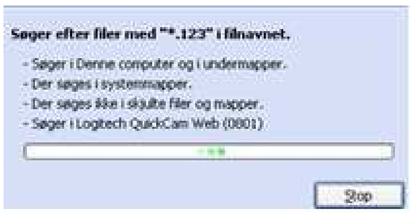
- Billeder, musik eller video
- Dokumenter (tekstbehandling, regneark, osv.)
- Alle filer og mapper
- Computere eller personer
- Oplysninger i Hjælp og support

Du kan også prøve at...

- Søge på Internettet
- Skifte indstillinger

4. Then you see a new box, where you need to enter search criteria. Then you are searching for all file with any name. Then click on Search.

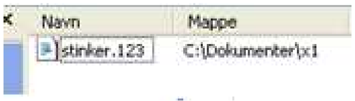
5. Then the search starts:


Søger efter filer med "*.123" i filnavnet.

- Søger i Denne computer og i undermapper.
- Der søges i systemmapper.
- Der søges ikke i skjulte filer og mapper.
- Søger i Logitech QuickCam Web (0801)

Stop

6. The file is soon found. It appears in the right Explorer window:



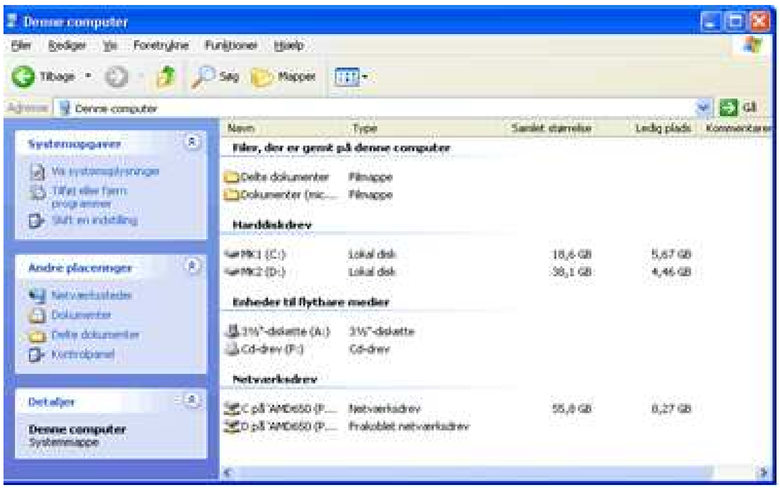
Navn	Mappe
stinker.123	C:\Dokumenter\x1

You can search files and folders in numerous ways. The most important are probably:

- Search by first name or a part thereof, or search by the file’s suffix (as in the preceding exercise).
- Search by a word or a text string, which is to be found in the file. That can be used to search for many types of documents that contain text, such as Word-files, txt-files and HTML-documents.
- Search by date or time when the file was last saved.

3.4 WINDOWS EXPLORER (My Computer)

Windows Explorer is really your tool to work with files, folders, programs, hyperlinks, Control panel and much more. The starting point for work with Windows Explorer is often *My Computer*. That is a system folder that shows and gives access to most of the pc’s resources. You have worked a lot with Windows Explorer in the previous sections. Windows Explorer is an application that comes with Windows XP that can be used to do many of the same file and folder management tasks that can be performed in My Computer such as copying, deleting and renaming files.



Navn	Type	Samlet størrelse	Ledig plads	Kommentarer
Filer, der er gemt på denne computer				
Delte dokumenter	Filmappe			
Dokumenter (mic...	Filmappe			
Harddiskdrev				
MK1 (C:)	Lokal disk	18,6 GB	5,67 GB	
MK2 (D:)	Lokal disk	38,1 GB	4,46 GB	
Enheder til flytbare medier				
3½"-diskette (A:)	3½"-diskette			
Cd-drev (F:)	Cd-drev			
Netværksdrev				
C på 'AMD650 (P...	Netværksdrev	55,8 GB	8,27 GB	
D på 'AMD650 (P...	Frakoblet netværksdrev			

Figure 2.1(m): Windows Explorer environment

3.5 CONTROL PANEL

The control panel is the entry to the pc’s various *settings* such as controlling hardware units through the so-called *drivers.*

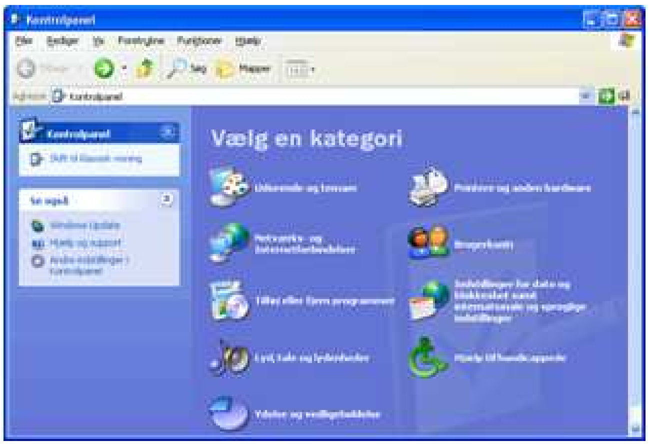

Kontrolpanel
Filer Rediger Vis Foretrukne Funktioner Hjælp
Tilbage Søg Mapper
Adresse Kontrolpanel Gå
Kontrolpanel
Skift til klassisk visning
Se også
Windows Update
Hjælp og support
Andre indstillinger i Kontrolpanel
Vælg en kategori
Udseende og temaer
Printere og anden hardware
Netværks- og internetforbindelser
Brugerkonti
Tilføj eller fjern programmer
Indstillinger for dato og klokkeslæt samt internationale og sproglige indstillinger
Lyd, tale og lydenheder
Hjælp til handicappede
Ydelse og vedligeholdelse

Figure 2.1(n): The control panel in Windows XP.

From Figure 2.1(n), settings such as Audio, network, computers graphics display and themes, printer setting, date and time, installations of programs and more can be done at the control panel.

3.0 CONCLUSION

Windows XP has better functionality as it support multitasking. Also has been built with an impressive graphics user interface. It is much more user friendly and interactive.

5.0 SUMMARY

All computers run software called the operating system. The operating system allows the computer to run other software and to perform basic tasks, such as communicating between the computer and the user, keeping track of files and folders, and controlling peripheral devices.

When a computer is turned on, the operating system software is automatically loaded into the computer's memory from the hard disk in a process called booting. When the Windows XP operating system is running, the computer screen is referred to as the Desktop. Multitasking is an operating system feature that allows more than one application to run at a time. Windows XP handles multitasking by placing each running application into its own window.

Applications, also called software, applications software, or programs, are written by professional programmers to perform specific tasks. A Windows application runs under the Windows operating system. The interface of an application is the way it looks on the screen and the way in which a user provides input to the application. An application window contains menus and toolbars. Each word on the menu bar can be clicked to display a menu of commands. A keyboard accelerator allows a command or menu to be selected by pressing a sequence of keys. Pointing to a button displays a ScreenTip, which describes the action that button will perform.

A dialog box is how a user communicates with an application. Dialog boxes can contain different elements including buttons, text boxes, check boxes, radio buttons, and lists. A file is a collection of related data stored on a lasting medium such as a hard disk, a CD, or a diskette. A file name is a unique name for a file stored on disk. A file extension indicates what application the file was created in. My Computer is an application that comes with Windows XP and is used to view the contents of the hard disk drive and of the computer's removable storage devices. Folders are used to organize commonly related files. In the My Computer window, selecting a file or folder expands the list of commands in the File and Folder Tasks section to include commands such as Rename this folder, Copy this file and Delete this file. The My Computer window can be used to find files and folders by selecting Search for files or folders in the System Tasks section. The control panel windows give access to user to reset or adjust settings of the computer such as its network settings, desktop themes and more.

6.0 TUTOR MARKED ASSIGNMENT

1 a) What is an operating system?
  b) What is booting?
  c) What is the Desktop?
2 Define the following terms; i) icons ii) taskbar iii) notification area and iv) quick launch bar.
3 a) What is multitasking?
  b) What is another name for applications?
  c) What is a dialog box used for?
  d) Which element in a dialog box enables the user to choose from a set of choices?
4 Describe how personal documents are stored in Windows XP.

5 a) What is My Computer?
   b) State at least 5 major features that can be seen in My Computer and discuss?
   c) Is it possible to search for a file if only a word or phrase in the file name is known?
6 Describe the steps to printing a document.
7 What is the control panel?

7.0 FURTHER READING

1. Joyce, Jerry; Moon, Marianne (2004). *Microsoft Windows XP Plain & Simple*. Microsoft Press. ISBN 9780735621121.

2. Microsoft PressPass. Microsoft. 5 February 2001. Retrieved 2009-09-17. "The XP name is short for "experience". http://www.microsoft.com/presspass/press/2001/feb01/02-05namingpr.mspx.

3. Windows XP - Beginners Tutorial; http://www.karbosguide.com/books/winxpbeginner.

4. Windows XP - Wikipedia, the free encyclopedia; http://en.wikipedia.org/wiki/Windows_XP.

5. An Introduction to Programming Using Microsoft Visual Basic .NET; Introducing Windows XP- http://www.lpdatafiles.com/opsys/xp.pdf

MODULE 2 – USING OPERATING SYSTEMS

UNIT 2: MICROSOFT OPERATING SYSTEM- Windows 7

1.0 INTRODUCTION

Windows 7 is an *operating system*. It controls the hardware of your computer and interprets the instructions from your software and operating systems to the hardware. When you use a menu command such as **File , Save**, Windows 7 is the driving force that writes the file to your disk or hard drive.

Windows 7 Versions: There are several different versions of Windows 7 including Windows 7 Professional, Windows 7 HomeBasic, Windows 7 Premium, and Windows 7 Starter and Windows 7 Ultimate. For the purposes of this class, Windows 7 Ultimate would be referred to as Windows 7.

2.0 OBJECTIVES

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

(a) To start and shut down Windows, use a mouse to manipulate items on the desktop.

(b) To open, resize, move, view, and close a window, and to use scroll bars to view more of a window.

(c) To use toolbar buttons, open menus, choose menu commands, and use menu shortcuts.

(d) To work on a dialogue box.

3.0 WINDOWS 7 ENVIRONMENT AND FEATURES

Windows 7 has a graphical user interface or GUI (pronounced “gooey”) that displays open applications and documents in areas on the screen called *windows*. Placing each open application into its own window allows Windows 7 to multitask. *Multitasking* is an operating system feature that allows more than one application to run at a time. When the Windows 7 operating system is running, the computer screen is referred to as the *Desktop*. Its desktop environment is attractive.

The desktop is the main screen area that you see after you turn on your computer and log on to Windows. Like the top of an actual desk, it serves as a surface for your work. When you open programs or folders, they appear on the desktop. You can also put things on the desktop, such as files and folders, and arrange them however you want.

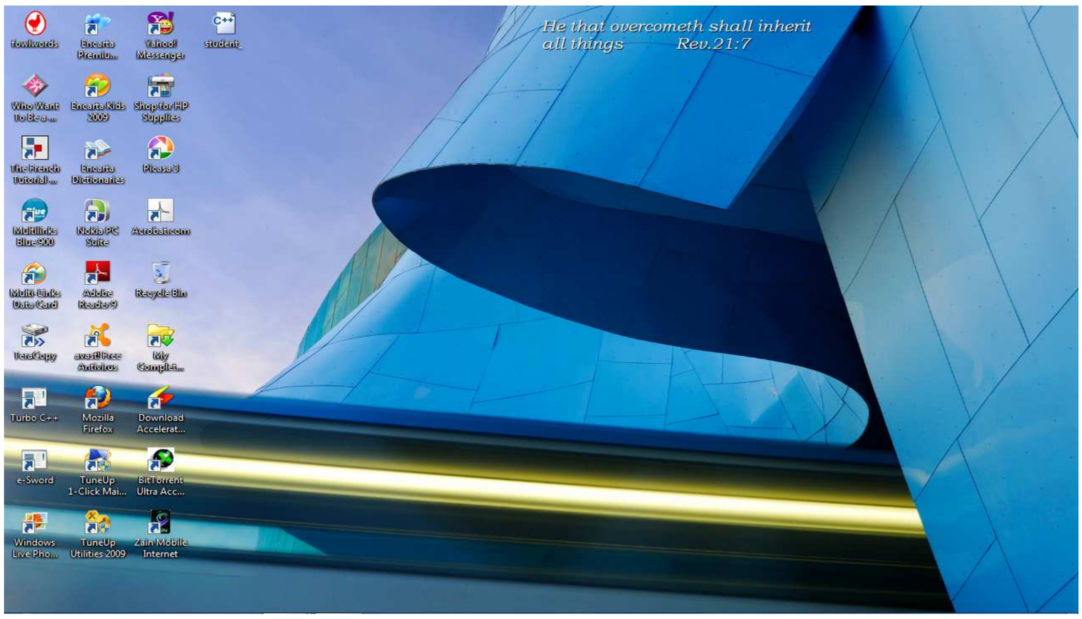


Figure 2.2(a): The Windows 7 desktop environment

The desktop is sometimes defined more broadly to include the taskbar. The taskbar sits at the bottom of your screen. It shows you which programs are running and allows you to switch between them. It also contains the Start button , which you can use to access programs, folders, and computer settings.

Some Features on the desktop environment include;

• **Icons** represent the applications and files available on the computer. Icons are small pictures that represent files, folders, programs, and other items. When you first start Windows, you'll see at least one icon on your desktop: The Recycle Bin Double-clicking an icon opens that file or application.

• The **Taskbar** displays buttons that represent each open file or application. In the example above, there are two open applications, My Pictures and WordPad. Clicking a button either displays or minimizes the window containing the file or application. The Taskbar also contains the start menu, Quick Launch toolbar, and the notification area.

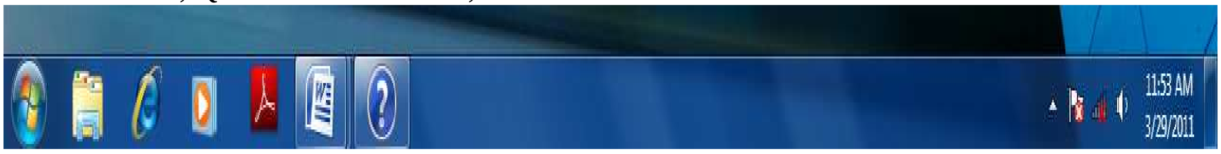


Figure 2.2(b): The Windows 7 Taskbar

• The start **menu** is used to run applications.

• The **Quick Launch toolbar** is used to start frequently used applications with just one click.

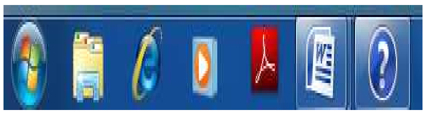


Figure 2.2(c) : The Windows 7 Quick Launch toolbar

• The **notification area** contains a clock and other icons that display the status of certain activities such as printing.

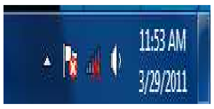


Figure 2.2(d) : The Windows 7 notification area

The Desktop's properties can be changed by right-clicking the Desktop and selecting Properties from the menu, which displays the Display Properties dialog box. This dialog box contains options that can be used to change the Desktop theme, background, and screen saver.

3.1 USING MENUS, BUTTONS, BARS, AND BOXES

Menus, buttons, scroll bars, and check boxes are examples of controls that you operate with your mouse or keyboard. These controls allow you to select commands, change settings, or work with windows. This section describes how to recognize and use controls that you'll encounter frequently while using Windows.

3.1.1 Using menus

Most programs contain dozens or even hundreds of commands (actions) that you use to work the program. Many of these commands are organized under menus. Like a restaurant menu, a program menu shows you a list of choices. To keep the screen uncluttered, menus are hidden until you click their titles in the menu bar, located just underneath the title bar.

To choose one of the commands listed in a menu, click it. Sometimes a dialog box appears, in which you can select further options. If a command is unavailable and cannot be clicked, it is shown in gray. Some menu items are not commands at all. Instead, they open other menus.

If you don't see the command you want, try looking at another menu. Move your mouse pointer along the menu bar and its menus open automatically; you don't need to click the menu bar again. To close a menu without selecting any commands, click the menu bar or any other part of the window.

Tips

- If a keyboard shortcut is available for a command, it is shown next to the command.
- You can operate menus using your keyboard instead of your mouse. See Using your keyboard.

3.1.2 Using scroll bars

When a document, webpage, or picture exceeds the size of its window, scroll bars appear to allow you to see the information that is currently out of view. The following picture shows the parts of a scroll bar.

To use a scroll bar:

- Click the up or down scroll arrows to scroll the window's contents up or down in small steps. Hold down the mouse button to scroll continuously.
- Click an empty area of a scroll bar above or below the scroll box to scroll up or down one page.
- Drag a scroll box up, down, left, or right to scroll the window in that direction.

Tip

- If your mouse has a scroll wheel, you can use it to scroll through documents and webpages. To scroll down, roll the wheel backward (toward you). To scroll up, roll the wheel forward (away from you).

3.1.3 Using command buttons

A command button performs a command (makes something happen) when you click it. You'll most often see them in dialog boxes, which are small windows that contain options for completing a task

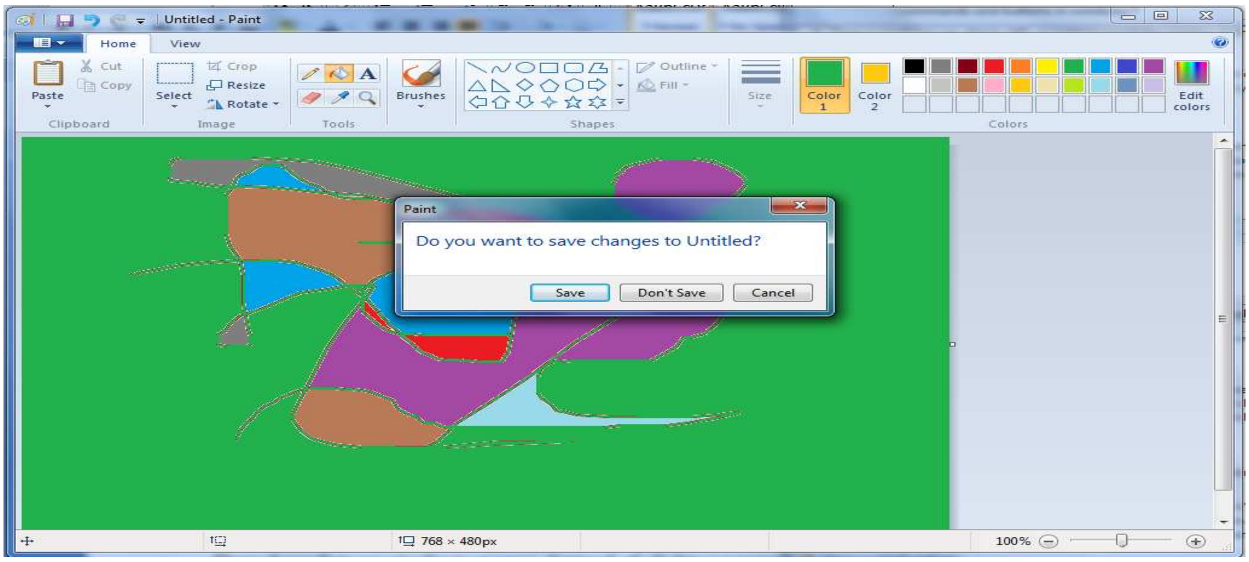

Do you want to save changes to Untitled?

Figure 2.2(e): Paint environment showing the command buttons

To close the picture, you must first click either the Save or Don't Save button. Clicking Save saves the picture and any changes you've made, and clicking Don't Save deletes the picture and discards any changes you've made. Clicking Cancel dismisses the dialog box and returns you to the program.

Tip

- Pressing Enter does the same thing as clicking a command button that is selected (outlined).

Outside of dialog boxes, command buttons vary in appearance, so it's sometimes difficult to know what's a button and what isn't. For example, command buttons often appear as small icons (pictures) without any text or rectangular frame. The most reliable way to determine if something is a command button is to rest your pointer on it. If it "lights up" and becomes framed with a rectangle, you've discovered a button. Most buttons will also display some text about their function when you point to them.

If a button changes into two parts when you point to it, you've discovered a split button. Clicking the main part of the button performs a command, whereas clicking the arrow opens a menu with more options.

3.1.4 Using option buttons and check boxes

Option buttons allow you to make one choice among two or more options. They frequently appear in dialog boxes. sCheck boxes allow you to select one or more independent options. Unlike option buttons, which restrict you to one choice, check box allow you to choose multiple options at the same time.

To use check boxes:

- Click an empty square to select or "turn on" that option. A check mark will appear in the square, indicating that the option is selected.
- To turn off an option, clear (remove) its check mark by clicking it.
- Options that currently can't be selected or cleared are shown in gray.

3.1.5 Using sliders and text boxes

A slider lets you adjust a setting along a range of values. A slider along the bar shows the currently selected value. To use a slider, drag the slider toward the value that you want. A text box allows you to type information, such as a search term or password. The following picture shows a dialog box containing a text box. We've entered "bear" into the text box.

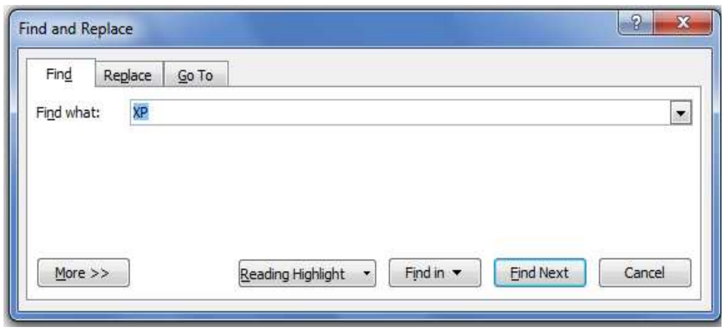


Figure2.2(e): Text box

A blinking vertical line called the cursor indicates where text that you type will appear. If you don't see a cursor in the text box, it means the text box isn't ready for your input. Click the box first, and then start typing. Text boxes that require you to enter a password will usually hide your password as you type it, in case someone else is looking at your screen.

3.1.6 Using drop-down lists

Drop-down lists are similar to menus. Instead of clicking a command, though, you choose an option. When closed, a drop-down list shows only the currently selected option. The other available options are hidden until you click the control, as shown below.

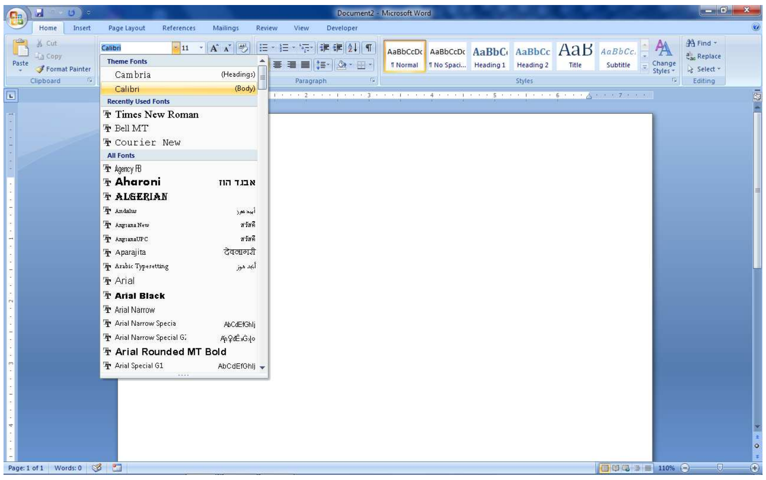


Figure 2.2(f): The drop-down list of fonts in windows

To open a drop-down list, click it. To choose an option from the list, click the option.

3.2 PROGRAMS/ APPLICATIONS ON WINDOWS 7

Almost everything you do on your computer requires using a program. For example, if you want to draw a picture, you need to use a drawing or painting program. To write a letter, you use a word processing program. To explore the Internet, you use a program called a web browser. Thousands of programs are available for Windows.

3.2.1 Opening a program

The Start menu is the gateway to all of the programs on your computer. To open the Start menu, click the Start button. The left pane of the Start menu contains a small list of programs, including your Internet browser, e-mail program, and recently used programs. To open a program, click it. If you don't see the program you want to open, but you know its name, type all or part of the name into the search box at the bottom of the left pane. Under Programs, click a program to open it. To browse a complete list of your programs, click the Start button, and then click All Programs

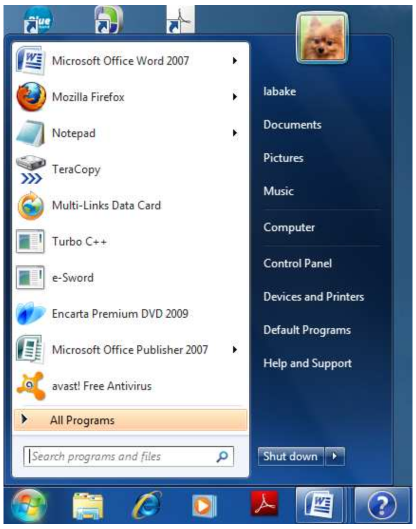


Figure 2.2(g): Start menu

Tip

- You can also open a program by opening a file. Opening the file automatically opens the program associated with the file. For more information, see Open a file or folder.

3.2.2 Using commands in programs

Most programs contain dozens or even hundreds of commands (actions) that you use to work the program. Many of these commands are organized in a Ribbon, located just under the title bar.

In some programs, commands might be located under menus. Like a restaurant menu, a program menu shows you a list of choices. To keep the screen uncluttered, menus are hidden until you click their titles in the menu bar, located under the title bar. Sometimes a dialog box will appear, in which you can select further options. If a command is unavailable and cannot be clicked, it is shown in gray.

3.2.3 Creating a new document

Many programs allow you to create, edit, save, and print documents. In general, a document is any type of file that you can edit. For example, a word processing file is a type of document, as is a spreadsheet, an e-mail message, and a presentation. However, the terms document and file are often used interchangeably; pictures, music clips, and videos that you can edit are usually called files, even though they are technically documents.

Some programs, including WordPad, Notepad, and Paint, open a blank, untitled document automatically when you open the program, so that you can start working right away. You'll see a large white area and a generic word like "Untitled" or "Document" in the program's title bar.

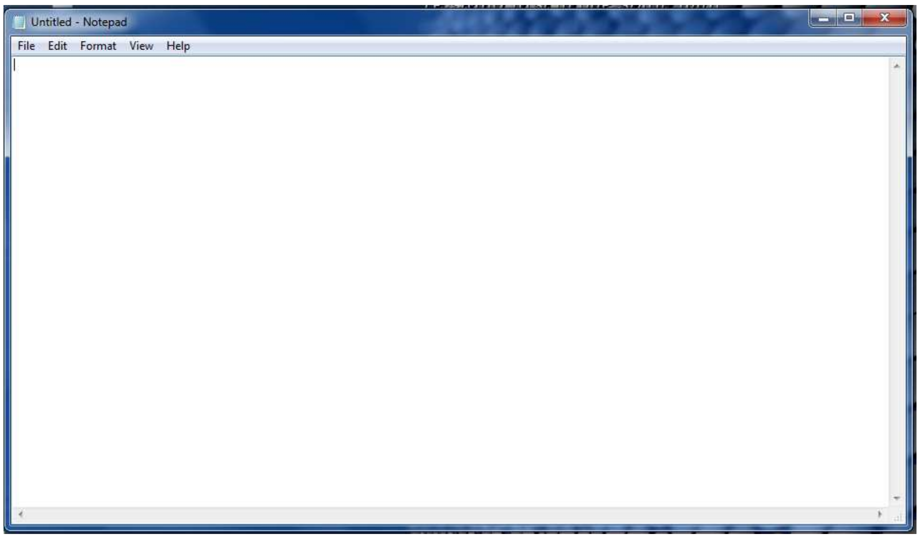


Figure 2.2(h): The title bar in NotePad

If your program doesn't open a new document automatically when it opens, you can do it yourself:

- Click the File menu in the program you are using, and then click New or Click the menu button, and then click New. If you can open more than one type of document in the program, you might also need to select the type from a list.

3.2.4 Saving a document

Saving a document allows you to name it and to store it permanently on your computer's hard disk. That way, the document is preserved even when your computer is turned off, and you can open it again later.

To save a document

1. Click the File menu, and click Save or Click the Save button .

2. If this is the first time you are saving the document, you'll be asked to provide a name for it and a location on your computer to save it to.

Even if you've saved a document once, you need to keep saving it as you work. That's because any changes you've made since you last saved the document are stored in RAM, not on the hard disk. To avoid losing work unexpectedly due to a power failure or other problem, save your document every few minutes.

3.2.5 Moving information between files

Most programs allow you to share text and images between them. When you copy information, it goes into a temporary storage area called the Clipboard. From there, you can paste it into a document.

Undoing your last action

Most programs allow you to undo (reverse) actions you take or mistakes you make. For example, if you delete a paragraph in a WordPad document accidentally, you can get it back by using the Undo command. If you draw a line in Paint that you don't want, undo your line right away and it vanishes or better still use ctrl + Z command.

To undo an action

- Click the Edit menu, and click Undo or Click the Undo button .

3.2.6 Exiting a program

To exit a program, click the Close button in the upper-right corner of the program window. Or, click the File menu, and click Exit. Remember to save your document before exiting a program. If you have unsaved work and try to exit the program, the program will ask you whether you want to save the document.

3.2.7 Installing or uninstalling programs

You're not limited to using the programs that came with your computer—you can buy new programs on CD or DVD or download programs (either free or for a fee) from the Internet. Installing a program means adding it to your computer. After a program is installed, it appears in your Start menu in the All Programs list. Some programs might also add a shortcut to your desktop. For more information, see Install a program.

3.3 WORKING WITH FILES AND FOLDERS IN WINDOWS 7

A file is an item that contains information—for example, text or images or music. When opened, a file can look very much like a text document or a picture that you might find on someone's desk or in a filing cabinet. On your computer, files are represented with icons; this makes it easy to recognize a type of file by looking at its icon.

A folder is a container you can use to store files in. If you had thousands of paper files on your desk, it would be nearly impossible to find any particular file when you needed it. That's why people often store paper files in folders inside a filing cabinet. On your computer, folders work the same way. Folders can also store other folders. A folder within a folder is usually called a subfolder. You can create any number of subfolders, and each can hold any number of files and additional subfolders.

3.3.1 Using libraries to access your files and folders

When it comes to getting organized, you don't need to start from scratch. You can use libraries, a feature new to this version of Windows, to access your files and folders, and arrange them in different ways. Here's a list of the four default libraries and what they're typically used for:

- **Documents library**. Use this library to organize and arrange word-processing documents, spreadsheets, presentations, and other text-related files. By default, files that you move, copy, or save to the Documents library are stored in the My Documents folder.
- **Pictures library**. Use this library to organize and arrange your digital pictures, whether you get them from your camera, scanner, or in e-mail from other people. By default, files that you move, copy, or save to the Pictures library are stored in the My Pictures folder.
- **Music library**. Use this library to organize and arrange your digital music, such as songs that you rip from an audio CD or that you download from the Internet. By default, files that you move, copy, or save to the Music library are stored in the My Music folder.
- **Videos library**. Use this library to organize and arrange your videos, such as clips from your digital camera or camcorder, or video files that you download from the Internet. By default, files that you move, copy, or save to the Videos library are stored in the My Videos folder.

To open the Documents, Pictures, or Music libraries, click the Start button, and then click Documents, Pictures, or Music. You can open common libraries from the Start menu

3.3.2 Understanding the parts of a window

When you open a folder or library, you see it in a window. The various parts of this window are designed to help you navigate around Windows or work with files, folders, and libraries more easily. Here's a typical window and each of its parts:

Table 2.2(a): Parts of Windows

Window part	**What it's useful for**
Navigation pane	Use the navigation pane to access libraries, folders, saved searches, and even entire hard disks. Use the Favorites section to open your most commonly used folders and searches; use the Libraries section to access your libraries. You can also use the Computer folder to browse folders and subfolders. For more information, see Working with the navigation pane.
Back and Forward buttons	Use the Back button and the Forward button to navigate to other folders or libraries you've already opened without closing the current window. These buttons work together with the address bar; after you use the address bar to change folders, for example, you can use the Back button to return to the previous folder.
Toolbar	Use the toolbar to perform common tasks, such as changing the appearance of your files and folders, burning files to a CD, or starting a digital picture slide show. The toolbar's buttons change to show only the tasks that are relevant. For example, if you click a picture file, the toolbar shows different buttons than it would if you clicked a music file.
Address bar	Use the address bar to navigate to a different folder or library or to go back to a previous one. For more information, see Navigate using the address bar.
Library pane	The library pane appears only when you are in a library (such as the Documents library). Use the library pane to customize the library or to arrange the files by different properties. For more information, see Working with libraries.
Column headings	Use the column headings to change how the files in the file list are organized. For example, you can click the left side of a column heading to change the order the files and folders are displayed in, or you can click the right side to filter the files in different ways. (Note that column headings are available only in Details view. To learn how to switch to Details view, see 'Viewing and arranging files and folders' later in this topic.)
File list	This is where the contents of the current folder or library are displayed. If you type in the search box to find a file, only the files that match your current view (including files in subfolders) will appear.
The search box	Type a word or phrase in the search box to look for an item in the current folder or library. The search begins as soon as you begin typing—so if you type "B," for example, all the files with names starting with the letter B will appear in the file list. For more information, see Find a file or folder.
Details pane	Use the details pane to see the most common properties associated with the selected file. File properties are information about a file, such as the author, the date you last changed the file, and any descriptive tags you might have added to the file. For more information, see Add tags and other properties to files.
Preview pane	Use the preview pane to see the contents of most files. If you select an e-mail message, text file, or picture, for example, you can see its contents without opening it in a program. If you don't see the preview pane, click the Preview pane

Window part	What it's useful for
	button in the toolbar to turn it on.

3.3.3 Viewing and arranging files and folders

When you open a folder or library, you can change how the files look in the window. For example, you might prefer larger (or smaller) icons or a view that lets you see different kinds of information about each file. To make these kinds of changes, use the Views button in the toolbar.

Each time you click the left side of the Views button, it changes the way your files and folders are displayed by cycling through five different views: Large Icons, List, a view called Details that shows several columns of information about the file, a smaller icon view called Tiles, and a view called Content that shows some of the content from within the file.

If you click the arrow on the right side of the Views button, you have more choices. Move the slider up or down to fine-tune the size of the file and folder icons. You can see the icons change size as you move the slider.

In libraries, you can go a step further by arranging your files in different ways. For example, say you want to arrange the files in your Music library by genre (such as Jazz and Classical):

1. Click the Start button, and then click Music.
2. In the library pane (above the file list), click the menu next to Arrange by, and then click Genre.

Finding files

Depending on how many files you have and how they are organized, finding a file might mean browsing through hundreds of files and subfolders—not an easy task. To save time and effort, use the search box to find your file

The search box is located at the top of every window. To find a file, open the folder or library that makes the most sense as a starting point for your search, click the search box, and start typing. The search box filters the current view based on the text that you type. Files are displayed as search results if your search term matches the file's name, tags or other properties, or even the text inside a text document.

If you're searching for a file based on a property (such as the file's type), you can narrow the search before you start typing by clicking the search box, and then clicking one of the properties just below the search box. This adds a search filter (such as "type") to your search text, which will give you more accurate results.

If you aren't seeing the file you're looking for, you can change the entire scope of a search by clicking one of the options at the bottom of the search results. For example, if you search for a file in the Documents library but you can't find it, you can click Libraries to expand the search to the rest of your libraries. For more information, see Find a file or folder.

Copying and moving files and folders

Occasionally, you might want to change where files are stored on your computer. You might want to move files to a different folder, for example, or copy them to removable media (such as CDs or memory cards) to share with another person.

Most people copy and move files using a method called drag and drop. Start by opening the folder that contains the file or folder you want to move. Then, open the folder where you want to move it to in a different window. Position the windows side by side on the desktop so that you can see the contents of both.

When using the drag-and-drop method, you might notice that sometimes the file or folder is copied, and at other times it's moved. If you're dragging an item between two folders that are stored on the same hard disk, then the item is moved so that two copies of the same file or folder aren't created in the same location. If you drag the item to a folder that is in a different location (such as a network location) or to removable media like a CD, the item is copied.

Tips

- The easiest way to arrange two windows on the desktop is to use Snap. For more information, see Arrange windows side by side on the desktop using Snap.
- If you copy or move a file or folder to a library, it will be stored in the library's default save location. To learn how to customize a library's default save location, see Customize a library.
- Another way to copy or move a file is to drag it from the file list to a folder or library in the navigation pane so you don't need to open two separate windows.

Creating and deleting files

The most common way to create new files is by using a program. For example, you can create a text document in a word-processing program or a movie file in a video-editing program.

Some programs create a file as soon as you open them. When you open WordPad, for example, it starts with a blank page. This represents an empty (and unsaved) file. Start typing, and when you are ready to save your work, click the Save button. In the dialog box that appears, type a file name that will help you find the file again in the future, and then click Save.

By default, most programs save files in common folders like My Documents and My Pictures, which makes it easy to find the files again next time.

When you no longer need a file, you can remove it from your computer to save space and to keep your computer from getting cluttered with unwanted files. To delete a file, open the folder or library that contains the file, and then select the file. Press Delete on your keyboard and then, in the Delete File dialog box, click Yes.

When you delete a file, it's temporarily stored in the Recycle Bin. Think of the Recycle Bin as a safety net that allows you to recover files or folders that you might have accidentally deleted. Occasionally, you should empty the Recycle Bin to reclaim all of the hard disk space being used by your unwanted files.

Opening an existing file

To open a file, double-click it. The file will usually open in the program that you used to create or change it. For example, a text file will open in your word-processing program. That is not always the case, though. Double-clicking a picture file, for example, will usually open a picture viewer. To change the picture, you need to use a different program. Right-click the file, click open with, and then click the name of the program that you want to use.

Print a document or file

The quickest way to print a document or file is to print using Windows. You don't have to open any programs or change any settings.

- Right-click the file you want to print, and then click Print.

  Windows will launch the program that created the file and send it to your default printer.

Windows Photo Viewer can make prints of your digital photos. If you don't have a printer handy, you can use Windows to order prints online.

To print your own pictures

Remember: print quality depends on several factors, including your choices of paper and ink, printer settings, and the quality of the original photo.

1. Double-click a picture to launch Windows Photo Viewer.
2. On the toolbar, click the Print button, and then click Print.
3. In the Print Pictures dialog box, select the printer, paper size, print quality, print style, and number of copies.
4. When you're done, click Print.

Notes

- o To launch the Print Pictures dialog box directly, right-click a photo and then click Print.
- o Some printers allow you to print a mirror image of a picture (sometimes called "reverse landscape" or "reverse portrait"). If this option is selected, the preview image in the Print Pictures dialog box won't change. However, the picture will print as a mirror image.

3.4 CONTROL PANEL IN WINDOWS 7

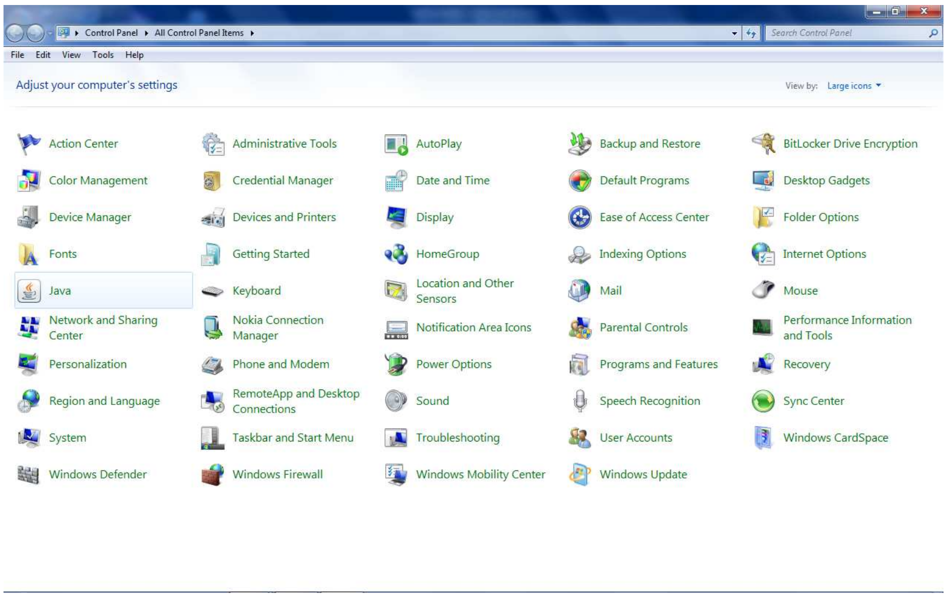


Figure 2.2(i): Icon view of the control panel

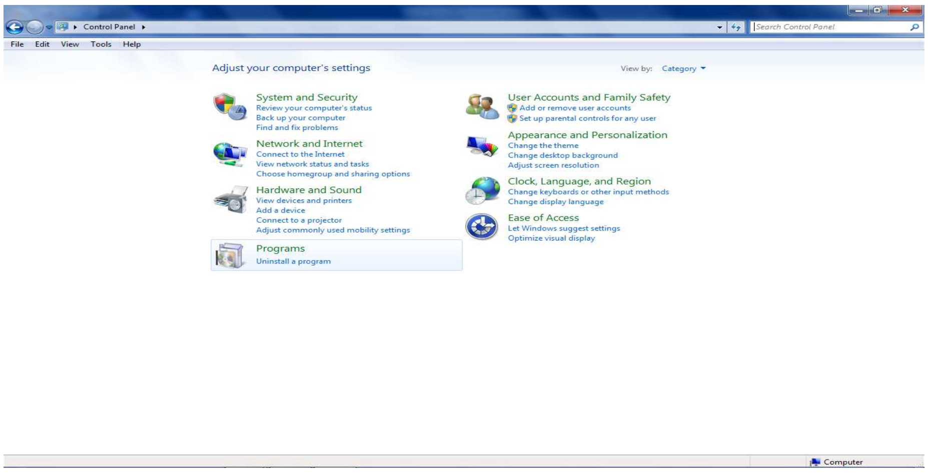

Control Panel ▸
Search Control Panel
File Edit View Tools Help
Adjust your computer's settings
View by: Category
System and Security
Review your computer's status
Back up your computer
Find and fix problems
Network and Internet
Connect to the Internet
View network status and tasks
Choose homegroup and sharing options
Hardware and Sound
View devices and printers
Add a device
Connect to a projector
Adjust commonly used mobility settings
Programs
Uninstall a program
User Accounts and Family Safety
Add or remove user accounts
Set up parental controls for any user
Appearance and Personalization
Change the theme
Change desktop background
Adjust screen resolution
Clock, Language, and Region
Change keyboards or other input methods
Change display language
Ease of Access
Let Windows suggest settings
Optimize visual display
Computer

Figure 2.2(j): Category view of the control panel

When viewing the category view of the control panel, you can use Control Panel to change settings for Windows. These settings control nearly everything about how Windows looks and works, and they allow you to set up Windows so that it's just right for you.

You can use two different methods to find the Control Panel item you are looking for:

- **Use Search:** To find a setting you are interested in or a task you want to perform, enter a word or phrase in the search box. For example, type "sound" to find specific tasks related to settings for your sound card, system sounds, and the volume icon on the taskbar.
- **Browse:** You can explore Control Panel by clicking different categories (for example, System and Security, Programs, or Ease of Access), and viewing common tasks listed under each category. Or, under View by, click either large icons or small icons to view a list of all Control Panel items.

Tip

- If you browse Control Panel by icons, you can quickly jump ahead to an item in the list by typing the first letter of the item's name. For example, to jump ahead to Gadgets, type G, and the first Control Panel item beginning with the letter G is selected in the window.

4.0 Conclusion

Windows 7 has made better impression on users than its predecessors especially with an impressive graphics user interface. It is much more user friendly and interactive.

5.0 Summary

In this unit, you have learnt basic operations on files and folders, programs and even on the access to the control panel. Proper updates on windows will help the system function well especially in the aspect of security of the operating system.

6.0 Tutor Marked Assignment

1 a) What is a scheduler
b) List two scheduling algorithms in windows 7.

2 a) What is My Computer?
b) State at least 5 major features that can be seen in My Computer and discuss?
c) Is it possible to search for a file if only a word or phrase in the file name is known?

7.0 Further Reading

Help and Support in Windows 7, copyright© 2010.

MODULE 2 – USING OPERATING SYSTEMS
UNIT 3: LINUX OPERATING SYSTEM

Content Page

1.0 INTRODUCTION

This unit introduces us to an operating system that is not so common or yet still that is fast moving into the world of portable computers and handheld gadgets. Linux is, in simplest terms, an operating system. It is the software on a computer that enables applications and the computer operator to access the devices on the computer to perform desired functions. The operating system (OS) relays instructions from an application to, for instance, the computer's processor. The processor performs the instructed task, and then sends the results back to the application via the operating system.

2.0 OBJECTIVES

On completing this unit, you would be able to:

- Have an idea on Linux
- Understand Commands used in Linux
- Understand the parts of Linux

3.0 HISTORY: USAGE AND BASIC CONCEPT

HISTORY

On August 25, 1991, a Finn computer science student named Linus Torvaldsing created an Operating system that is a variant of the UNIX operating system, using UNIX as a guideline for his operating system design. Torvalds built the core of the Linux operating system, known as the kernel. A kernel alone does not make an operating system with GNU tools the operating system will be complete. Torvalds' matching of GNU tools with the Linux kernel marked the beginning of the Linux operating system as it is known today. Linux is in many ways still only at the beginning of its potential, even though it has enjoyed tremendous success since Torvalds' first request for help in 1991.

Linux has gained strong popularity amongst UNIX developers, who like it for its portability to many platforms, its similarity to UNIX, and its free software license. Around the turn of the century, several commercial developers began to distribute Linux, including VA Linux, TurboLinux, Mandrakelinux and Red Hat. Today, Linux is a multi-billion dollar industry, with companies and governments around the world taking advantage of the operating system's security and flexibility. Thousands of companies use Linux for day-to-day use, attracted by the lower licensing and support costs. Governments around the world are deploying Linux to save money and time, with some governments commissioning their own versions of Linux. The analyst group IDC has projected Linux will be a $49 billion business by 2011, and there are any indications in the market that this figure will be achieved. The Linux Foundation is a non-profit consortium dedicated to the growth of Linux.

3.1 USAGE

One of the most noted properties of Linux is where it can be used. Windows and OS X are predominantly found on personal computing devices such as desktop and laptop computers. Other operating systems, such as Symbian, are found on small devices such as phones and PDAs, while mainframes and supercomputers found in major academic and corporate labs use specialized operating systems such as AS/400 and the Cray OS.

Linux, which began its existence as a server OS and has become useful as a desktop OS, can also be used on all of these devices. “From wristwatches to supercomputers,” is the popular description of Linux' capabilities.

An abbreviated list of some of the popular electronic devices Linux is used on today includes:

1. **Dell Inspiron Mini 9 and 12**
2. **Google Android Dev Phone 1**
3. **HP Mini 1000**
4. **One Laptop Per Child XO2**
5. **Volvo In-Car Navigation System**
6. **Motorola MotoRokr EM35 Phone**
7. **Yamaha Motif Keyboard**

These are just the most recent examples of Linux-based devices available to consumers worldwide. The Linux Foundation is building a centralized database that will list all currently offered Linux-based products, as well as archive those devices that pioneered Linux-based electronics. There are several versions of Linux and it includes; Ubuntu, Debian, Mandriva, Red Hat and Slackware.

3.2 BASIC CONCEPT

Linux is a multitasking, multiuser operating system, which means that many people can run many different applications on one computer at the same time. This differs from MS-DOS, where only one person can use the system at any one time. Under Linux, to identify yourself to the system, you must **log in**, which entails entering your **login name** (the name the system uses to identify you), and entering your **password**, which is your personal key for logging in to your account. Because only you know your password, no one else can log in to the system under your user name.

On traditional UNIX systems, the system administrator assigns you a user name and an initial password when you are given an account on the system. However, because in Linux you are the system administrator, you must set up your own account before you can log in. For the following discussions, we'll use the imaginary user name, ``larry."

In addition, each system has a **host name** assigned to it. It is this host name that gives your machine a name, gives it character and charm. The host name is used to identify individual machines on a network, but even if your machine isn't networked, it should have a host name. For our examples below, the system's host name is ``mousehouse''.

3.2.1 Creating an account.

Before you can use a newly installed Linux system, you must set up a user account for yourself. It's usually not a good idea to use the root account for normal use; you should reserve the root account for running privileged commands and for maintaining the system as discussed below. In order to create an account for yourself, log in as root and use the useradd or adduser command.

3.2.2 Logging in.

At login time, you'll see a prompt; enter your user name "larry," and press the Enter key. Next, enter your password. The characters you enter won't be echoed to the screen, so type carefully. If you mistype your password, you'll see the message and you'll have to try again. Once you have correctly entered the user name and password, you are officially logged in to the system, and are free to roam.

3.2.3 Virtual consoles.

The system's **console** is the monitor and keyboard connected directly to the system. (Because Linux is a multiuser operating system, you may have other terminals connected to serial ports on your system, but these would not be the console.) Linux, like some other versions of UNIX, provides access to **virtual consoles** (or VCs), that let you have more than one login session on the console at one time. To demonstrate this, log in to your system. Next, press Alt-F2. You should see the login: prompt again. You're looking at the second virtual console. To switch back to the first VC, press Alt-F1. You're back to your first login session.

A newly-installed Linux system probably lets you to access only the first half-dozen or so VCs, by pressing Alt-F1 through Alt-F4, or however many VCs are configured on your system. It is possible to enable up to 12 VCs--one for each function key on your keyboard. As you can see, use of VCs can be very powerful because you can work in several different sessions at the same time.

While the use of VCs is somewhat limiting (after all, you can look at only one VC at a time), it should give you a feel for the multiuser capabilities of Linux. While you're working on the first VC, you can switch over to the second VC and work on something else.

3.2.4 Shells and commands.

For most of your explorations in the world of Linux, you'll be talking to the system through a **shell**, a program that takes the commands you type and translates them into instructions to the operating system. A shell is just one interface to Linux. As soon as you log in, the system starts the shell, and you can begin entering commands. Here's a quick example. Larry logs in and is waiting at the shell **prompt**. The last line of this text is the shell's prompt, indicating that it's ready to take commands.

What is a command? When you enter a command, the shell does several things. First, it checks the command to see if it is internal to the shell. (That is a command which the shell knows how to execute itself. There are a number of these commands, and we'll go into them later.) The shell also checks to see if the command is an alias, or substitute name, for another command. If neither of these conditions apply, the shell looks for a program, on disk, having the specified name. If successful, the shell runs the program, sending the arguments specified on the command line.

3.2.5 Logging out.

Before we delve much further, we should tell you how to log out of the system. At the shell prompt, use the command to log out. There are other ways of logging out, but this is the most foolproof one.

3.2.6 Changing your password.

You should also know how to change your password. The command passwd prompts you for your old password, and a new password. It also asks you to reenter the new password for validation. Be careful not to forget your password--if you do, you will have to ask the system administrator to reset it for you

3.2.7 Files and directories.

Under most operating systems (including Linux), there is the concept of a **file**, which is just a bundle of information given a name (called a **filename**). Examples of files might be your history term paper, an e-mail message, or an actual program that can be executed. Essentially, anything saved on disk is saved in an individual file.

Files are identified by their file names. These names usually identify the file and its contents in some form that is meaningful to you. There is no standard format for file names as there is under MS-DOS and some other operating systems; in general, a file name can contain any character (except the / character--see the discussion of path names, below) and is limited to 256 characters in length. With the concept of files comes the concept of directories. A **directory** is a collection of files. It can be thought of as a ``folder'' that contains many different files. Directories are given names, with which you can identify them. Furthermore, directories are maintained in a tree-like structure; that is, directories may contain other directories.

Consequently, you can refer to a file by its **path name**, which is made up of the filename, preceded by the name of the directory containing the file. For example, let's say that Larry has a directory called papers, which contains three files: history-final, english-lit, and masters-thesis. Each of these three files contains information for three of Larry's ongoing projects.Therefore, a path name is really like a path to the file. The directory that contains a given subdirectory is known as the **parent directory**. Here, the directory paper is the parent of the notes directory.

3.3 The Code

Linux is also unique from other operating systems in that it has no single owner. Torvalds still manages the development of the Linux kernel, but commercial and private developers contribute other software to make the whole Linux operating system.

The parts of the Linux operating system will be examined as follows;

3.3.1 The Kernel

All operating systems have kernels, built around the architectural metaphor that there must be a central set of instructions to direct device hardware, surrounded by various modular layers of functionality. The Linux kernel is unique and flexible because it is also modular in nature.

Modularity is desirable because it allows developers to shed parts of the kernel they don't need to use. Typically a smaller kernel is a faster kernel, because it isn't running processes it does not need. If a device developer wants a version of Linux to run on a cell phone, she does not need the kernel functionality that deals with disk drives, Ethernet devices, or big monitor screens. She can pull out those pieces (and others), leaving just the optimized kernel to use for the phone.

The kernel of the Window operating system (which few people outside of Microsoft are allowed to look at without paying for the privilege) is a solidly connected piece of code, unable to be easily broken up into pieces. It is difficult (if not impossible) to pare down the Windows kernel to fit on a phone. This modularity is significant to the success of Linux. Modularity also effects stability and security as well. If one piece of the kernel code happens to fail, the rest of the kernel will not crash. Similarly, an illicit attack on one part of the kernel (or the rest of the operating system) might hamper that part of the code, but should not compromise the security of the whole device.

3.3.2 The Operating System

Developers need special tools (like the compilers and command lines found in GNU) to write applications that can talk to the kernel. They also need tools and applications to make it easy for outside applications to access the kernel after the application is written and installed.

This collective set of tools, combined with a kernel, is known as the operating system. It is generally the lowest layer of the computer's software that is accessible by the average user. General users get to the operating system when they access the command line.

Linux provides powerful tools with which to write their applications: developer environments, editors, and compilers are designed to take a developer's code and convert it to something that can access the kernel and get tasks done. Like the kernel, the Linux operating system is also modular. Developers can pick and choose the operating tools to provide users and developers with a new flavor of Linux designed to meet specific tasks.

3.3.3 The Environments

The windows, menus, and dialog boxes most people think of as part of the operating system are actually separate layers, known as the windowing system and the desktop environment. These layers provide the human-oriented graphical user interface (GUI) that enables users to easily work with applications in the operating system and third-party applications to be installed on the operating system.

In Linux, there a lot of choices for which windowing system and desktop environment can be used, something that Linux allow users to decide. This cannot be done in Windows and it's difficult to do in OS X. Like the operating system and kernel, there are tools and code libraries available that let application developers to more readily work with these environments (e.g., gtk+ for GNOME, Qt for KDE).

3.3.4 The Applications

Operating systems have two kinds of applications: those that are essential components of the operating system itself, and those that users will install later. Closed operating systems, like Windows and OS X, will not let users (or developers) pick and choose the essential component applications they can use. Windows developers must use Microsoft's compiler, windowing system, and so on. Linux application developers have a larger set of choices to develop their application. This allows more flexibility to build an application, but it does mean a developer will need to decide which Linux components to use.

3.3.5 The Distributions

This is the highest layer of the Linux operating system: the container for all of the aforementioned layers. A distribution's makers have decided which kernel, operating system tools, environments, and applications to include and ship to users.

Distributions are maintained by private individuals and commercial entities. A distribution can be installed using a CD that contains distribution-specific software for initial system installation and configuration. For the users, most popular distributions offer mature application management systems that allow users to search, find, and install new applications with just a few clicks of the mouse. There are, at last count, over 350 distinct distributions of Linux.

3.4 LINUX FILE STRUCTURE

In the Linux file structure files are grouped according to purpose. Ex: commands, data files, documentation. Below are some common data files commands and file structure for users;

root - The home directory for the root user
l **home** - Contains the user's home directories along with directories for services
m ftp
m HTTP
m samba

m George
l **usr** - Contains all commands, libraries, man pages, games and static files for normal operation.
m **bin** - Almost all user commands. some commands are in /bin or /usr/local/bin.
m **sbin** - System admin commands not needed on the root filesystem. e.g., most server programs.
m **include** - Header files for the C programming language. Should be below /user/lib for consistency.
m **lib** - Unchanging data files for programs and subsystems
m **local** - The place for locally installed software and other files.
m **man** - Manual pages
m **info** - Info documents
m **doc** - Documentation
l **var** - Contains files that change for mail, news, printers log files, man pages, temp files
m **file**
m **lib** - Files that change while the system is running normally
m **local** - Variable data for programs installed in /usr/local.
m **lock** - Lock files. Used by a program to indicate it is using a particular device or file
m **log** - Log files from programs such as login and syslog which logs all logins and logouts.
m **run** - Files that contain information about the system that is valid until the system is next booted
m **spool** - Directories for mail, printer spools, news and other spooled work.
m **tmp** - Temporary files that are large or need to exist for longer than they should in /tmp.
m **catman** - A cache for man pages that are formatted on demand

File and Directory management

apropos	Search the whatis database for files containing specific strings.
bdflush	Kernel daemon that saves dirty buffers in memory to the disk.
cd	Change the current directory. With no arguments "cd" changes to the users home directory.
chmod	chmod <specification> <filename> - Effect: Change the file permissions. Ex: chmod 751 myfile Effect: change the file permission to rwx for owner, re for group Ex: chmod go=+r myfile Effect: Add read permission for the owner and the group

character meanings u-user, g-group, o-other, + add permission, - remove, r-read, w-write,xexe

chown	chown <owner1> <filename> Effect: Change ownership of a file to owner1.
chgrp	chgrp <group1> <filename> Effect: Change group.
cksum	Perform a checksum and count bytes in a file.
cp	cp <source> <destination> Copy a file from one location to another.
dd	Convert and copy a file formatting according to the options. Disk or data duplication.
dir	List directory contents.
dircolors	Set colors up for ls.
file	Determines file type. Also can tell type of library (a.out or ELF).
install	Copy multiple files and set attributes.

ln	Make links between files.
locate	File locating program that uses the slocate database.
losetup	Loopback device setup.
ls	List files. Option -a, lists all, see man page "man ls" Ex: "ls Docum Projects/Linux" - The contents of the directories Docum and Projects/Linux are listed.

To list the contents of every subdirectory using the ls command:
1. Change to your home directory.
2. Type: ls -R

mkdir	Make a directory.
mknod	Make a block or character special file.
mktemp	Make temporary filename.
mv	Move or rename a file. Syntax: mv <source> <destination> Ex: mv filename directoryname/
pathchk	Check whether filenames are valid or portable.
pwd	Print or list the working directory with full path (present working directory).
rm	Delete system files (Remove files).
rmdir	rmdir <directory> - Remove a directory. The directory must be empty.
sum	Checksum and count the blocks in a file.
test	Check file types and compare values.

3.5 INSTALLATION

BIOS provides the basic functions needed to boot your machine to allow your operating system to access your hardware. Before installing, you *must* ensure that your BIOS is setup correctly; not doing so can lead to intermittent crashes or an inability to install Linux. Many BIOS set-up menus allow you to select the devices that will be used to bootstrap the system.

Here's a road map for the steps you will take during the installation process.

1. Back up any existing data or documents on the hard disk where you plan to install.
2. Gather information about your computer and any needed documentation, before starting the installation and ensure that your computer will allow you to carry out the type of installation you want. Table 2.3(a) shows the types of requirements.

Table 2.3(a): Recommended Minimum System Requirements

Install Type	**RAM**	**Hard Drive**
No desktop	32 megabytes	400 megabytes
With Desktop	128 megabytes	2 gigabytes
Server	128 megabytes	4 gigabytes

3. Create partition-able space for Linux on your hard disk.
4. Locate and/or download the installer software and any specialized driver files your machine requires (except Linux CD users).
5. Set up boot tapes/floppies/USB sticks, or place boot files (most Linux CD users can boot from one of the CDs).
6. Boot the installation system.
7. Select installation language.
8. Activate the Ethernet network connection, if available.
9. Create and mount the partitions on which Linux will be installed.
10. Watch the automatic download/install/setup of the *base system*.
11. Install a *boot loader* which can start up Linux and/or your existing system.
12. Load the newly installed system for the first time.

3.6 LICENSING

Code is contributed to the Linux kernel under a number of licenses, but all code must be compatible with version 2 of the GNU General Public License (GPLv2), which is the license covering the kernel distribution as a whole. In practice, that means that all code contributions are covered either by GPLv2 (with, optionally, language allowing distribution under later versions of the GPL) or the three-clause BSD license. Any contributions which are not covered by a compatible license will not be accepted into the kernel.

All code merged into the mainline kernel retains its original ownership; as a result, the kernel now has thousands of owners. One implication of this ownership structure is that any attempt to change the licensing of the kernel is doomed to almost certain failure. There are few practical scenarios where the agreement of all copyright holders could be obtained (or their code removed from the kernel). It is imperative that all code contributed to the kernel be legitimately free software. For that reason, code from anonymous (or pseudonymous) contributors will not be accepted. All contributors are required to "sign off" on their code, stating that the code can be distributed with the kernel under the GPL. Code which has not been licensed as free software by its owners cannot be contributed.

3.7 COMMUNITY

Linux communities come in two basic forms: developer and user communities. One of the most compelling features of Linux is that it is accessible to developers; anybody with the requisite skills can improve Linux and influence the direction of its development.

3.7.1 Developer Communities

Developer communities can volunteer to maintain and support whole distributions. The improvements to these community distributions are then incorporated into the commercial server and desktop products from these companies. The Linux kernel itself is primarily supported by its developer community as well and is one of the largest and most active free software projects in existence. A typical three-month kernel development cycle can involve over 1000 developers working for more than 100 different companies (or for no company at all).

With the growth of Linux has come an increase in the number of developers (and companies) wishing to participate in its development. Hardware vendors want to ensure that Linux supports their products well, making those products attractive to Linux users. Embedded systems vendors, who use Linux as a component in an integrated product, want Linux to be as capable and well-suited to the task at hand as possible. Distributors and other software vendors who base their products on Linux have a clear interest in the capabilities, performance, and reliability of the Linux kernel. Other developer communities focus on different applications and environments that run on Linux, such as Firefox, OpenOffice.org, GNOME, and KDE.

3.7.2 Users Communities

End users, too, can make valuable contributions to the development of Linux. With online communities such as Linux.com, LinuxQuestions, and the many and varied communities hosted by distributions and applications, the Linux user base is an often vocal, usually positive advocate and guide for the Linux operating system. The Linux community is not just a presence online. Local groups known as Linux Users Groups (LUGs) often meet to discuss issues regarding the Linux operating system, and provide other local users with free demonstrations, training, technical support, and install fests.

3.8 DEVELOPMENT

Linux is an operating system that is comprised of many different development languages. A very large percentage of the distributions' code is written in either the C (52.86%) or C++ (25.56%) languages. All of the rest of the code falls into single-digit percentages, with Java, Perl, and Lisp rounding out the rest of the top 5 languages. The Linux kernel itself has an even more dominant C presence, with over 95 percent of the kernel's code written in that language. But other languages make up the kernel as well, making it more heterogenous than other operating systems.

The kernel community has evolved its own distinct ways of operating which allow it to function smoothly (and produce a high-quality product) in an environment where thousands of lines of code are being changed every day. This means the Linux kernel development process differs greatly from proprietary development methods.

The kernel's development process may come across as strange and intimidating to new developers, but there are good reasons and solid experience behind it. A developer who does not understand the kernel community's ways (or, worse, who tries to flout or circumvent them) will have a frustrating experience in store. The development community, while being helpful to those who are trying to learn, has little time for those who will not listen or who do not care about the development process. While many Linux developers still use text-based tools such as Emacs or Vim to develop their code; Eclipse, Anjuta, and Netbeans all provide more robust integrated development environments for Linux.

4.0 CONCLUSION

Linux operating system is an interesting operating system that is now gaining grounds in the IT industry among common devices. Though because of its openness in design, there is yet standard documentation to be published on it compared to other operating system.

5.0 SUMMARY

In this unit, you have learnt that Linux is very similar to other operating systems, such as Windows and OS X. But something sets Linux apart from these operating systems. Linux was designed to be a multi-tasking, multi-user system and better able to run more than one program at the same time, and more secure than many operating systems. Also, the installation process is quite different from the usual in other Operating system like Windows.

6.0 TUTOR MARKED ASSIGNMENT

1. Explain the major function of the kernel and shell in linux .

7.0 FURTHER READING

1. CTDP Linux Files and Command Reference;

http://www.comptechdoc.org/os/linux/commands/linux_crfmanview.html

2. 3 Linux Tutorial; http://tldp.org/LDP/gs/node5.html © Copyright 1998-2011 by Vincent Veselosky.

3. Linux Installation, Step by Step; http://www.control-escape.com/linux/lx-install.html

MODULE 3 – LABORATORY EXERCISES USING SQL

UNIT 1: CONCEPTS OF SQL

1.0 INTRODUCTION

SQL is the standard language for all databases. SQL is a nonprocedural language, in contrast to the procedural or third-generation languages (3GLs) such as COBOL and C that had been created up to that time. It is called a nonprocedural language because it describes what operation is to be performed rather how it is been performed. This unit studies the history of Structure Query Language and discusses the concepts of Relational Database Management System (RDMS) in relation to its use in SQL. It also highlights some basic commands and terminologies.

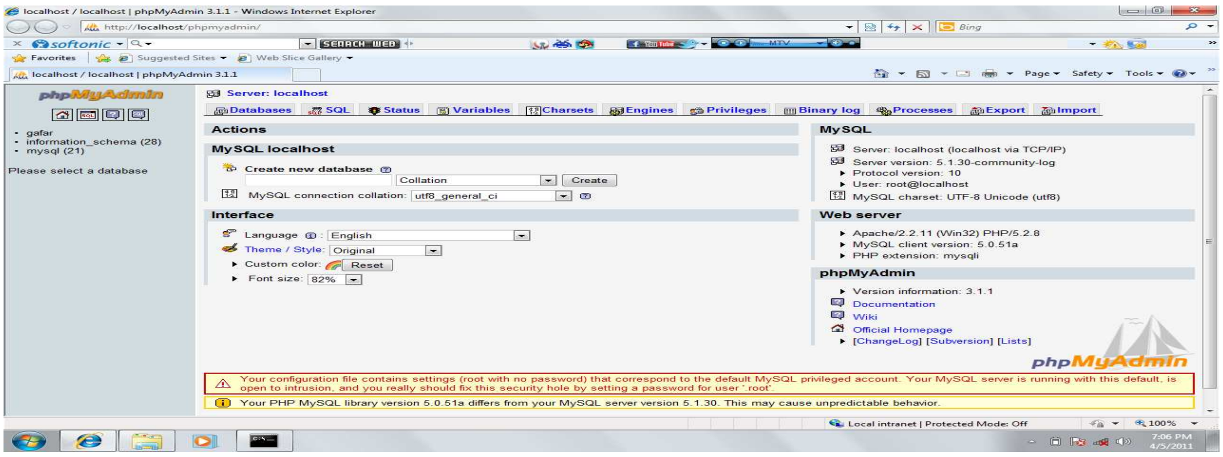
2.0 OBJECTIVES

After completing this unit, you should be able to do the following:

i. Explain the history of the Structure Query Language
ii. Explain the history of databases
iii. Explain the concept of relational database management system
iv. List and explain the basic SQL commands
v. Define some SQL terminologies

3.0 INTRODUCTION TO SQL AND RELATIONAL DATABASES

The database used in this course is the MySQL platform. To work with MySQL platform user needs to get a copy of the MySQL platform setup or the WAMP Server and that will be uused for the purpose of this course. When the user launches the application after installation the Startup window is shown in Figure 3.1(a).



localhost / localhost | phpMyAdmin 3.1.1 - Windows Internet Explorer

http://localhost/phpmyadmin/

Server: localhost

Databases | SQL | Status | Variables | Charsets | Engines | Privileges | Binary log | Processes | Export | Import

- gafar
- information_schema (28)
- mysql (21)

Please select a database

Actions

MySQL localhost

Create new database
Collation | Create
MySQL connection collation: utf8_general_ci

Interface

Language: English
Theme / Style: Original
Custom color: Reset
Font size: 82%

MySQL

Server: localhost (localhost via TCP/IP)
Server version: 5.1.30-community-log
Protocol version: 10
User: root@localhost
MySQL charset: UTF-8 Unicode (utf8)

Web server

Apache/2.2.11 (Win32) PHP/5.2.8
MySQL client version: 5.0.51a
PHP extension: mysqli

phpMyAdmin

Version information: 3.1.1
Documentation
Wiki
Official Homepage
[ChangeLog] [Subversion] [Lists]

Your configuration file contains settings (root with no password) that correspond to the default MySQL privileged account. Your MySQL server is running with this default, is open to intrusion, and you really should fix this security hole by setting a password for user 'root'.

Your PHP MySQL library version 5.0.51a differs from your MySQL server version 5.1.30. This may cause unpredictable behavior.

Local intranet | Protected Mode: Off

7:06 PM 4/5/2011

Figure 3.1(a) : An interface showing the start up window of the MySQL platform.

3.1 HISTORY OF SQL AND RELATIONAL DATABASES

3.1.1 History of SQL

The history of SQL began in an IBM laboratory in San Jose, California, where SQL was developed in the late 1970s. The initials stand for Structured Query Language, and the language itself is often referred to as "sequel." It was originally developed for IBM's DB2 product (a relational database management system, or RDBMS, that can still be bought today for various platforms and environments). In fact, SQL makes an RDBMS possible. Two standards organizations, the American National Standards Institute (ANSI) and the International Standards Organization (ISO), currently promote SQL standards to industry. The ANSI-92 standard is the standard for the SQL used throughout this book. Although these standard-making bodies prepare standards for database system designers to follow, all database products differ from the ANSI standard to some degree. In addition, most systems provide some proprietary extensions to SQL that extend the language into a true procedural language.

SQL has the following advantages:

- It is efficient
- It is easy to learn and use
- It is functional complete (with SQL, data in a table can be defined, retrieved and manipulated)

3.1.2 History of Relational Databases

Database systems store information in every conceivable business environment. From large tracking databases such as airline reservation systems to a child's baseball card collection, database systems store and distribute the data that we depend on. Until the last few years, large database systems could be run only on large mainframe computers. These machines have traditionally been expensive to design, purchase, and maintain. However, today's generation of powerful, inexpensive workstation computers enables programmers to design software that maintains and distributes data quickly and inexpensively.

The most popular data storage model is the relational database, which grew from the seminal paper "A Relational Model of Data for Large Shared Data Banks," written by Dr. E. F. Codd in 1970. SQL evolved to service the concepts of the relational database model. Dr. Codd defined 13 rules, oddly enough referred to as Codd's 12 Rules, for the relational model:

Rule 0: A relational DBMS must be able to manage databases entirely through its relational capabilities.
Rule 1: Information rule-- All information in a relational database (including table and column names) is represented explicitly as values in tables.
Rule 2: Guaranteed access--Every value in a relational database is guaranteed to be accessible by using a combination of the table name, primary key value, and column name.
Rule 3: Systematic null value support--The DBMS provides systematic support for the treatment of null values (unknown or inapplicable data), distinct from default values, and independent of

any domain.
Rule 4: Active, online relational catalog--The description of the database and its contents is represented at the logical level as tables and can therefore be queried using the database language.
Rule 5: Comprehensive data sublanguage--At least one supported language must have a well-defined syntax and be comprehensive. It must support data definition, manipulation, integrity rules, authorization, and transactions.
Rule 6: View updating rule--All views that are theoretically updatable can be updated through the system.
Rule 7: Set-level insertion, update, and deletion--The DBMS supports not only set-level retrievals but also set-level inserts, updates, and deletes.
Rule 8: Physical data independence--Application programs and ad hoc programs are logically unaffected when physical access methods or storage structures are altered.
Rule 9: Logical data independence--Application programs and ad hoc programs are logically unaffected, to the extent possible, when changes are made to the table structures.
Rule 10: Integrity independence--The database language must be capable of defining integrity rules. They must be stored in the online catalog, and they cannot be bypassed.
Rule 11: Distribution independence--Application programs and ad hoc requests are logically unaffected when data is first distributed or when it is redistributed.
Rule 12: Nonsubversion--It must not be possible to bypass the integrity rules defined through the database language by using lower-level languages.

3.2 INTRODUCTION TO DATABASE

A table will be used and it is called the employees table. The structure of the employees table is shown below:

Table 3.1(a): The structure of employees table

Name	**Null?**	**Type**
EMPLOYEE_ID		NUMBER(6)
FIRST_NAME		VARCHAR2(20)
LAST_NAME	NOT NULL	VARCHAR2(25)
SALARY		NUMBER(8,2)
HIRE_DATE	NOT NULL	DATE
DEPARTMENT_ID		NUMBER(4)

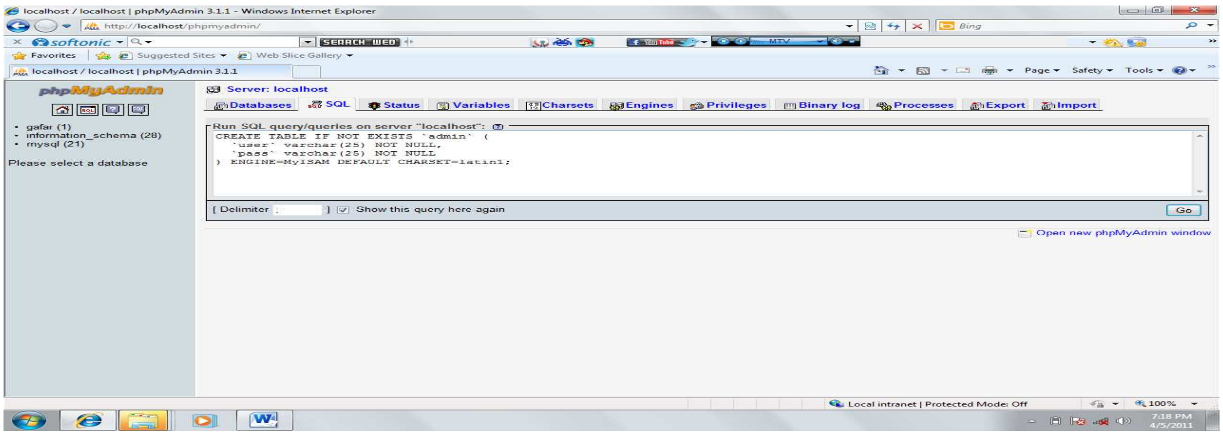
The content of the employee table is shown in the table below:

Table 3.1(b): The data in the employees table

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	HIRE_DATE	DEPARTMENT_ID
100	Steven	King	24000	17-JUN-87	90
101	Neena	Kochhar	17000	21-SEP-89	20
102	Lex	De Haan	17000	13-JAN-93	40
103	Alexander	Hunold	9000	03-JAN-90	60
104	Bruce	Ernst	6000	21-MAY-91	60
105	David	Austin	4800	25-JUN-97	50
106	Valli	Pataballa	4800	05-FEB-98	50
107	Diana	Lorentz	4200	07-FEB-99	60
108	Nancy	Greenberg	12000	17-AUG-94	70
109	Daniel	Faviet	9000	16-AUG-94	100
110	John	Chen	8200	28-SEP-97	100
111	Ismael	Sciarra	7700	30-SEP-97	70
112	Jose Manuel	Urman	7800	07-MAR-98	100
113	Luis	Popp	6900	07-DEC-99	100
114	Den	Raphaely	11000	07-DEC-94	30
115	Alexander	Khoo	3100	18-MAY-95	80
116	Shelli	Baida	2900	24-DEC-97	30
117	Sigal	Tobias	2800	24-JUL-97	80
118	Guy	Himuro	2600	15-NOV-98	30
119	Karen	Colmenares	2500	10-AUG-99	30

CREATING TABLES ON MySQL(WAMP SERVER)

Users can create tables graphically by >> Selecting Database>>Enter Table Name>>Click Create Table from the start up window. Alternatively Click SQL from the start up window to Create using SQL Query, an illustration is displayed in the Figure 3.1(b).



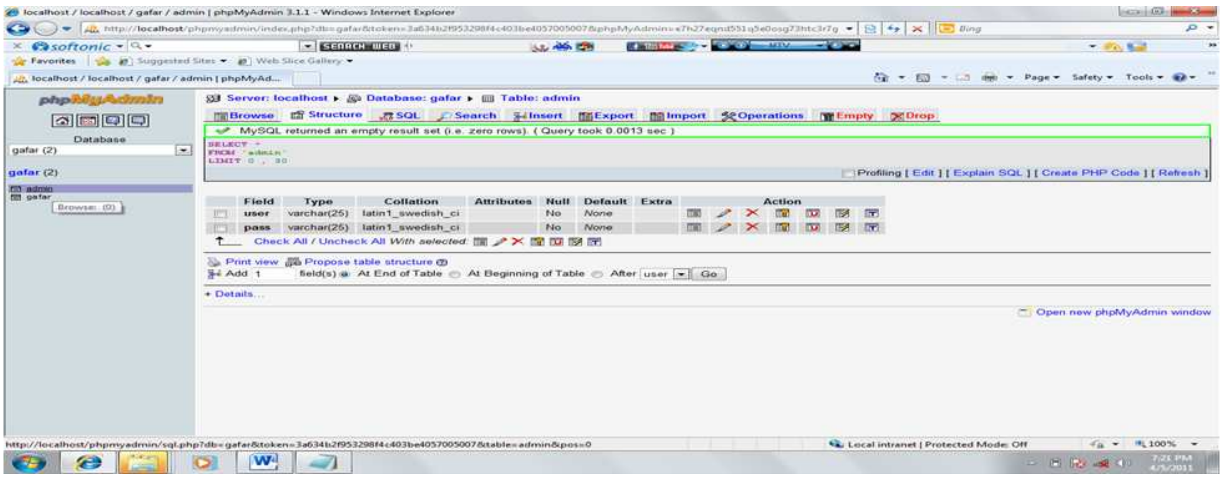
Run SQL query/queries on server "localhost":

```
CREATE TABLE IF NOT EXISTS `admin` (
  `user` varchar(25) NOT NULL,
  `pass` varchar(25) NOT NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

[Delimiter ;] Show this query here again

Figure 3.1(b) : An interface showing the SQL view.

OUTPUT



Server: localhost ▸ Database: gafar ▸ Table: admin

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0013 sec)

```
SELECT *
FROM `admin`
LIMIT 0 , 30
```

Field	Type	Collation	Attributes	Null	Default	Extra
user	varchar(25)	latin1_swedish_ci		No	None	
pass	varchar(25)	latin1_swedish_ci		No	None	

Figure 3.1(c) : An interface showing the output of a Create Table Query.

3.2.1 DATABASE TERMINOLOGIES AND SQL COMMANDS

3.2.1.1 Database Terminologies

The storage and maintenance of valuable data is the reason for any database's existence. . A table is the most common and simplest form of data. The following section takes a closer look at the elements within a table

A Field: Every table is broken up into smaller entities called fields. The fields in the employees table consist of employee_id, first_name, last_name, salary, hire_date, department_id. These fields categorize the specific information that is maintained in a given table. A *field* is a column in a table that is designed to maintain specific information about every record in the table.

A Record: A *record*, also called a *row* of data, is each individual entry that exists in a table. Looking at the employees table shown in Table 1.1(b) consider the following first record in that table:

Table 3.1(c): A record inside employees table

100	Steven	King	24000	17-JUN-87	90

The record is obviously composed of an employee_id, first_name, last_name,hire_date and department_id. For every distinct employee, there should be a corresponding record in the employees table. A record is a horizontal entity in a table. **Note:** A *row of data* is an entire record in a relational database table

A Column: A *column* is a vertical entity in a table that contains all information associated with a specific field in a table. For example, a column in the employees table having to do with the first names of employees would consist of the following:

Table 3.1(d): A column in the employees table showing the 20 rows selected

FIRST_NAME
Steven
Neena
Lex
Alexander
Bruce
David
Valli
Diana
Nancy
Daniel
John
Ismael
Jose Manuel
Luis
Den
Alexander
Shelli
Sigal
Guy
Karen

This column is based on the first_name, i.e employees' first_name. A column pulls information about a certain field from every record within a table.

The Primary Key: A *primary key* is a column that makes each row of data in the table unique in a relational database. The primary key in the employees table is employee_id which is typically initialized during the table creation process. The nature of the primary key is to ensure that all product identifications are unique, so that each record in the employees table has its own employee_id. Primary keys prevents the possibility of a duplicate record in a table and they do not take null as values.

A NULL Value

NULL is the term used to represent a missing value. A NULL value in a table is a value in a field that appears to be blank. A field with a NULL value is a field with no value. It is very important to understand that a NULL value is different from a zero value or a field that contains spaces. A field with a NULL value is one that has been left blank during record creation.

3.2.1.2 SQL COMMANDS

The following section discusses the basic categories of commands used in SQL to perform various functions. These functions include building database objects, manipulating objects, populating database tables with data, updating existing data in tables, deleting data, performing database queries, controlling database access, and overall database administration.

The main categories are

- *DQL* (Data Query Language)
- *DML* (Data Manipulation Language)
- *DDL (*Data Definition Language)

- *DCL* (Data Control Language)

- Data administration commands

- Transactional control commands

Data Query Language

Though comprised of only one command, Data Query Language (*DQL*) is the most concentrated focus of SQL for modern relational database users. A *query* is an inquiry to the database for information. A query is usually issued to the database through an application interface or via a command line prompt. The basic command is as follows:

- SELECT

The SELECT command will be discussed in Unit 2.

Data Manipulation Language

Data Manipulation Language, DML , is the part of SQL used to manipulate data within objects of a relational database.

There are three basic *DML* commands:

- INSERT

- UPDATE

- DELETE

These commands are discussed in detail in Unit 3 of this module.

Data Definition Language

Data Definition Language, DDL, is the part of SQL that allows a database user to create and restructure database objects, such as the creation or the deletion of a table.

Some of the most fundamental commonly used *DDL* commands include the following:

- CREATE TABLE
- ALTER TABLE
- DROP TABLE
- CREATE INDEX
- ALTER INDEX
- DROP INDEX
- CREATE VIEW

These commands, accompanied by many options and clauses, are used to compose queries against a relational database. Queries, from simple to complex, from vague to specific, can be easily created.

Data definition commands are discussed in details in Unit 4

Data Control Language

Data control commands in SQL allow you to control access to data within the database. These *DCL* commands are normally used to create objects related to user access and also control the distribution of privileges among users. Some data control commands are as follows:

- ALTER PASSWORD
- GRANT
- REVOKE
- CREATE SYNONYM

Data Administration Commands

Data administration commands allow the user to perform audits and perform analyses on operations within the database. They can also be used to help analyze system performance. Two general data administration commands are as follows:

- START AUDIT
- STOP AUDIT

Do not get data administration confused with database administration. *Database administration* is the overall administration of a database, which envelops the use of all levels of commands. *Database administration* is much more specific to each SQL implementation than are those core commands of the SQL language.

Transactional Control Commands

In addition to the previously introduced categories of commands, there are commands that allow the user to manage database transactions.

- COMMIT Saves database transactions
- ROLLBACK Undoes database transactions
- SAVEPOINT Creates points within groups of transactions in which to ROLLBACK
- SET TRANSACTION Places a name on a transaction

4.0 CONCLUSION

You have been introduced to the standard language of SQL and have been given a brief history of how the standard has evolved over the last several years. Database systems and current technologies were also discussed, including the relational database all of which are vital to your understanding of SQL.

The database that will be used during your course of study was also introduced. You should have acquired some overall background knowledge of the fundamentals of SQL and should have understood the concept of a modern database.

5.0 SUMMARY

In this unit, you should have learned:

- The history of the Structured Query Language
- The definition and history of relational databases
- Some database terminologies
- List and explain the basic SQL commands categories

6.0 TUTOR MARKED ASSIGNMENTS

1. What makes SQL a non procedural language?
2. Explain the connection between SQL and relational databases?
3. List and explain the basic SQL commands that we have?

4. Mention two attributes that a Primary key must have?
5. What is a null value?

7.0 FURTHER READING AND OTHER RESOURCES

1. Oracle Database 10g SQL Fundamentals I
2. Teach Yourself SQL in 24 Hours by Ryan Stephens & Ron Plew
3. Teach Yourself SQL in 21 Days, 2nd Edition by Ron Plew & Ryan Stephenss

MODULE 3 - LABORATORY EXERCISES USING SQL

UNIT 2 - DATA QUERY LANGUAGE

Content Page

1.0 INTRODUCTION

The structured query language (SQL) SELECT statement retrieves information from the database. This unit examines different ways of using the SELECT statement and explains with more workable examples.

2.0 UNIT OBJECTIVES

After completing this unit, you should be able to do the following

i. List the capabilities of SQL SELECT statements
ii. Write an SQL query
iii. Select and list all rows and columns from a table
iv. Select and list specific rows and columns from a table

3.0 INTRODUCTION TO SQL SELECT STATEMENT

Apart from the use of SQL SELECT statement described in section 1.0, SELECT statement can also be used to restrict the columns that are displayed. The SELECT statement has the following capabilities:

Projection: Choose the columns in a table that are returned by a query. Choose as few or as many of the columns as needed.

Selection: Choose the rows in a table that are returned by a query. Various criteria can be used to restrict the rows that are retrieved.

Joining: Bring together data that is stored in different tables by specifying the link between them.

The **Projection** and **Selection** capabilities will be discussed in this Unit.

3.1 THE SQL SELECT STATEMENT

The general syntax of the SQL SELECT STATEMENT is:

SELECT *|{[DISTINCT] *column|expression* [*alias*],...}
FROM *table*

In its simplest form, a SELECT statement must include the following:

- A SELECT clause, which specifies the columns to be displayed
- A FROM clause, which identifies the table containing the columns that are listed in the SELECT clause.

In the syntax:

SELECT	is a list of one or more columns
*	selects all columns

DISTINCT	suppresses duplicates
column/expression	selects the named column or the expression
alias	gives selected columns different headings
FROM table	specifies the table containing the columns

3.1.1 Selecting All Columns

You can display all columns of data in a table by following the SELECT keyword with an asterisk (*). For example, to display all the data contained in the employees table,

INPUT: SELECT * FROM EMPLOYEES;

OUTPUT:

Table 3.2(a): Output shown on Selecting all the rows and columns in employees table

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	HIRE_DATE	DEPARTMENT_ID
100	Steven	King	24000	17-JUN-87	90
101	Neena	Kochhar	17000	21-SEP-89	20
102	Lex	De Haan	17000	13-JAN-93	40
103	Alexander	Hunold	9000	03-JAN-90	60
104	Bruce	Ernst	6000	21-MAY-91	60
105	David	Austin	4800	25-JUN-97	50
106	Valli	Pataballa	4800	05-FEB-98	50
107	Diana	Lorentz	4200	07-FEB-99	60
108	Nancy	Greenberg	12000	17-AUG-94	70
109	Daniel	Faviet	9000	16-AUG-94	100
110	John	Chen	8200	28-SEP-97	100
111	Ismael	Sciarra	7700	30-SEP-97	70
112	Jose Manuel	Urman	7800	07-MAR-98	100
113	Luis	Popp	6900	07-DEC-99	100
114	Den	Raphaely	11000	07-DEC-94	30
115	Alexander	Khoo	3100	18-MAY-95	80
116	Shelli	Baida	2900	24-DEC-97	30
117	Sigal	Tobias	2800	24-JUL-97	80
118	Guy	Himuro	2600	15-NOV-98	30
119	Karen	Colmenares	2500	10-AUG-99	30

In the example, the employees table contains six columns:

EMPLOYEE_ID, FIRST_NAME, LAST_NAME, SALARY, HIRE_DATE, DEPARTMENT_ID. The table contains twenty rows, one for each employee.

You can display all columns in the table by listing all the columns after the SELECT keyword. For example, the following SQL statement (like the above example) displays all columns and all rows of the EMPLOYEES table.

INPUT:
```
SELECT employee_id, first_name, last_name, salary, hire_date, department_id
FROM employees;
```
OUTPUT:

Table 3.2(b): An alternative way of selecting all the rows and columns in the employees table

EMPLOYEE_ID	**FIRST_NAME**	**LAST_NAME**	**SALARY**	**HIRE_DATE**	**DEPARTMENT_ID**
100	Steven	King	24000	17-JUN-87	90
101	Neena	Kochhar	17000	21-SEP-89	20
102	Lex	De Haan	17000	13-JAN-93	40
103	Alexander	Hunold	9000	03-JAN-90	60
104	Bruce	Ernst	6000	21-MAY-91	60
105	David	Austin	4800	25-JUN-97	50
106	Valli	Pataballa	4800	05-FEB-98	50
107	Diana	Lorentz	4200	07-FEB-99	60
108	Nancy	Greenberg	12000	17-AUG-94	70
109	Daniel	Faviet	9000	16-AUG-94	100
110	John	Chen	8200	28-SEP-97	100
111	Ismael	Sciarra	7700	30-SEP-97	70
112	Jose Manuel	Urman	7800	07-MAR-98	100
113	Luis	Popp	6900	07-DEC-99	100
114	Den	Raphaely	11000	07-DEC-94	30
115	Alexander	Khoo	3100	18-MAY-95	80
116	Shelli	Baida	2900	24-DEC-97	30
117	Sigal	Tobias	2800	24-JUL-97	80
118	Guy	Himuro	2600	15-NOV-98	30
119	Karen	Colmenares	2500	10-AUG-99	30

3.1.2 Selecting Specific Columns

You can use the SELECT statement to display specific columns of the table by specifying the column names, separated by commas. For example, to display the employee_id, first_name and last_name of all employees, the query below achieves this;

INPUT:
```
SELECT employee_id, first_name, last_name
FROM employees;
```
OUTPUT:

Table 3.2(c): Selecting all the rows but specific columns in the employees table

EMPLOYEE_ID	**FIRST_NAME**	**LAST_NAME**
100	Steven	King
101	Neena	Kochhar
102	Lex	De Haan
103	Alexander	Hunold
104	Bruce	Ernst
105	David	Austin
106	Valli	Pataballa
107	Diana	Lorentz
108	Nancy	Greenberg
109	Daniel	Faviet
110	John	Chen
111	Ismael	Sciarra
112	Jose Manuel	Urman
113	Luis	Popp
114	Den	Raphaely
115	Alexander	Khoo
116	Shelli	Baida
117	Sigal	Tobias
118	Guy	Himuro
119	Karen	Colmenares

Note: In the SELECT clause, specify the columns that you want, in the order in which you want them to appear in the output.

3.1.3 Arithmetic Operations With The Select Statement

You may need to modify the way in which data is displayed, or you may want to perform calculations or look at what-if scenarios. These are all possible using arithmetic expressions. An arithmetic expression can contain column names, constant numeric values and the arithmetic operators. The table below lists the arithmetic operators that are available in SQL.

Table 3.2(d): Arithmetic operators used in SQL

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide

Example 3.2(a): An increment of 700 naira is to be added to employees' salary.

INPUT:

```
SELECT last_name, salary, salary+700
FROM employees;
```

The query above displays the last_name, old salary and the new incremented salary of all employees.

OUTPUT:

Table 3.2(e): A table showing the effect of arithmetic operators on query results

LAST_NAME	**SALARY**	**SALARY+700**
King	24000	24700
Kochhar	17000	17700
De Haan	17000	17700
Hunold	9000	9700
Ernst	6000	6700
Austin	4800	5500
Pataballa	4800	5500
Lorentz	4200	4900
Greenberg	12000	12700
Faviet	9000	9700
Chen	8200	8900
Sciarra	7700	8400
Urman	7800	8500
Popp	6900	7600
Raphaely	11000	11700

Khoo	3100	3800
Baida	2900	3600
Tobias	2800	3500
Himuro	2600	3300
Colmenares	2500	3200

20 rows selected.

Note: The resultant calculated column SALARY+700 is not a new column in the employees table; it is for display only. By default, the name of a new column comes from the calculation that generated it- in this case salary+700.

3.1.4 Conditional Selection and Comparison Condition

You can restrict the rows that are returned from a query by using the WHERE clause. A WHERE clause contains a condition that must be met and it directly follows the FROM clause. If the condition is true, the rows meeting the condition is returned. Modifying the SELECT syntax gives the syntax of the WHERE clause shown below;

SELECT *|{[DISTINCT] *column|expression* [*alias*],...}
FROM *table*
[WHERE *condition(s)*];

In the syntax:

WHERE	restricts the query to rows that meet a condition
Condition	is composed of column names, expressions, constants and a comparison operator

The WHERE clause can compare values in columns, literal values, arithmetic expressions, or functions. It consists of three elements:

- Column name
- Comparison Condition
- Column name, constant, or list of values

Example 3.2(b): To write a query that retrieves the employee ID, first name, last name and the department ID of employees in department 30, the query below achieves this;

INPUT:

```
SELECT employee_id, first_name, last_name, department_id
FROM employees
WHERE department_id=30;
```

OUTPUT:

Table 3.2(f): A table showing specific rows and specific columns in the employees table

EMPLOYEE_ID	**FIRST_NAME**	**LAST_NAME**	**DEPARTMENT_ID**
114	Den	Raphaely	30
116	Shelli	Baida	30
118	Guy	Himuro	30
119	Karen	Colmenares	30

4rows selected.

Comparison Conditions

Comparison conditions are used in conditions that compare one expression to another value or expression. They are used in the WHERE clause in the following format:

…WHERE *expr operator value*

The table below shows a list of some Comparison Conditions that can be used with the WHERE clause.

Table 3.2(g): List of Comparison Conditions operators

Operator	**Meaning**
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to
BETWEEN …AND…	Between two values (inclusive)
IN (set)	Match any of a list of values

Example 3.2(c): Write a query to retrieve the last name and salary from the EMPLOYEES table for any employee whose salary is less than or equal to 2900.

INPUT:
```
SELECT last_name, salary
FROM employees
WHERE salary<=2900;
```
OUTPUT:

Table 3.2(h): A table showing the last names and salary of those earning below 2900 or 2900

LAST_NAME	**SALARY**
Baida	2900
Tobias	2800
Himuro	2600
Colmenares	2500

4 rows selected.

Example 3.2(d): Write a query to retrieve the last name and salary from the EMPLOYEES table for any employee whose salary is between 2500 and 2600.

INPUT:
```
SELECT last_name, salary
FROM employees
WHERE salary BETWEEN 2500 AND 2900;
```

OUTPUT:

Table 3.2(i): A table showing the last names and salary of those earning between 2500 and2900

LAST_NAME	**SALARY**
Baida	2900
Tobias	2800
Himuro	2600
Colmenares	2500

4 rows selected.

Note: Values that are specified with the BETWEEN condition are inclusive. The lower limit must be specified first.

Example: Write a query to retrieve the employee numbers, last name and salary from the EMPLOYEES table for any employee whose employee number is 101,103,105,106 or 110

INPUT:
```
SELECT employee_id, last_name, salary
FROM employees
WHERE employee_id IN (101,103,105,106,110);
```
OUTPUT:

Table 3.2(j): A table showing the employee_id, last_name and salary of employees 101,103,105,106 and 110

EMPLOYEE_ID	**LAST_NAME**	**SALARY**
101	Kochhar	17000
103	Hunold	9000
105	Austin	4800
106	Pataballa	4800
110	Chen	8200

5 rows selected

3.2 Sorting

The order of rows that are returned in a query result is undefined. The ORDER BY clause can be used to sort the rows. The ORDER BY clause sorts retrieved rows in the following formats;

- ASC: ascending order,default
- DESC: descending order

The ORDER BY clause comes last in the SELECT statement.

Example 3.2(e): Write a query to display the employee number, first_name and hire_date of employees. The result should be displayed in such a way that the first names will be arranged alphabetically.

INPUT: Either of the two queries written below solves the problem

Query 1:

```
SELECT employee_id, first_name, hire_date
FROM employees
ORDER BY first_name;
```

Query 2:

```
SELECT employee_id, first_name, hire_date
FROM employees
ORDER BY first_name ASC;
```

OUTPUT:

Table 3.2(k): A table showing employee id, first name and hire_date with first names alphabetically arranged.

EMPLOYEE_ID	**FIRST_NAME**	**HIRE_DATE**
103	Alexander	03-JAN-90
115	Alexander	18-MAY-95
104	Bruce	21-MAY-91
109	Daniel	16-AUG-94
105	David	25-JUN-97
114	Den	07-DEC-94
107	Diana	07-FEB-99
118	Guy	15-NOV-98
111	Ismael	30-SEP-97
110	John	28-SEP-97
112	Jose Manuel	07-MAR-98
119	Karen	10-AUG-99
102	Lex	13-JAN-93
113	Luis	07-DEC-99
108	Nancy	17-AUG-94
101	Neena	21-SEP-89
116	Shelli	24-DEC-97
117	Sigal	24-JUL-97
100	Steven	17-JUN-87
106	Valli	05-FEB-98

20 rows selected.

Example 3.2(e): Write a query to display the employee numbers, first names and salaries of all employees in a way the highest paid employee will appear first and the least paid will appear last.

INPUT:

```
SELECT employee_id, first_name, salary
FROM employees

ORDER BY salary DESC;
```

OUTPUT:

Table 3.2(l): A table showing employee id, first name and salary with salaries alphabetically arranged in descending order

EMPLOYEE_ID	FIRST_NAME	SALARY
100	Steven	24000
101	Neena	17000
102	Lex	17000
108	Nancy	12000
114	Den	11000
103	Alexander	9000
109	Daniel	9000
110	John	8200
112	Jose Manuel	7800
111	Ismael	7700
113	Luis	6900
104	Bruce	6000
105	David	4800
106	Valli	4800
107	Diana	4200
115	Alexander	3100
116	Shelli	2900
117	Sigal	2800
118	Guy	2600
119	Karen	2500

20 rows selected.

4.0 CONCLUSION

The Data Query Language which basically comprises the SELECT statement is very important for data retrieval from the database. The SELECT statement has 3 major capabilities which are Selection, Projection and Join.

Selection has to do with the rows that are retrieved from the database. The SELECT statement in conjunction with the WHERE clause allows the rows retrieved from the database to be restricted or limited.

Projections deals with the columns that ate retrieved from the database.

Join deals with the retrieval of data from more than one table at the same time.

Query results can be sorted by using the ORDER BY clause. Sort operation can be in ascending order or descending order. The default is ascending order.

Note: You can use SQL from within a programming language. Embedded SQL is normally a language extension, most commonly seen in COBOL, in which SQL is written inside of and compiled with the program. Microsoft has created an entire Application Programming Interface (API) that enables programmers to use SQL from Visual Basic, C or C++.

5.0 SUMMARY

In this unit, you should have learnt how to:

- Write a SELECT statement that:
  - Returns all rows and columns from a table
  - Returns specified columns from a table
- Use the WHERE clause to restrict rows of output
  - Use the comparison conditions
  - Use the BETWEEN, IN, NOT conditions
- Use the ORDER BY clause to sort rows of output

6.0 TUTOR MARKED ASSIGNMENT

1. Use the employees table shown in Table 3.2(b) to answer the following questions.
   a. Write a query that returns all the fields of everyone in the database whose salary is 4800
   b. Write a query that returns all employees fields whose employee id is 105 or 107 or 117
   c. Write a query to display all the fields of those that are earning between 2500 and 3000
   d. Write a query to display all the fields of all employees with the earliest hire date appearing first and the latest date appearing last.
   e. Write a query to display all employees with their department id arranged in descending order.
   f. Write a query to display the employee_id and last names of all employees
   g. Write a table to display the employee_id, first_name and salary of employees that have employee_id 103, 105, 106, 108, 109 or 119.
   h. Write a query to display the employee_id, last_name and hire dates of all employees in such a way that the earliest date appears first.

7.0 FURTHER READING AND OTHER RESOURCES

1. Oracle Database 10g SQL Fundamentals I
2. Teach Yourself SQL in 24 Hours by Ryan Stephens & Ron Plew
3. PHP5 and MySQL Bible by Tim Converse and Joyce Park with Clark Morgan

MODULE 3 - LABORATORY EXERCISES USING SQL

UNIT 3 - DATA MANIPULATION LANGUAGE

Content Page

1.0 INTRODUCTION

Data manipulation Language (DML) is a core part of SQL. When you want to add, update or delete data in the database, you execute a DML statement. A collection of DML statements that form a logical unit of work is called a *transaction.* This unit examines the different major operations you can perform on a database namely: INSERT, UPDATE and DELETE using the DML statement.

2.0 UNIT OBJECTIVES

After completing this unit, you should be able to do the following:

- Describe each data manipulation language (DML) statement
- Insert rows into a table
- Update records in a table
- Delete rows from a table

3.0 INTRODUCTION TO THE DATA MANIPULATION LANGUAGE

A DML statement is executed when you:

- Add new rows to a table
- Modify existing rows in a table
- Remove existing rows from a table

3.1 WORKING WITH DATA MANIPULATION LANGUAGE

Consider a banking database. When a bank customer transfers money from a savings account to a checking account, the transaction might consist of three separate operations: decrease the savings account, increase the checking account and record the transaction in the transaction journal. When something prevents one of the statements in the transaction from executing, the other statements of the transaction must be undone.

3.1.1 The Insert Statement

The INSERT statement adds new rows to a table. The syntax of the INSERT statement is;

INSERT INTO *table* [(*column* [, *column...*])]
VALUES *(value* [, *value...*]);

With this syntax, only one row is inserted at a time.

In the syntax:

table	is the name of the table
column	is the name of the column in the table to populate
value	is the corresponding value for the column

Because you can insert a new row that contains values for each column, the column list is not required in the INSERT clause. However, if you do not use the column list, the values must be listed according to the default order of the columns in the table and a value must be provided for each column. The following examples explain this.

Example: A new employee called Harry Higins just joined an organization. Write a query to insert the record of a new employee into the employees table assuming the employee has an id of 120, earns a salary of 15000, was employed 0n 6th April 2009 and belongs to department 20.

INPUT: Two queries can be written to achieve this

Query 1:

```
INSERT INTO employees values (120,’Harry’,’Haggins’, 15000,’06-APR-2009’,20)
```

Query 2:

```
INSERT INTO employees (employee_id,first_name,last_name,salary,hire_date,department_id)

Values(120,’Harry’,’Higgins’,15000,’06-APR-2009’,20);
```

OUTPUT:

The new table becomes:

Table 3.3(a): 1 row created.

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	HIRE_DATE	DEPARTMENT_ID
100	Steven	King	24000	17-JUN-87	90
101	Neena	Kochhar	17000	21-SEP-89	20
102	Lex	De Haan	17000	13-JAN-93	40
103	Alexander	Hunold	9000	03-JAN-90	60
104	Bruce	Ernst	6000	21-MAY-91	60
105	David	Austin	4800	25-JUN-97	50
106	Valli	Pataballa	4800	05-FEB-98	50
107	Diana	Lorentz	4200	07-FEB-99	60
108	Nancy	Greenberg	12000	17-AUG-94	70
109	Daniel	Faviet	9000	16-AUG-94	100
110	John	Chen	8200	28-SEP-97	100
111	Ismael	Sciarra	7700	30-SEP-97	70
112	Jose Manuel	Urman	7800	07-MAR-98	100
113	Luis	Popp	6900	07-DEC-99	100
114	Den	Raphaely	11000	07-DEC-94	30
115	Alexander	Khoo	3100	18-MAY-95	80
116	Shelli	Baida	2900	24-DEC-97	30
117	Sigal	Tobias	2800	24-JUL-97	80
118	Guy	Himuro	2600	15-NOV-98	30
119	Karen	Colmenares	2500	10-AUG-99	30
• 120	Harry	higgins	15000	06-APR-09	20

Note: The format of query 1 is applicable if the record to be inserted has all the required fields. The format of query 1 is applicable if the record to be inserted has some missing fields, fields must be respectively listed in the order in which the values are listed. The format of query 2 can also be used when all fields are required, but it is like reinventing the wheels, It wastes time.

3.1.2 The Update Statement

The UPDATE statement modifies existing rows in a table. The syntax of the UPDATE syntax is shown below;

UPDATE *table*
SET *column* = *value* [, *column* = *value, ...*]
[WHERE *condition*];

The UPDATE statement can update more than one row at a time (if required).

In the syntax:

table	is the name of the table
column	is the name of the column in the table to populate
value	is the corresponding value or subquery for the column
condition	identifies the rows to be updated and is composed of column names, expressions, constants, subqueries and comparison operators.

Example: Due to the recent redeployment in an Organization employee 113 has been transferred to department 70. Write a query to effect this.

INPUT:

```
UPDATE employees

SET department_id=70

WHERE employee_id=113;
```

OUTPUT:

Table 3.3(b): 1 row updated.

EMPLOYEE_ID	**FIRST_NAME**	**LAST_NAME**	**SALARY**	**HIRE_DATE**	**DEPARTMENT_ID**
100	Steven	King	24000	17-JUN-87	90
101	Neena	Kochhar	17000	21-SEP-89	20
102	Lex	De Haan	17000	13-JAN-93	40
103	Alexander	Hunold	9000	03-JAN-90	60
104	Bruce	Ernst	6000	21-MAY-91	60
105	David	Austin	4800	25-JUN-97	50
106	Valli	Pataballa	4800	05-FEB-98	50
107	Diana	Lorentz	4200	07-FEB-99	60
108	Nancy	Greenberg	12000	17-AUG-94	70
109	Daniel	Faviet	9000	16-AUG-94	100
110	John	Chen	8200	28-SEP-97	100
111	Ismael	Sciarra	7700	30-SEP-97	70
112	Jose Manuel	Urman	7800	07-MAR-98	100
113	Luis	Popp	6900	07-DEC-99	• **70**
114	Den	Raphaely	11000	07-DEC-94	30
115	Alexander	Khoo	3100	18-MAY-95	80
116	Shelli	Baida	2900	24-DEC-97	30
117	Sigal	Tobias	2800	24-JUL-97	80
118	Guy	Himuro	2600	15-NOV-98	30
119	Karen	Colmenares	2500	10-AUG-99	30
120	Harry	higgins	15000	06-APR-09	20

Note: All rows in a table are modified if the where clause is omitted in a UPDATE statement. This following query shows this;

Example: Write a query to change all employees' salary to 113

INPUT:

UPDATE employees

SET department_id=113;

OUTPUT:

New table becomes:

Table 3.3(c) : 22 rows updated.

EMPLOYEE_ID	**FIRST_NAME**	**LAST_NAME**	**SALARY**	**HIRE_DATE**	**DEPARTMENT_ID**
100	Steven	King	24000	17-JUN-87	113
101	Neena	Kochhar	17000	21-SEP-89	113
102	Lex	De Haan	17000	13-JAN-93	113
103	Alexander	Hunold	9000	03-JAN-90	113
104	Bruce	Ernst	6000	21-MAY-91	113
105	David	Austin	4800	25-JUN-97	113
106	Valli	Pataballa	4800	05-FEB-98	113
107	Diana	Lorentz	4200	07-FEB-99	113
108	Nancy	Greenberg	12000	17-AUG-94	113
109	Daniel	Faviet	9000	16-AUG-94	113
110	John	Chen	8200	28-SEP-97	113
111	Ismael	Sciarra	7700	30-SEP-97	113
112	Jose Manuel	Urman	7800	07-MAR-98	113
113	Luis	Popp	6900	07-DEC-99	113
114	Den	Raphaely	11000	07-DEC-94	113
115	Alexander	Khoo	3100	18-MAY-95	113
116	Shelli	Baida	2900	24-DEC-97	113
117	Sigal	Tobias	2800	24-JUL-97	113
118	Guy	Himuro	2600	15-NOV-98	113
119	Karen	Colmenares	2500	10-AUG-99	113
120	Harry	higgins	15000	06-APR-09	113

Multiple columns can be updated in the SET clause of an UPDATE statement by using the following syntax;

UPDATE *table*
SET *column1 = value, Column2=value, …, Columnn= value*
[WHERE *condition*];

An example of update of multiple columns is shown below;

Example: Write a query to update the salary and employee id of employee 114 respectively to 9000 and 130.

INPUT:

```
UPDATE employees
SET employee_id=130,
    salary=9000
where employee_id=114;
```

OUTPUT:

Table 3.3(d) : 1 row updated

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	HIRE_DATE	DEPARTMENT_ID
100	Steven	King	24000	17-JUN-87	90
101	Neena	Kochhar	17000	21-SEP-89	20
102	Lex	De Haan	17000	13-JAN-93	40
103	Alexander	Hunold	9000	03-JAN-90	60
104	Bruce	Ernst	6000	21-MAY-91	60
105	David	Austin	4800	25-JUN-97	50
106	Valli	Pataballa	4800	05-FEB-98	50
107	Diana	Lorentz	4200	07-FEB-99	60
108	Nancy	Greenberg	12000	17-AUG-94	70
109	Daniel	Faviet	9000	16-AUG-94	100
110	John	Chen	8200	28-SEP-97	100
111	Ismael	Sciarra	7700	30-SEP-97	70
112	Jose Manuel	Urman	7800	07-MAR-98	100
113	Luis	Popp	6900	07-DEC-99	100
• 130	Den	Raphaely	• 9000	07-DEC-94	30
115	Alexander	Khoo	3100	18-MAY-95	80
116	Shelli	Baida	2900	24-DEC-97	30
117	Sigal	Tobias	2800	24-JUL-97	80
118	Guy	Himuro	2600	15-NOV-98	30
119	Karen	Colmenares	2500	10-AUG-99	30
120	Harry	higgins	15000	06-APR-09	20

.

Note: In general, use the primary key to identify a single row. Using other columns can unexpectedly cause several rows to be updated. e.g Identifying a single row in the EMPLOYEES table by name is dangerous, because more than one employee may have the same name.

3.1.3 The Delete Statement

The DELETE statement allows the removal of existing rows from a table by using the DELETE statement. The syntax of the DELETE statement is shown below;

DELETE [FROM] ***table***
[WHERE ***condition*****];**

In the syntax,

table	is the table name
condition	identifies the rows to be deleted and is composed of column names, expressions, constants, subqueries and comparison operators.

The FROM clause is written in square brackets in the syntax to show that it is optional.

The WHERE *condition* is optional. Specific rows are deleted if the WHERE condition is specified, otherwise, all the rows in the table are deleted

Depending on the use of the DELETE statement's WHERE clause, SQL can do the following:

- Delete single rows
- Delete multiple rows
- Delete all rows
- Delete no rows

Here are several points to remember when using the DELETE statement:

- The DELETE statement cannot delete an individual field's values (use UPDATE instead). The DELETE statement deletes entire records from a single table.
- Using the DELETE statement deletes only records, not the table itself.

Note: If no rows are deleted, the message "0 rows deleted" is returned.

Example: The following query removes all the rows from the employees table

INPUT:

```
DELETE EMPLOYEES
```

OUTPUT:

```
20 rows deleted.
```

It can also be written as DELETE FROM EMPLOYEES

Example: Write a query to delete the record of employee 102 record from the employees table.

INPUT:

```
DELETE FROM EMPLOYEES
WHERE EMPLOYEE_ID=102;
```

OUTPUT:

Table 3.3(e) : 1 row deleted.

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	HIRE_DATE	DEPARTMENT_ID
100	Steven	King	24000	17-JUN-87	90
101	Neena	Kochhar	17000	21-SEP-89	20
103	Alexander	Hunold	9000	03-JAN-90	60
104	Bruce	Ernst	6000	21-MAY-91	60
105	David	Austin	4800	25-JUN-97	50
106	Valli	Pataballa	4800	05-FEB-98	50
107	Diana	Lorentz	4200	07-FEB-99	60
108	Nancy	Greenberg	12000	17-AUG-94	70
109	Daniel	Faviet	9000	16-AUG-94	100
110	John	Chen	8200	28-SEP-97	100
111	Ismael	Sciarra	7700	30-SEP-97	70
112	Jose Manuel	Urman	7800	07-MAR-98	100
113	Luis	Popp	6900	07-DEC-99	100
114	Den	Raphaely	11000	07-DEC-94	30
115	Alexander	Khoo	3100	18-MAY-95	80
116	Shelli	Baida	2900	24-DEC-97	30
117	Sigal	Tobias	2800	24-JUL-97	80
118	Guy	Himuro	2600	15-NOV-98	30
119	Karen	Colmenares	2500	10-AUG-99	30
120	Harry	higgins	15000	06-APR-09	20

4.0 CONCLUSION

SQL provides three statements that you can use to manipulate data within a database.

The INSERT...VALUES statement inserts a set of values into one record. The UPDATE statement changes the values of one or more columns based on some condition. This updated value can also be the result of an expression or calculation.

The DELETE statement is the simplest of the three statements. It deletes all rows from a table based on the result of an optional WHERE clause. If the WHERE clause is omitted, all records from the table are deleted.

5.0 SUMMARY

In this unit, you should have learnt how to:

- Insert records into a table
- Update records in a table (single column and multiple columns)
- Delete records from a table(all records and specific record)

6.0 TUTOR MARKED ASSIGNMENT

1. Five employees just joined an organization, their records are shown in the table below; write queries to insert the records into the employees table.

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	HIRE_DATE	DEPARTMENT_ID
131	Omotoso	Grace	30000	10-JAN-08	20
132	Adekola	Charles	32000	20-JAN-08	20
133	Adesola	Judas	35000	30-JAN-08	30
134	Ogunjimi	Ayodeji	38000	02-FEB-09	40
135	Ogunbiyi	Doyin	40000	13-MAR-09	50

Figure 3.3(a): A table showing the records of 5 new employees

2. Due to some rescheduling in the Organization, the human resources manager asked employees 131 and 135 to be transferred to department 60 and 70 respectively and have their salary increased respectively to 45000 and 50000. Write a query to update the records.
3. It was discovered that employee 132 and 133 were not qualified for their posts due to fake documents tendered during their interview. Thus, they are being sacked from the Organization. Write a query to remove their records from the employees table

7.0 FURTHER READING AND OTHER RESOURCES

1. Oracle Database 10g SQL Fundamentals I
2. Teach Yourself SQL in 24 Hours (Ryan Stephens/Ron Plew)
3. Teach Yourself SQL in 21 Days, 2nd Edition (Ron Plew/ Ryan Stephens)

MODULE 3 - LABORATORY EXERCISES USING SQL

UNIT 4 - CREATING DATABASE OBJECTS

Content Page

1.0 INTRODUCTION

Database objects are the underlying backbone of the relational database. They help in storing data in various forms This unit examines these forms and gives some practical exercises to guide the students.

2.0 OBJECTIVES

After completing this unit, you should be able to do the following;

i. Identify some database Objects
ii. Create tables
iii. Create Views
iv. Create Sequences

3.0 INTRODUCTION TO DATABASE OBJECTS

Database objects are the underlying backbone of the relational database. These *objects* are logical units within the database that are used to store information, and are referred to as the *back-end database.* A *schema* is a collection of database objects associated with one particular database username. This username is called the *schema owner,* or the owner of the related group of objects. You may have one or multiple schemas in a database. Basically, any user who creates an object has just created his or her own schema. So, based on a user's privileges within the database, the user has control over objects that are created, manipulated, and deleted.

3.1 DATABASE OBJECTS

The table below shows some database objects that are commonly used.

Table 3.4(a): A table showing a list of some database objects

Object	**Description**
Table	Basic unit of storage composed of rows
View	Logically represents subsets of data from one or more tables
Sequences	Generates numeric values
Index	Improves the performance of some queries
Synonym	Gives alternative names to objects

In this Unit, tables, views and sequences will be considered.

3.1.1 Tables

Tables are basic database objects used for storing data. The syntax for creating tables is shown below:

CREATE TABLE [*schema*.]*table*
(*column datatype* [DEFAULT *expr*][, ...]);

In the syntax,

Schema	is the same as the owner of the name
table	is the name of the table
DEFAULT *expr*	specifies a default value if a value is omitted in the INSERT statement
column	is the name of the column
datatype	is the column's data type and length

Example: To create a table that contains students' records, we can have the following query:

INPUT:

```
CREATE TABLE STUDENTS(

MATRIC_NUM CHAR(20),

FIRST_NAME CHAR(30),

LAST_NAME CHAR(30),

AGE NUMBER(5),

DEPARTMENT VARCHAR(20));
```

OUTPUT:

```
Table created
```

This statement creates a table named STUDENTS. Within the STUDENTS table are four fields: MATRIC NUMBER, FIRST NAME, LAST NAME, AGE and DEPARTMENT NAME.

The matric number field has a data type of characters and can store up to 20 characters long.

The FIRST NAME and LAST NAME field has a data type of character and can store strings up to 30 characters long.

The AGE field has a data type of numbers and can store numbers that are not more than 5 digits, the DEPARTMENT field has a data type of characters and can store up to 20 characters long.

After creating tables, the structure of the table can be viewed by using the DESCRIBE statement. The syntax of the DESCRIBE statement is shown below;

DESCRIBE *table_name;*

Example: To display the structure of the STUDENTS table, we have

INPUT:

```
DESCRIBE STUDENTS;
```

OUTPUT:

Table 3.4(b): A table showing the structure of STUDENTS table

Name	Null?	Type
MATRIC_NUM		CHAR(20)
FIRST_NAME		CHAR(30)
LAST_NAME		CHAR(30)
AGE		NUMBER(5)
DEPARTMENT		VARCHAR2(20)

When tables are no longer needed, they can be dropped by using the DROP TABLE statement.

The syntax of the DROP TABLE syntax is shown below:

DROP TABLE *table_name;*

In the syntax, table_name is the name of the table to be dropped.

Example: To drop the STUDENTS table that was created in the last example, we have the following query;

INPUT:

DROP TABLE STUDENTS;

OUTPUT:

Table dropped.

3.1.2 VIEWS

A view is a logical table based on a table or another view. A view contains no data of its own but is like a window through which data from tables can be viewed or changed. The table on which a view is based is called base tables. The advantages of views are listed below;

- Views restrict access to the data because the view can display selected columns from the table
- Views can be used to make simple queries to retrieve the results of complicated queries. For example, views can be used to query information from multiple tables without the user knowing how to write a join statement.
- Views provide data independence for ad hoc users and application programs. One view can be used to retrieve data from several tables.
- Views provide groups of users access to data according to their particular criteria.

The syntax for creating a view is shown overleaf:

CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW *view*
[(*alias*[, *alias*]...)]
AS *subquery*
[WITH CHECK OPTION [CONSTRAINT *constraint*]]
[WITH READ ONLY [CONSTRAINT *constraint*]];

Note: The statements in square brackets are optional.

In the syntax:

OR REPLACE	recreates the view if it already exists
FORCE	creates the view regardless of whether or not the base tables exist
NOFORCE	creates the view only if the base tables exist (This is the default.)
View	specifies names for the expressions selected by the view's query
Subquery	is a complete SELECT statement

WITH CHECK OPTION specifies that only those rows that are accessible to the view can be inserted or updated

Constraint	is the name assigned to the CHECK OPTION constraint
WITH READ ONLY	ensures that no DML operations can be performed on this view.

Example: Write a query to create a view that contains records of employees in department 20. The view should be called dept20.

Solution:

INPUT:

```
Create view dept20

AS SELECT * from EMPLOYEES

WHERE department_id=20;
```

OUTPUT:

```
View created.
```

Just like tables, the structure of views can be displayed by using the DESCRIBE statement. For example, to display the structure of the dept20, we have;

INPUT:

```
DESCRIBE dept20;
```

OUTPUT:

Table 3.4(c) : Table created using DESCRIBE.

Name	**Null?**	**Type**
EMPLOYEE_ID		NUMBER(6)
FIRST_NAME		VARCHAR2(20)
LAST_NAME	NOT NULL	VARCHAR2(25)
SALARY		NUMBER(8,2)
HIRE_DATE	NOT NULL	DATE
DEPARTMENT_ID		NUMBER(4)

3.1.3 SEQUENCES

A sequence is a database object that creates integer values. Sequences can be created and later used to generate numbers. A sequence is a user-created database object that can be shared by multiple users to generate integers. Sequences can be defined to generate unique values or to recycle and use the numbers again. A typical usage for sequences is to create a primary key value, which must be unique for each row. This sequence is generated and incremented (or decremented) by an internal database routine. This can be a time saving routine because it can reduce the amount of application code needed to write a sequence-generating routine.

The syntax for creating a sequence is as shown below;

```
CREATE SEQUENCE sequence
[INCREMENT BY n]
[START WITH n]
[{MAXVALUE n | NOMAXVALUE}]
[{MINVALUE n | NOMINVALUE}]
[{CYCLE | NOCYCLE}]
[{CACHE n | NOCACHE}];
```

In the syntax:

Sequence is the name of the sequence generator

INCREMENT BY *n* specifies the interval between sequence numbers, where n is an integer. If the clause is omitted, the sequence is incremented by 1;

START WITH *n* specifies the first sequence number to be generated(If this clause is omitted, the sequence starts with 1)

MAXVALUE *n* specifies the maximum value to be generated. (If this clause is omitted,the sequence starts with 1.)

NOMAXVALUE specifies a maximum value of 10^27 for an ascending sequence and -1 for a descending sequence. (This is the default option).

MINVALUE *n* specifies the minimum sequence value

NOMINVALUE specifies a minimum value of 1 for an ascending sequence and -(10^26) for a descending sequence (This is the default option).

CYCLE|NOCYCLE specifies whether the sequence continues to generate values after reaching its maximum or minimum value. (NO CYCLE is the default option).

CACHE *n*| NO CACHE specifies how many values the database server preallocates and keeps in memory.

Example 3.4(a): To create a sequence that generates serial numbers from 1 to 100, we can us the following query;

INPUT:

```
CREATE SEQUENCE serial_number
INCREMENT BY 1
START WITH 1
MAXVALUE 100
NOCACHE
NOCYCLE;
```

OUTPUT:

```
Sequence created.
```

After a sequence is created, it generates sequential numbers for use in your tables. Reference the sequence values by using the NEXTVAL and CURRVAL pseudocolumns.

The NEXTVAL pseudocolumn is used to extract successive sequence numbers from a specified sequence. NEXTVAL must be qualified with the sequence name. When you reference *sequence.NEXTVAL,* a new sequence number is generated and the current sequence number is placed in CURRVAL.

The CURRVAL pseudocolumn is used to refer to a sequence number that the current user has just generated. NEXTVAL must be used to generate a sequence number in the current user’s session before CURRVAL can be referenced. CURRVAL must also be qualified with the sequence name. When sequence.CURRVAL is referenced, the last value returned to that user’s process is displayed.

4.0 CONCLUSION

Database objects helps in storing data in various forms. Tables are the basic database objects used for storing data. Views are windows through which the records of a table can be viewed. Sequences are basically used for generating sequential numbers. There are other database objects like indexes and synonyms which are respectively used for fast data retrieval and renaming database objects.

5.0 SUMMARY

In this Unit, you should have learnt how to do the following;

- Identify some database objects
- Create tables, describe the structure of table and drop tables
- Create sequences

6.0 TUTOR MARKED ASSIGNMENT

1. Write a query to create a table called PRACTICE whose structure is shown below:

Column Name	ID	First Name	Last Name	Dept ID
Data Type	Number	Char	Char	Number
Length	7	20	20	5

2. Insert records displayed in the table below into your table

ID	First Name	Last Name	Dept ID
200	Ade	Olu	20
210	Ola	Jide	30
220	Oye	Sola	40

3. Create a view called PRACTICE_VIEW with the PRACTICE table acting as a base table. The view should contain all rows with fields last name and First name.
    a. View the structure of PRACTICE_VIEW
    b. Drop the PRACTICE_VIEW view.
    c. Drop the PRACTICE TABLE

7.0 FURTHER READING AND OTHER RESOURCES

1. Oracle Database 10g SQL Fundamentals I
2. Teach Yourself SQL in 24 Hours by Ryan Stephens & Ron Plew
3. Teach Yourself SQL in 21 Days, 2nd Edition by Ron Plew & Ryan Stephens

MODULE 3 - LABORATORY EXERCISES USING SQL

UNIT 5 - AGGREGATE FUNCTION

Content Page

1.0 INTRODUCTION

Aggregate functions operate on sets of rows to give one result per group. SQL Aggregate functions return a single value, using values in a table column. The SQL COUNT function returns the number of rows in a table satisfying the criteria specified in the WHERE clause. The SQL AVG function retrieves the average value for a numeric column. This unit examines all these and provides adequate examples to enhance understanding.

2.0 OBJECTIVES

After completing this unit, you should be able to do the following;

- Identify the available group functions
- Describe the use of group some functions

3.0 INTRODUCTION TO AGGREGATE FUNCTIONS

Aggregate functions operate on sets of rows to give one result per group. These functions greatly increase your ability to manipulate the information retrieved from an SQL query. Aggregate functions are also called group functions.

3.1 AGGREGATE FUNCTIONS

There are various aggregate functions that act on the result of an SQL query. These functions are shown in the table below:

Table 3.5(a): A table showing a list of aggregate functions

Function	**Description**
AVG(*n*)	Average value of n, ignoring null values
COUNT(* | [DISTINCT] | *expr*)	Number of rows, where *expr* evaluates to something other than null
MAX(*expr*)	Maximum value or expr, ignoring null values
MIN(*expr*)	Minimum value of expr, ignoring null values
STDDEV(*n*)	Standard deviation of n, ignoring null values
SUM(*n*)	Sum values of n, ignoring null values
VARIANCE(*n*)	Variance of n, ignoring null values

In this unit, the MAX(), MIN(), SUM(), AVG() and COUNT() functions will be discussed.

3.1.1 The MIN() And MAX() Function

The MIN() function is used to find the least value in a column. For example to find the least salary earned by an employee. We use the query

INPUT:

SELECT MIN(salary)

FROM employees;

OUTPUT:

Table 3.5(b): The minimum salary earned by an employee

MIN(SALARY)
2500

The MAX() function is used to find the least value in a column. For example to find the highest salary earned by an employee. We use the query

INPUT:

SELECT MAX(salary)

FROM employees;

OUTPUT:

Table 3.5(c): The maximum salary earned by an employee

MAX(SALARY)
24000

3.1.2 The SUM() and AVG() Function

The SUM() function sums the values in a column. For example, to find the total salary paid to employees, we use the query;

INPUT:

SELECT SUM(salary)

FROM employees;

OUTPUT:

Table 3.5(d): The total salary earned by employees

SUM(SALARY)
163300

The AVG() function returns the average of values in a column. For example, to find the average salary paid to employees, we use the query;

INPUT:

```
SELECT AVG(salary)

FROM employees;
```

OUTPUT:

Table 3.5(e): The average salary earned by employees

AVG(SALARY)
8165

3.1.3 The COUNT() Function

The COUNT function has three formats

- COUNT(*)
- COUNT(expr)
- COUNT(DISTINCT expr)

COUNT(*) returns the number of rows in a table that satisfy the criteria of the SELECT statement, including duplicate rows and rows containing null values in any of the columns. If a WHERE clause is included in the SELECT statement, COUNT(*) returns the number of rows that satisfy the condition in the WHERE c;lause. For example, To write a query to count the number of records in the employees table we have;

INPUT:

```
SELECT count(*)

FROM employees;
```

OUTPUT:

Table 3.5(f): A table showing the number of records in the employees table

COUNT(*)
20

In contrast, COUNT(expr) returns the number of non-null values that are in the column identified by expr. For example, to count the number of department_id s having the value of 20, we use the query

INPUT:

```
SELECT COUNT(department_id)

FROM employees

WHERE department_id=20;
```

OUTPUT:

COUNT(DEPARTMENT_ID)
2

COUNT(DISTINT expr) returns the number of unique, non-null values that are in the column identified by expr. For example, To count the number of unique department_id in the employees table, we have;

INPUT:

```
SELECT COUNT(DISTINCT department_id)

FROM employees;
```

OUTPUT:

COUNT(DISTINCTDEPARTMENT_ID)
9

4.0 CONCLUSION

Aggregate functions are used in grouping columns of data as an entity. Decisions can be made based on the values returned by the functions and a great deal of time can be saved through the use of aggregate functions especially where large records are involved.

5.0 SUMMARY

In this Unit, you should have learnt the following:

- Identifying some aggregate functions
- Using group functions

6.0 TUTOR MARKED ASSIGNMENT

Create the table shown next page and insert records into the table. The table should be named EMP

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	HIRE_DATE	DEPARTMENT_ID
100	Steven	King	24000	17-JUN-87	90
101	Neena	Kochhar	17000	21-SEP-89	20
102	Lex	De Haan	17000	13-JAN-93	40
103	Alexander	Hunold	9000	03-JAN-90	60
104	Bruce	Ernst	6000	21-MAY-91	60
105	David	Austin	4800	25-JUN-97	50
106	Valli	Pataballa	4800	05-FEB-98	50
107	Diana	Lorentz	4200	07-FEB-99	60
108	Nancy	Greenberg	12000	17-AUG-94	70
109	Daniel	Faviet	9000	16-AUG-94	100
110	John	Chen	8200	28-SEP-97	100
111	Ismael	Sciarra	7700	30-SEP-97	70
112	Jose Manuel	Urman	7800	07-MAR-98	100
113	Luis	Popp	6900	07-DEC-99	100
114	Den	Raphaely	11000	07-DEC-94	30
115	Alexander	Khoo	3100	18-MAY-95	80
116	Shelli	Baida	2900	24-DEC-97	30
117	Sigal	Tobias	2800	24-JUL-97	80
118	Guy	Himuro	2600	15-NOV-98	30
119	Karen	Colmenares	2500	10-AUG-99	30

Using the table shown above, answer the following questions:

1. Write a query to calculate the average salary of employees in department 100
2. Write a query to count the number of people earning 17000
3. Write a query to count the number of distinct department an employee can belong to
4. Write a query to calculate the total salary earned by employees in department 100
5. Write a query to display the least salary in department 100

7.0 FURTHER READING AND OTHER RESOURCES

1. Oracle Database 10g SQL Fundamentals I
2. Teach Yourself SQL in 24 Hours by Ryan Stephens & Ron Plew
3. Teach Yourself SQL in 21 Days, 2nd Edition by Ron Plew & Ryan Stephens

MODULE 4 -   USING ADA AND COBOL

UNIT 1 – ADA Programming Language

Content Page

1.0 INTRODUCTION

Ada is a programming language named after Augusta Ada King; it has built-in features that directly support structured, object-oriented, generic, distributed and concurrent programming. Ada puts unique emphasis on, and provides strong support for, good software engineering practices that scale well to very large software systems. It should come as no surprise that Ada is heavily used in the aerospace, defense, medical, railroad,
and nuclear industries. Ada has the following language features:

- Allows the programmer to construct powerful abstractions that reflect the real world, and allows the compiler to detect many logic errors before they become bugs.
- Modularity
- Information hiding; the language separates interfaces from implementation, and provides fine-grained control over visibility.
- Readability, which helps programmers review and verify code. Ada favours the reader of the program over the writer, because a program is written once but read many.
- Portability: the language definition allows compilers to differ only in a few controlled ways, and otherwise defines the semantics of programs very precisely.

Other features include restrictions (it is possible to restrict which language features are accepted in a program) and features that help review and certify the object code generated by the compiler; it has powerful specialized features supporting low-level programming for real-time. Ada shines even more in software maintenance, which often accounts for 80% of the total cost of development.

2.0 OBJECTIVES

On completing this unit, you would be able to:

- Create and write an Ada source code
- Understand Control Structure in Ada
- Understand I/Os in Ada
- Create and Understand Packages
- Understand Arrays in Ada

3.0 FIRST ADA PROGRAM

The "Hello, world!" is going to be our first program to practice with before going further.

A common example of a language's syntax is the Hello world program. Here a straight-forward Implementation:

File: hello_world_1.adb

```
with Ada.Text_IO;
procedure Hello is
begin
Ada.Text_IO.Put_Line("Hello, world!");
end Hello;
```

The **with** statement adds the package Ada.Text_IO to the program. This package comes with every Ada compiler and contains all functionality needed for textual Input/Output. The **with** statement makes the declarations of Ada.Text_IO available to procedure Hello. This includes the types declared in Ada.Text_IO, the subprograms of Ada.Text_IO and everything else that is declared in Ada.Text_IO for public use. In Ada, packages can be used as toolboxes. Text_IO provides a collection of tools for textual input and output in one easy-to-access module.

KEYWORDS OF ADA PROGRAMMING

Ada **keywords** have different functions depending on where they are used. A good example is **for** which is used in a control structure. A keyword is also a reserved word, so it cannot be used as an identifier. The following are examples of popular keywords in Ada;
Else, new, return, abs, elsif, not, reverse, abstract, end, null, accept, entry, and, for, out, array, function, overriding, tagged, at, package, terminate, begin, goto, pragma, then, body, private, type, if, procedure, case, in, protected, until, constant, interface, use, record, while, delta, loop, rem, with, digits …

3.1 I/O LIBRARY, STATEMENTS AND VARIABLES

3.1.1 I/O (Input / Output) LIBRARY

There are 5 independent libraries for input/output operations in Ada and they are;

1. **Text I/O: This is** probably the most used Input/Output library. Text I/O provides support for line and page layout but the standard is free form text. The general form is Ada.Text_IO;
2. **Direct I/O: This is** used for random access files which contain only elements of one specific type. The general form is Ada.Direct_IO;
3. **Sequential I/O: This is** used for random access files which contain elements of more than one specific type. You have to read and write the elements one after the other. It just the opposite to Direct I/O. The general form is Ada.Sequential_IO;
4. **Storage I/O: This** allows you to store *one* element inside a memory buffer. Storage I/O is useful in Concurrent programming where it can be used to move elements from one task to another.
5. **Stream I/O: This** is the most powerful input/output library in Ada, it allows you to mix objects from different element types in one sequential file. In order to read/write from/to a streameach type provides a 'Read and 'Write attribute as well as an 'Input and 'Output attribute.

3.1.2 STATEMENTS

Most programming languages have the concept of a statement. A ***statement*** is a command that the programmer gives to the computer. For example:
Ada.Text_IO.Put_Line ("Hi there!");
This command has a verb “Put_Line". In this case, the command "Put_Line" means “show on the screen," not "print on the printer." The programmer either gives the statement directly to the computer (by typing it while running a special program), or creates a text file with the command in it then run it. If you have more than one command in the file, each will be performed in the same order.

So the file could contain:

```
Ada.Text_IO.Put_Line (“Hi there!");
Ada.Text_IO.Put_Line (“Strange things are afoot...");
```

Ada allows you to declare shorter aliasnames if you need a long statement very often. In the above case, the computer will look at the first statement, determine that it's a Put_Line statement, look at what needs to be printed, and display that text on the computer screen. It'll look like this:

```
Hi there!
```

Note that the quotation marks aren't there. Their purpose in the program is to tell the computer where the text begins and ends, just like in English prose. The computer will then continue to the next statement, perform its command, and the screen will look like this:

```
Hi there!
Strange things are afoot...
```

When the computer gets to the end of the text file, it stops. There are many different kinds of statements, depending on which programming language is being used. For example, there could be a *beep* statement that causes the computer to output a beep on its speaker, or a *window* statement that causes a new window to pop up.

Also, the way statements are written will vary depending on the programming language. These differences are fairly superficial. The set of rules like the first two is called a programming language's ***syntax***.

3.1.3 VARIABLES

Variables are *references* that stand in for a *value* that is contained at a certain memory address. Variables store everything in your program. Variables are said to have a value and *may* have a data type. If a variable has a type, then only values of this type may be assigned to it. An *assignment statement* is used to set a variable to a new value. Assignment statements are written as:

name **:=** *value*

```
X := 10;
```

The example assigns the variable *X* to the integer value of *10*. The assignment statement overwrites the contents of the variable and the previous value is lost. In some languages, before a variable can be used, it will have to be declared, where the declaration specifies the type so also in Ada. The declaration is as follows:

```
declare
X : Integer := 10;
begin
Do_Something (X);
end;
```

3.2 CHARACTER SETS, STRINGS, DELIMITERS AND COMMENT

3.2.1 CHARACTER SET

The character set used in Ada programs is composed of:

• Upper-case letters: A, ..., Z and lower-case letters: a, ..., z.
• Digits: 0, ..., 9.
• Special characters.

Take into account that in Ada the letter range includes accented characters and other letters used in Western Europe languages such as c, n, d, etc. The identifiers and comments can be written in almost any language in the world. Ada is a case-insensitive language, i.e. the upper-case set is equivalent to the lower-case set. Not only does Ada 2005 now support a new 32-bit character type — called "Wide_Wide_Character" but the source code itself may be of this extended character set as well. Thus Russians and Indians, for example, will be able to use their native language in identifiers and comments. And mathematicians will rejoice: The whole Greek and fractur character sets are available for identifiers.

3.2.2 STRINGS

Ada supports three different types of strings and they are designed to solve a different problem.Every string type is implemented for each available Characters type (Character, Wide_Character, Wide_Wide_Character) giving a complement of nine combinations.

i. Fixed-length string handling

Fixed-Length Strings are arrays of Character, and of a fixed length. Since a fixed length string is an indefinite data type the length does not need to be known at compile time - the length may well be calculated at run time. In the following example the length is calculated from command-line argument 1:

```
X : String := Ada.Command_Line.Argument (1);
```

However once the length has been calculated and the strings have been created the length stays constant. Fixed-Length String Handling which allows padding of shorter strings and truncation of longer strings.

Exercise: Try the following example to see how it works:

File: show_commandline_2.adb

```
with Ada.Text_IO;
with Ada.Command_Line;
with Ada.Strings.Fixed;
procedure Show_Commandline_2 is
package T_IO renames Ada.Text_IO;
package CL renames Ada.Command_Line;
package S renames Ada.Strings;
package SF renames Ada.Strings.Fixed;
X : String := CL.Argument (1);
begin
T_IO.Put ("Argument 1 = ");
T_IO.Put_Line (X);
SF.Move (
Source => CL.Argument (2),
Target => X,
Drop => S.Right,
Justify => S.Left,
Pad => S.Space);
T_IO.Put ("Argument 2 = ");
T_IO.Put_Line (X);
end Show_Commandline_2;
```

ii. Bounded-length string handling

Bounded-Length Strings can be used when the maximum length of a string is known and/or restricted. This is often the case in database applications where only a limited amount of characters can be stored.

Like Fixed-Length Strings the maximum length does not need to be known at compile time - it can also be calculated at runtime - as the example below shows:

File: show_commandline_3.adb

```
with Ada.Text_IO;
with Ada.Command_Line;
with Ada.Strings.Bounded;
procedure Show_Commandline_3 is
package T_IO renames Ada.Text_IO;
package CL renames Ada.Command_Line;
function Max_Length (Value_1 : Integer;Value_2 : Integer)
return
Integer
is
Retval : Integer;
begin
if Value_1 > Value_2 then
Retval := Value_1;
else
Retval := Value_2;
end if;
return Retval;
end Max_Length;
pragma Inline (Max_Length);
package SB
is new Ada.Strings.Bounded.Generic_Bounded_Length (Max => Max_Length (Value_1 => CL.Argument (1)'Length,Value_2 => CL.Argument (2)'Length));
X : SB.Bounded_String
:= SB.To_Bounded_String (CL.Argument (1));
begin
T_IO.Put ("Argument 1 = ");
T_IO.Put_Line (SB.To_String (X));
X := SB.To_Bounded_String (CL.Argument (2));
T_IO.Put ("Argument 2 = ");
T_IO.Put_Line (SB.To_String (X));
end Show_Commandline_3;
```

You should know that Bounded-Length Strings have some distinct disadvantages. Most noticeable is that each Bounded-Length String is a different type which makes converting them rather cumbersome.

Also a Bounded-Length String type always allocates memory for the maximum permitted string length for the type. The memory allocation for a Bounded-Length String is equal to the maximum number of string “characters" plus an implementation dependent number containing the string length (each character can require allocation of more than one byte per character, depending on the underlying character type of the string).

iii. Unbounded-length string handling

Last but not least there is the Unbounded-Length String. If you are not doing embedded or database programming this will be the string type you are going to use most often as it gives you the maximum amount of flexibility.

As the name suggest the Unbounded-Length String can hold strings of almost any length – limited only to the value of Integer Last or your available heap memory.

File: show_commandline_4.adb

```
with Ada.Text_IO;
with Ada.Command_Line;
with Ada.Strings.Unbounded;
procedure Show_Commandline_4 is
package T_IO renames Ada.Text_IO;
package CL renames Ada.Command_Line;
package SU renames Ada.Strings.Unbounded;
X : SU.Unbounded_String
:= SU.To_Unbounded_String (CL.Argument (1));
begin
T_IO.Put ("Argument 1 = ");
T_IO.Put_Line (SU.To_String (X));
X := SU.To_Unbounded_String (CL.Argument (2));
T_IO.Put ("Argument 2 = ");
T_IO.Put_Line (SU.To_String (X));
end Show_Commandline_4;
```

As you can see the Unbounded-Length String example is also the shortest (discarding the first example) - this makes using Unbounded-Length Strings very appealing.

3.2.3 DELIMITERS

A **Delimiter** is a character or space marking the beginning or end of a data element. In Ada, there are two categories of delimiters and they include the following with examples under them.

i. Single character delimiters

& ampersand (also operator &)
' apostrophe, tick
(left parenthesis
) right parenthesis
* asterisk, multiply (also operator *)
+ plus sign (also operator +)
, comma
- hyphen, minus (also operator -)
. full stop, point, dot
/ solidus, divide (also operator /)
: colon
; semicolon
< less than sign (also operator)

= equal sign (also operator =)
> greater than sign (also operator)
| vertical line

ii. Compound character delimiters

=> arrow
.. double dot
** double star, exponentiate (also operator **)
:= assignment
/= inequality (also operator)
>= greater than or equal to (also operator)
<= less than or equal to (also operator)
<< left label bracket
>> right label bracket
<> box
The following special characters are reserved but currently unused in Ada:
[left square bracket
] right square bracket
{ left curly bracket
} right curly bracket

The following ones are special characters but not delimiters:
" quotation mark, used in string literals.
# number sign, used in number literals with base.

3.2.4 COMMENTS

Comments in Ada start with two consecutive hyphens (--) and end in the end of line. For example:

```
-- This is a comment in a full line
My_Savings := My_Savings * 10.0; -- This is a comment in a line after a sentece
My_Savings := My_Savings * -- This is a comment inserted inside a sentence
1_000_000.0;
```

A comment can appear where an end of line can be inserted. It can also appear in the middle of a statement or a line before the statement. The comment makes the program readable and shows the meaning and functions of some lines of code. It is advisable that when written the program you insert a comment so the when someone else who is not the code author can understand what the program does.

3.3 PROCEDURE AND FUNCTION

A procedures call is a statement and does not return any value, whereas a function returns a value and must therefore be a part of an expression.

3.3.1 PROCEDURE

Procedure is a category of subprograms in Ada.A procedure call in Ada constitutes a statement by itself.
For example:

```
procedure A_Test (A, B: in Integer; C: out Integer) is
begin
```

```
 C := A + B;
end A_Test;
```

When the procedure is called with the statement A_Test (5 + P, 48, Q);
the expressions 5 + P and 48 are evaluated (expressions are only allowed for in parameters), and then assigned to the formal parameters A and B, which behave like constants. Then, the value A + B is assigned to formal variable C, whose value will be assigned to the actual parameter Q when the procedure finishes. C, being an **out** parameter, is an unitialized variable before the first assignment. Within a procedure, the return statement can be used without arguments to exit the procedure and return the control to the caller.
For example, to solve an equation of the kind of a Quadratic equation :

```
 with Ada.Numerics.Elementary_Functions;
 use Ada.Numerics.Elementary_Functions;
 procedure Quadratic_Equation
(A, B, C : Float; -- By default it is "in". R1, R2 : out Float; Valid : out Boolean)
 is
 Z : Float;
 begin
 Z := B**2 - 4.0 * A * C;
 if Z < 0.0 or A = 0.0 then
 Valid := False; -- Being out parameter, it must be modified at least once.
 R1 := 0.0;
 R2 := 0.0;
 else
 Valid := True;
  R1 := (-B + Sqrt (Z)) / (2.0 * A);
  R2 := (-B - Sqrt (Z)) / (2.0 * A);
  end if;
  end Quadratic_Equation;
```

The function SQRT calculates the square root of non-negative values. If the roots are real, they are given back in R1 and R2, but if they are complex or the equation degenerates (A = 0), the execution of the procedure finishes after assigning to the Valid variable the False value, so that it is controlled after the call to the procedure

3.3.2 FUNCTIONS

A function is a subprogram that can be invoked as part of an expression. Functions can only take **in** (the default) or **access** parameters. In this sense, Ada functions behave more like mathematical functions than in other languages. The specification of the function is necessary to show to the clients all the information needed to invoke it.
A function body example can be:

```
function Minimum (A, B : Integer) return Integer is
begin
if A <= B then
return A;
else
return B;
```

```
end if;
end Minimum;
```

The formal parameters of a function behave as local constants whose values are provided by the corresponding actual parameters. The statement **return** is used to indicate the value returned by the function call and to give back the control to the expression that called the function. The expression of the
return statement must be of the same type declared in the specification. If an incompatible type is used, the compiler gives an error. The body of the function can contain several **return** statements and the execution of any of them will finish the function, returning control to the caller. If the flow of control within the function branches in several ways, it is necessary to make sure that each one of them is finished with a **return** statement. If at run time the end of a function is reached without encountering a **return** statement, the exception Program_Error is raised. Therefore, the body of a function must have at least one such **return** statement.
When the function finalizes, its objects disappear. Therefore, it is possible to call the function recursively. For example, consider this implementation of the factorial function:

```
function Factorial (N : Positive) return Positive is
begin
if N = 1 then
return 1;
else
return (N * Factorial (N - 1));
end if;
end Factorial;
```

When evaluating the expression Factorial (4); the function will be called with parameter 4 and within the function it will try to evaluate the expression Factorial (3), calling itself as a function, but in this case parameter N would be 3 (each call copies the parameters) and so on until N = 1 is evaluated which will finalize the recursion and then the expression will begin to be completed in the reverse order.

3.4 PACKAGES IN ADA

Ada programming languages has well defined system of modularization and separate compilation. Even though Ada allows separate compilation, it maintains the strong type checking among the various compilations by enforcing rules of compilation order and compatibility checking. Packages are program units that allow the specification of groups of logically related entities. Typically, a package contains the declaration of a type along with the declaration of primitive subprograms (procedures or functions) of the type, which can be called from outside the package, while their inner workings remain hidden from outside users.

3.4.1 PARTS OF PACKAGES

There are 3 parts of a package but only the specification is necessary; the public package specification, private package specification and the package body. You can leave out the body and the private part if you don't need them.

I. The public package specification

This package specification of an Ada package describes all the subprogram specifications, variables, types, and constants etc that are visible to anyone who wishes to use the package.

```
package Public_Only_Package is
type Range_10 is range 1 .. 10;
end Public_Only_Package;
```

II. The private package specification

The private part of a package has two main jobs:

• To complete the defered definition of private types and constants.

• To export entities only visible to the children of the package

```
package Package_With_Private is
type Private_Type is private;
private
type Private_Type is array (1 .. 10) of Integer;
end Package_With_Private;
```

III. The package body

The package body defines the implementation of the package. All the subprograms defined in the specification have to be implemented in the body. New subprograms, types and objects can be defined in the body that is not visible to the users of the package.

```
package Package_With_Body is
type Basic_Record is private;
procedure Set_A (This : in out Basic_Record;
An_A : in Integer);
function Get_A (This : Basic_Record) return Integer;
private
type Basic_Record is
record
A : Integer;
end record ;
pragma Pure_Function (Get_A);
pragma Inline (Get_A);
pragma Inline (Set_A);
end Package_With_Body;
package body Package_With_Body is
procedure Set_A (This : in out Basic_Record;
An_A : in Integer)
is
begin
This.A := An_A;
end Set_A;
function Get_A (This : Basic_Record) return Integer is
begin
return This.A;
end Get_A;
end Package_With_Body;
```

To utilize a package it's needed to name it in a **with** clause, whereas to have direct visibility of that package it's needed to name it in a **use** clause.

3.4.2 TYPES OF PACKAGES

I. Nested packages: A nested package is a package declared inside a package. Like a normal package, it has a public part and a private part. A package may also be nested inside a subprogram. In fact, packages can be declared in any declarative part, including those of a block.

```
package P is
D: Integer;
-- a nested package:
package N is
X: Integer;
private
Foo: Integer;
end N;
```

II. Child packages: In Ada, you can extend the functionality of a unit (package) with so-called children (child packages). All the functionality of the parent is available to a child including all public and private declarations of the parent package are visible to all child packages.

```
with Ada.Strings.Unbounded; use Ada.Strings.Unbounded;
package Shelf.Books is
type Literature is (
Play,
Novel,
Poem,
Story,
Text,
Art);
type Book (Kind : Literature; Identifier : ID) is new Item (Identifier)
with record
Authors : Unbounded_String;
Title : Unbounded_String;
Year : Integer;
end record;
function match(it: Book; text: String) return Boolean;
end Shelf.Books;
```

Book has two components of type Unbounded_String, so **Ada.Strings.Unbounded** appears in a **with** clause of the child package. This is unlike the nested packages case which requires that all units needed by any one of the nested packages be listed in the context clause of the enclosing package. Child packages thus give better control over package dependences; **with** clauses are more local.

3.5 CONDITIONAL, UNCONDITIONAL AND LOOP CONTROL STRUCTURES

3.5.1 CONDITIONAL CONTROL STRUCTURES

Conditional clauses (**conditional control structures**) are blocks of code that will only execute if a particular expression (the condition) is true. In Ada, we can have two type; **i. *if-else* and ii. *case.***

i. *If-else*

The if-else statement is the simplest of the conditional statements. When the program arrives at an “if" statement during its execution, control will “branch" off into one of two or more “directions". An if-else statement is generally in the following form:

```
if condition then
statement;
else
other statement;
end if;
```

If the original condition is met, then all the code within the first statement is executed. The optional else section specifies an alternative statement that will be executed if the condition is false.

ii. *Case*

Often it is necessary to compare one specific variable against several constant expressions. For this kind of conditional expression the case statement exists. For example:

```
case X is
when 1 =>
Walk_The_Dog;
when 5 =>
Launch_Nuke;
when 8 | 10 =>
Sell_All_Stock;
when others =>
Self_Destruct;
end case;
```

3.5.2 UNCONDITIONAL CONTROL STRUCTURES

This control structure let you change the flow of your program without a condition but you should be careful when using unconditional controls because often they make you programs difficult to understand by the user. We can also have two types of unconditional controls in Ada; i. ***return*** and ii. ***goto***

i. *return*

This control ends a function and return to the calling procedure or function.

For procedures:

```
return;
```

For functions:

```
return Value;
```

ii. *goto*

The goto structure transfers control to the statement after the label.

```
goto Label;
```

```
Dont_Do_Something;
<<Label>>
...
```

3.5.3 LOOP CONTROL STRUCTURES

Loops allow you to have a set of statements repeated over and over again as many times as you choice. There are three categories of the loop structure in Ada; i. ***Endless loop*** ii. ***Loop with a condition*** iii. ***for loop***

i. ***Endless loop***

The endless loop is a loop which never ends and the statements inside are repeated forever. Never is meant as a relative term here — if the computer is switched off the loop will end.

```
Endless_Loop :
loop
Do_Something;
end loop Endless_Loop;
```

ii. ***Loop with a condition***

This loop can have a condition at the beginning (The statements are repeated as long as the condition is met. If the condition is not met at the very beginning then the statements inside the loop are never executed) or at the middle (the condition is within statements which has to be repeated it a criterion is met) or at the end (the statements are repeated until the condition is met and the check is at the end the statements which are at least executed once.). The syntax for the loop with on condition is as follows;

Loop with a Condition at the Beginning: *While_Loop* :

```
while X <= 5 loop
X := Calculate_Something;
end loop While_Loop;
```

Loop with a condition at the End: *Until_Loop* :

```
loop
X := Calculate_Something;
exit Until_Loop when X > 5;
end loop Until_Loop;
```

Loop with a condition at the Middle: *Exit_Loop* :

```
loop
X := Calculate_Something;
exit Exit_Loop when X > 5;
Do_Something (X);
end loop Exit_Loop;
```

iii. ***for loop***

Often, you may need a loop where a specific variable is counted from a given start value up or down to a specific end value.

```
For_Loop :
for I in Integer range 1 .. 10 loop
```

Do_Something (I)
end loop *For_Loop*;

the need for a loop which iterates over every element of an array brings **for loop** to play most times. The following sample code shows you how to achieve this:

Array_Loop :
for I **in** X'Range **loop**
X (I) := Get_Next_Element;
end loop *Array_Loop*;

With X being an array, this is mostly used on arrays but it will also work with other types when a full iteration is needed.

3.6 ARRAY AND RECORDS

To access element or a group of elements in index, they must first be stored in storage units as arrays and records.

3.6.1 ARRAY

An array is a collection of elements which can be accessed by one or more index values. In Ada any definite type is allowed as element and any discrete type, i.e. Range, Modular or Enumeration, can be used as an index. Ada's arrays are quite powerful and so there are quite a few syntax variations, which are presented below.

Basic Syntax

The basic form of an Ada array is:

array (Index_Range) **of** Element_Type

where Index_Range is a range of values within a discrete index type, and Element_Type is a definite subtype. If you for example want to count how often a specific letter appears inside a text, you could use:

type Character_Counter **is array** (Character) **of** Natural;

As a general advice, do not use Integer as the index range, since most of the time negative indices do not make sense. It is also a good style when using numeric indices, to define them starting in 1 instead of 0.

Arrays With Known Subrange

array (First .. Last) **of** Element_Type

Note that if First and Last are numeric literals, this implies the index type Integer.
If in the example above the character counter should only count upper case characters and discard all other characters, you can use the following array type:

type Character_Counter **is array** (Character **range** 'A' .. 'Z') **of** Natural;

Arrays With Unknown Subrange

Sometimes the range actually needed is not known until runtime or you need objects of different lengths. In Ada have the box ‘<>’, which allows us to declare indefinite arrays:

array (Index_Type **range** <>) **of** Element_Type

Arrays can have more than one index. Consider the following 2-dimensional array:

type Character_Display **is**
array (Positive **range** <>, Positive **range** <>) **of** Character;

This type permits declaring rectangular arrays of characters.

Example 4.1(a):

```
Magic_Square: constant Character_Display :=
(('S', 'A', 'T', 'O', 'R'),
('A', 'R', 'E', 'P', 'O'),
('T', 'E', 'N', 'E', 'T'),
('O', 'P', 'E', 'R', 'A'),
('R', 'O', 'T', 'A', 'S'));
```

By adding more dimensions to an array type, we could have squares, cubes (or ≪ bricks ≫), etc., of homogenous data items.

Finally, an array of characters is a string. Therefore, Magic_Square may simply be declared like this:

```
Magic_Square: constant Character_Display :=
("SATOR",
"AREPO",
"TENET",
"OPERA",
"ROTAS");
```

Concatenation Of Array

The operator "&" can be used to concatenate arrays:

Name := First_Name & ' ' & Last_Name;

In both cases, if the resulting array does not fit in the destination array, Constraint_Error is raised.

If you try to access an existing element by indexing outside the array bounds, Constraint_Error is raised (unless checks are suppressed).

Array Attributes

There are four Attributes which are important for arrays: 'First, 'Last, 'Length and 'Range. Lets look
at them with an example. Say we have the following three strings:

Hello_World : **constant** String := "Hello World!";
World : **constant** String := Hello_World (7 .. 11);
Empty_String : **constant** String := "";

Then the four attributes will have the following values:

Array 'First 'Last 'Length 'Range
Hello_World 1(first) 12(last) 12(length) 1 .. 12(range)
World 7(first) 11(last) 5(length) 7 .. 11(range)
Empty_String 1(first) 0(last) 0(length) 1 .. 0(range)

The example was choosen to show a few common beginner's mistakes:
1. The assumption that strings begin with the index value 1 is wrong.
2. The assumption (which follows from the first one) that X'Length = X'Last is wrong.
3. And last the assumption that X'Last > X'First; this is not true for empty strings.

Empty or Null Arrays

As you have seen in the section above, Ada allows for empty arrays. And — of course — you can have empty arrays of all sorts, not just String:

```
type Some_Array is array (Positive range <>) of Boolean;
Empty_Some_Array : constant Some_Array (1 .. 0) := (others => False);
```

If you give an initial expression to an empty array (which is a must for a constant), the expression in the aggregate will of course not be evaluated since there are no elements actually stored.

3.6.2 RECORDS

A **record** is a composite type that groups one or more fields. A field can be of any type, even a record. There are two types of records;

i. **Basic Record**: This type of record contains elements/data of different data types; the general syntax is of the form:

```
type Basic_Record is
record
A : Integer;
end record;
```

ii. **Null Record**: This type of record does not contain any data type; the general syntax is:

```
type Null_Record is
record
null;
```

3.7 DATA TYPE

Data types are all similar in most programming language but the way they are declared are different. In Ada, the most common datatypes are called predefined types and stored in the standard package; they include:

3.7.1 INTEGER

This type covers at least the range -2**15+1 .. +2**15-1. A **range** is an integer value which ranges from a First to a last Last. It is defined as

range First .. Last

When a value is assigned to a range it is checked for vality and an exceptions is raised when the value is not within First to Last. Unsigned integers in Ada have a value range from 0 to some positive number (not necessarily one
less than a power of 2). They are defined using the **mod** keyword because they implement a wrap-around arithmetic.

mod Modulus

where 'First is 0 and 'Last is Modulus - 1. Wrap-around arithmetic means that 'Last + 1 = 0 = 'First, and 'First - 1 = 'Last. Additionally to the normal arithmetic operators, bitwise **and**, **or** and **xor** are defined for the type.

3.7.2 FLOAT

There is only a very weak implementation requirement on this type; most of the time you would define your own floating-point types, and specify your precision and range requirements. To define a floating point type, you only have to say how many **digits** are needed, i.e. you define the relative precision:

digits Num_Digits

If you like, you can declare the minimum range needed as well:

digits Num_Digits **range** Low .. High

This facility is a great benefit of Ada over (most) other programming languages. In other languages, you just choose between "float" and "long float". In Ada, you specify the accuracy you need, and the compiler will choose an appropriate floating point type with *at least* the accuracy you asked for. This way, your requirement is guaranteed.

3.7.3 FIXED POINT

A fixed point type defines a set of values that are evenly spaced with a given absolute precision. In contrast, floating point values are all spaced according to a relative precision. The absolute precision is given as the delta of the type. There are two kinds of fixed point types; ordinary and decimal.

1. **Ordinary Fixed Point** types; The delta gives a hint to the compiler how to choose the small value if it is not specified: It can be *any power of two* not greater than delta. You may specify the small via an attribute clause to be *any value* not greater than delta. (If the compiler cannot conform to this small value, it has to reject the declaration.) **delta** *Delta* **range** *Low .. High*

2. **Decimal Fixed Point** types; the small is defined to be the delta, which in turn must be a power of ten. (Thus you cannot specify the small by an attribute clause.)

For example, if you define a decimal fixed point type with a delta of 0.1, you will be able to accurately store the values 0.1, 1.0, 2.2, 5.7, etc. You will not be able to accurately store the value 0.01. Instead, the value will be rounded down to 0.0.

If the compiler accepts your fixed point type definition, it guarantees that values represented by that type will have at least the degree of accuracy specified (or better). If the compiler cannot support the type definition (e.g. due to limited hardware) then a compile-time error will result. **delta** *Delta* **digits** *Num_Digits*

3.7.4 CHARACTER

A special form of Enumerations. There are two predefined kinds of character types: 8-bit characters (called Character) and 16-bit characters (called Wide_Character). Ada 2005 adds a 32-bit character type called Wide_Wide_Character.

3.7.5 STRING AND WIDE_STRING

Two indefinite array types, of Character and Wide_Character respectively. Ada 2005 adds a Wide_Wide_String type. The standard library contains packages for handling strings in three variants: fixed length (Ada.Strings.Fixed), with varying length below a certain upper bound (Ada.Strings.Bounded), and unbounded length (Ada.Strings.Unbounded). Each of these packages has a Wide_ and a Wide_Wide_ variant.

3.7.6 BOOLEAN

A Boolean in Ada is an Enumeration of False and True with special semantics. These types makes use of the logical and the relationnal operators on data; such as >, <, <= and so on.

3.8 OPERATORS ON DATA

Operators are either are keywords or delimiters -- hence all operator pages are redirects to the appropiate keyword or delimiter. Here is a list of sorted operators from lowest precedence to highest precedence.

1. Logical operators

- and $x \wedge y$ (also keyword and)

- or $x \vee y$ (also keyword or)
- xor :exclusive or $(x \wedge \bar{y}) \vee (\bar{x} \wedge y)$ (also keyword xor)

2. Relational operators

- /= Not Equal $x \neq y$ (also special character /=)
- = Equal $x = y$ (also special character =)
- < Less than $x < y$ (also special character <)
- <= Less than or equal to ($x \leq y$) (also special character <=)
- Greater than ($x > y$), (also special character >)
- >= Greater than or equal to ($x \geq y$), (also special character >=)

3. Binary adding operators

- + Add $x + y$ (also special character +)
- - Subtract $x - y$ (also special character -)
- & Concatenate x & y (also special character &)

4. Unary adding operators

- + Plus sign $+x$ (also special character +)
- - Minus sign **-x** (also special character -)

5. Multiplying operator

- Multiply $x \times y$ (also special character *)
- / Divide x/y (also special character /)
- mod modulus (also keyword mod)
- rem remainder (also keyword rem)

6. Highest precedence operator

- ** Power x^y (also special character **)
- not logical not $\neg x$ (also keyword not)
- abs absolute value $|x|$ (also keyword abs)

4.0 CONCLUSION

Being a master in Ada programming language gives you an edge in software engineering and maintenance. Though Ada is not a common programming language but it has been recognized as one of the most powerful and flexible language; further study and research on the language will provide you a deeper understanding of the program.

5.0 SUMMARY

In this unit, you learnt the basic operations on data types in Ada and how they are declared. You learnt how to use procedures, function and declare array. Also you learnt how to use arrays and control structure.

6.0 TUTOR MARKED ASSIGNMENT

1. Who invented Ada programming language?
2. State the features of Ada programming language
3. Write a simple interactive Ada program that will output your name, state of origin and favorite food.
4. What are comments, procedures and functions?
5. Using the operations on data, write a program to calculate 7 factorial.
6. Write a simple program that will calculate sum of numbers 1 to 10 using any control structure.

7.0 FURTHER READING

http://en.wikipedia.org/wiki/Wikibooks:Ada_Programming

MODULE 4 - USING ADA AND COBOL

UNIT 2 – COBOL Programming Language

Content Page

1.0 INTRODUCTION

The aim of this unit is to provide a brief introduction to the programming language COBOL. To provide a context in which its uses might be understood. To provide an introduction to the major structures present in a COBOL program. **COBOL** is one of the oldest programming languages. Its name is an acronym for **CO**mmon **B**usiness-**O**riented **L**anguage, defining its primary domain in business, finance, and administrative systems for companies and governments.

2.0 OBJECTIVES

1. Know what the acronym COBOL stands for.
2. Be aware of the significance of COBOL in the marketplace.
3. Be aware of the COBOL coding rules
4. Understand the structure of COBOL programs
5. Understand the purpose of the IDENTIFICATION, ENVIRONMENT, DATA and PROCEDURE divisions.

3.0 BRIEF HISTORY OF COBOL

It appeared in 1959 and was designed by Grace Hopper, William Selden, Gertrude Tierney, Howard Bromberg, Howard Discount, Vernon Reeves, Jean E. Sammet. The stable release was in 2002; the COBOL 2002 standard includes support for object-oriented programming and other modern language features. The specifications were to a great extent inspired by the FLOW-MATIC language invented by Grace Hopper - commonly referred to as "the mother of the COBOL language." The decision to use the name "COBOL" was made at a meeting of the committee held on 18 September 1959. The subcommittee completed the specifications for COBOL in December 1959. The first compilers for COBOL were subsequently implemented during the year 1960 and on 6 and 7 December essentially the same COBOL program was run on two different makes of computers, an RCA computer and a Remington-Rand Univac computer, demonstrating that compatibility could be achieved. The language continues to evolve today. In the early 1990s it was decided to add object-orientation in the next full revision of COBOL. The initial estimate was to have this revision completed by 1997. Some implementers (including Micro Focus, Fujitsu, Veryant, and IBM) introduced object-oriented syntax based on the 1997 or other drafts of the full revision. The final approved ISO Standard (adopted as an ANSI standard by INCITS) was approved and made available in 2002. Like the C++ and Java programming languages, object-oriented COBOL compilers are available even as the language moves toward standardization.

COBOL programs are in use globally in governmental and military agencies and in commercial enterprises, and are running on operating systems such as IBM's z/OS, the POSIX families (Unix/Linux etc.), and Microsoft's Windows as well as ICL's VME operating system and Unisys' OS 2200. Three ANSI standards for COBOL have been produced: in 1968, 1974 and 1985. A new COBOL standard introducing object-oriented programming to COBOL is due within the next few years.

For over four decades COBOL has been the dominant programming language in the business computing domain. In that time it it has seen off the challenges of a number of other languages such as PL1, Algol68, Pascal, Modula, Ada, C, C++. All these languages have found a niche but none has yet displaced COBOL. Two recent challengers though, Java and Visual Basic, are proving to be serious contenders.

3.1 FEATURES OF COBOL

COBOL as defined in the original specification included a PICTURE clause for detailed field specification. It did not support local variables, recursion, dynamic memory allocation, or structured programming constructs. Support for some or all of these features has been added in later editions of the COBOL standard. COBOL has many reserved words (over 400), called keywords. Below are some features of COBOL;

1. Self-modifying code

The original COBOL specification supported self-modifying code via the infamous "ALTER X TO PROCEED TO Y" statement. X and Y are paragraph labels, and any "GOTO X" statements executed after such an ALTER statement have the meaning "GOTO Y" instead. Most compilers still support it, but it should not be used in new programs.

2. Syntactic features

COBOL provides an update-in-place syntax, for example;

```
ADD YEARS TO AGE
```

The equivalent construct in many procedural languages would be

```
age = age + years
```

This syntax is similar to the compound assignment operator later adopted by C:

```
age += years
```

The abbreviated conditional expression

```
IF SALARY > 9000 OR SUPERVISOR-SALARY OR = PREV-SALARY
```

is equivalent to

```
IF SALARY > 9000
    OR SALARY > SUPERVISOR-SALARY
    OR SALARY = PREV-SALARY
```

COBOL provides "named conditions" (so-called 88-levels). These are declared as sub-items of another item (the conditional variable). The named condition can be used in an IF statement, and tests whether the conditional variable is equal to any of the values given in the named condition's VALUE clause. The SET statement can be used to make a named condition TRUE (by assigning the first of its values to the conditional variable). COBOL allows identifiers to be up to 30 characters long. When COBOL was introduced, much shorter lengths (e.g., 6 characters for FORTRAN) were prevalent. The concept of *copybooks* was introduced by COBOL; these are chunks of code which can be inserted into a program's code. This is done with the COPY statement, which also allows parts of the copybook's code to be replaced with other code (using the REPLACING ... BY ... clause).

3. Simplicity Feature
COBOL is a simple language (no pointers, no user defined functions, no user defined types) with a limited scope of function. It encourages a simple straightforward programming style. Curiously enough though, despite its limitations, COBOL has proven itself to be well suited to its targeted problem domain (business computing). Most COBOL programs operate in a domain where the program complexity lies in the business rules that have to be encoded rather than in the sophistication of the data structures or algorithms required. And in cases where sophisticated algorithms are required COBOL usually meets the need with an appropriate verb such as the SORT and the SEARCH.

We noted above that COBOL is a simple language with a limited scope of function. And that is the way it used to be but the introduction of OO-COBOL has changed all that. OO-COBOL retains all the advantages of previous versions but now includes -

- User Defined Functions
- Object Orientation
- National Characters - Unicode
- Multiple Currency Symbols
- Cultural Adaptability (Locales)
- Dynamic Memory Allocation (pointers)
- Data Validation Using New VALIDATE Verb
- Binary and Floating Point Data Types
- User Defined Data Types

4. Non-proprietary (portable) Feature
The COBOL standard does not belong to any particular vendor. The vendor independent ANSI COBOL committee legislates formal, non-vendor-specific syntax and semantic language standards. COBOL has been ported to virtually every hardware platform - from every favour of Windows, to every falser of Unix, to AS/400, VSE, OS/2, DOS, VMS, Unisys, DG, VM, and MVS.

5. COBOL is Maintainable

COBOL has a proven track record for application maintenance, enhancement and production support at the enterprise level. COBOL applications were actually cheaper to fix than applications written in more recent languages. One reason for the maintainability of COBOL programs has been given above - the readability of COBOL code. Another reason is COBOL's rigid hierarchical structure. In COBOL programs all external references, such as to devices, files, command sequences, collating sequences, the currency symbol and the decimal point symbol, are defined in the Environment Division.

When a COBOL program is moved to a new machine, or has new peripheral devices attached, or is required to work in a different country; COBOL programmers know that the parts of the program that will have to be altered to accommodate these changes will be isolated in the Environment Division. In other programming languages, programmer discipline could have ensured that the references liable to change were restricted to one part of the program but they could just as easily be spread throughout the program. In COBOL programs, programmers have no choice. COBOL's rigid hierarchical structure ensures that these items are restricted to the Environment Division.

3.2 COBOL APPLICATIONS

COBOL applications are often very large. Many COBOL applications consist of more than 1,000,000 lines of code - with 6,000,000+ line applications not considered unusually large in many shops. COBOL applications are also very long-lived. The huge investment in creating a software application consisting of some millions of lines of COBOL code means that the application cannot simply be discarded when some new programming language or technology appears. As a consequence business applications between 10 and 30 years-old are common. This accounts for the predominance of COBOL programs. COBOL applications often run in critical areas of business. For instance, over 95% of finance–insurance data is processed with COBOL. COBOL applications often deal with enormous volumes of data. Single production files and databases measured in terabytes are not uncommon.

3.3 DATA TYPES IN COBOL

Standard COBOL provides the following data types

Table 4.2(a): Standard COBOL Data Types:

Data type	**Sample declaration**	**Notes**
Character	PIC X(20) PIC A(4)9(5)X(7)	Alphanumeric and alphabetic-only Single-byte character set (SBCS)
Edited character	PIC X99BAXX	Formatted and inserted characters
Numeric fixed-point	PIC S999V99	Binary 16, 32, or 64 bits (2, 4, or

binary	[USAGE] COMPUTATIONAL or BINARY	8 bytes) Signed or unsigned. Conforming compilers limit the maximum value of variables based on the picture clause and not the number of bits reserved for storage.
Numeric fixed-point packed decimal	PIC S999V99 PACKED-DECIMAL	1 to 18 decimal digits (1 to 10 bytes) Signed or unsigned
Numeric fixed-point zoned decimal	PIC S999V99 [USAGE DISPLAY]	1 to 18 decimal digits (1 to 18 bytes) Signed or unsigned Leading or trailing sign, overpunch or separate
Numeric floating-point	PIC S9V999ES99	Binary floating-point
Edited numeric	PIC +Z,ZZ9.99 PIC $***,**9.99CR	Formatted characters and digits

Group *(record)*	01 CUST-NAME.
	  05 CUST-LAST PIC X(20).
	  05 CUST-FIRST PIC X(20).
		Aggregated elements

Table *(array)*	OCCURS 12 TIMES	Fixed-size array, row-major order Up to 7 dimensions
Variable-length table	OCCURS 0 to 12 TIMES DEPENDING ON CUST-COUNT	Variable-sized array, row-major order Up to 7 dimensions
Renames *(variant or union data)*	66 RAW-RECORD RENAMES CUST-RECORD	Character data overlaying other variables

Condition name	88 IS-RETIRED-AGE VALUES 65 THRU 150	Boolean value dependent upon another variable
Array index	[USAGE] INDEX	Array subscript

Most vendors provide additional types, such as:

Table 4.2(b): Additional COBOL Data Types

Data type	**Sample declaration**	**Notes**
Numeric floating-point single precision	PIC S9V999ES99 [USAGE] COMPUTATIONAL-1	Binary floating-point (IBM extension)
Numeric floating-point double precision	PIC S9V999ES99 [USAGE] COMPUTATIONAL-2	Binary floating-point (IBM extension)
Numeric fixed-point packed decimal	PIC S9V999 [USAGE] COMPUTATIONAL-3	same as PACKED DECIMAL (IBM extension)
Numeric fixed-point binary	PIC S999V99 [USAGE] COMPUTATIONAL-4	same as COMPUTATIONAL or BINARY (IBM extension)
Numeric fixed-point binary (native binary)	PIC S999V99 [USAGE] COMPUTATIONAL-5	Binary 16, 32, or 64 bits (2, 4, or 8 bytes) Signed or unsigned. The maximum value of variables based on the number of bits reserved for storage and not on the picture clause. (IBM extension)
Numeric fixed-point binary in native byte order	PIC S999V99 [USAGE] COMPUTATIONAL-4	Binary 16, 32, or 64 bits (2, 4, or 8 bytes) Signed or unsigned

Numeric fixed-point binary in big-endian byte order	PIC S999V99 [USAGE] COMPUTATIONAL-5	Binary 16, 32, or 64 bits (2, 4, or 8 bytes) Signed or unsigned
Wide character	PIC G(20)	Alphanumeric Double-byte character set (DBCS)
Edited wide character	PIC G99BGGG	Formatted and inserted wide characters
Edited floating-point	PIC +9.9(6)E+99	Formatted characters and decimal digits
Data pointer	[USAGE] POINTER	Data memory address
Code pointer	[USAGE] PROCEDURE-POINTER	Code memory address
Bit field	PIC 1(*n*) [USAGE] COMPUTATIONAL-5	*n* can be from 1 to 64, defining an *n*-bit integer Signed or unsigned
Index	[USAGE] INDEX	Binary value corresponding to an occurrence of a table element May be linked to a specific table using INDEXED BY

Comparing COBOL with other programming language, it is obvious that they are not similar.

3.4 COBOL PROGRAMMING

Programming in COBOL in particular, is done by writing some simple COBOL programs that use the three main programming constructs - Sequence, Iteration and Selection. Any program consists of three main things;

1. The computer statements needed to do the job
2. Declarations for the data items that the computer statements need.
3. A plan, or algorithm, that arranges the computer statements in the program so that the computer executes them in the correct order.

We want to write a program which will accept two numbers from the users keyboard, multiply them together and display the result on the computer screen;

1. We will need a statement to take in the first number and store it in the named memory location (a variable) - Num1
   ACCEPT Num1.
2. We will need a statement to take in the second number and store it in the named memory location - Num2
   ACCEPT Num2.
3. We will need a statement to multiply the two numbers together and to store the result in the named location - Result
   MULTIPLY Num1 BY Num2 GIVING Result.
4. We will need a statement to display the value in the named memory location "**Result**" on the computer screen -
   DISPLAY "Result is = ", Result.

Another example is the “Hello world” program in COBOL;

```
IDENTIFICATION DIVISION.

PROGRAM-ID. HELLO-WORLD.

PROCEDURE DIVISION.

   DISPLAY 'Hello, world'.

   STOP RUN.
```

The fundamentals of constructing COBOL programs, explains the notation used in COBOL syntax diagrams and enumerates the COBOL coding rules. It shows how user-defined names are constructed and examines the structure of COBOL programs. When COBOL was developed one of the design goals was to make it as English-like as possible. As a result, COBOL uses structural concepts normally associated with English prose such as section, paragraph and sentence. It also has an extensive reserved word list with over 300 entries and the reserved words themselves tend to be long.

Although modern COBOL (COBOL 85 and OO-COBOL) has introduced many of the constructs required to write well structured programs it also still retains elements which, if used, make it difficult, and in some cases impossible, to write good programs. COBOL syntax is defined using particular notation sometimes called the COBOL MetaLanguage.

Words in mixed case represent names that must be devised by the programmer (like data item names). When material is enclosed in curly braces **{ }**, a choice must be made from the options within the braces. If there is only one option then that item in mandatory.

Material enclosed in square brackets **[ ]**, indicates that the material is optional, and may be included or omitted as required. The ellipsis symbol **...** (three dots), indicates that the preceding syntax element may be repeated at the programmer's discretion.

To simplify the syntax diagrams and reduce the number of rules that must be explained, in some diagrams special operand endings have been used (note that this is my own extension - it is not standard COBOL).

These special operand endings have the following meanings:

Table 4.2(c)

$i	uses an alphanumeric data-item
$il	uses an alphanumeric data-item or a string literal
#i	uses a numeric data-item
#il	uses a numeric data-item or numeric literal
$#i	uses a numeric or an alphanumeric data-item

3.5 NAME RULES IN COBOL

All user-defined names, such as data names, paragraph names, section names condition names and mnemonic names, must adhere to the following rules:

1. They must contain at least one character, but not more than 30 characters.
2. They must contain at least one alphabetic character.
3. They must not begin or end with a hyphen.
4. They must be constructed from the characters A to Z, the numbers 0 to 9, and the hyphen.
5. They must not contain spaces.
6. Names are not case-sensitive: TotalPay is the same as totalpay, Totalpay or TOTALPAY.

3.6 HIERARCHICAL STRUCTURE OF COBOL PROGRAM

COBOL programs are hierarchical in structure. Each element of the hierarchy consists of one or more subordinate elements. The hierarchy consists of Divisions, Sections, Paragraphs, Sentences and Statements. We can represent the COBOL hierarchy using the COBOL metalanguage. A

Division may contain one or more Sections, a Section one or more Paragraphs, a Paragraph one or more Sentences and a Sentence one or more Statements.

I. Divisions
A division is a block of code, usually containing one or more sections, that starts where the division name is encountered and ends with the beginning of the next division or with the end of the program text.

II. Sections
A section is a block of code usually containing one or more paragraphs. A section begins with the section name and ends where the next section name is encountered or where the program text ends.

Section names are devised by the programmer, or defined by the language. A section name is followed by the word SECTION and a period.
See the two example names below -

```
SelectUnpaidBills SECTION.
FILE SECTION.
```

III. Paragraphs
A paragraph is a block of code made up of one or more sentences. A paragraph begins with the paragraph name and ends with the next paragraph or section name or the end of the program text.

A paragraph name is devised by the programmer or defined by the language, and is followed by a period.
See the two example names below -

```
PrintFinalTotals.
PROGRAM-ID.
```

IV. Sentences and statements
A sentence consists of one or more statements and is terminated by a period.
For example:

```
MOVE .21 TO VatRate
   MOVE 1235.76 TO ProductCost
   COMPUTE VatAmount = ProductCost * VatRate.
```

A statement consists of a COBOL verb and an operand or operands.
For example:

```
SUBTRACT Tax FROM GrossPay GIVING NetPay
```

3.7 QUICK LOOK AT DIVISION

At the top of the COBOL hierarchy are the four divisions. These divide the program into distinct structural elements. Although some of the divisions may be omitted, the sequence in which they are specified is fixed, and must follow the order below.

IDENTIFICATION DIVISION.
Contains program information

ENVIRONMENT DIVISION.
Contains environment information

DATA DIVISION.
Contains data descriptions

PROCEDURE DIVISION.
Contains the program algorithms

1**. IDENTIFICATION DIVISION**: This supplies information about the program to the programmer and the compiler. Most entries in the IDENTIFICATION DIVISION are directed at the programmer. The compiler treats them as comments. The PROGRAM-ID clause is an exception to this rule. Every COBOL program must have a PROGRAM-ID because the name specified after this clause is used by the linker when linking a number of subprograms into one run unit, and by the CALL statement when transferring control to a subprogram. The IDENTIFICATION DIVISION has the following structure:

IDENTIFICATION DIVISION
PROGRAM-ID. NameOfProgram.
[AUTHOR. YourName.]
other entries here

The keywords - IDENTIFICATION DIVISION - represent the division header, and signal the commencement of the program text. PROGRAM-ID is a paragraph name that must be specified immediately after the division header. NameOfProgram is a name devised by the programmer, and must satisfy the rules for user-defined names.

Here's a typical program fragment:

IDENTIFICATION DIVISION.
PROGRAM-ID. SequenceProgram.
AUTHOR. Onashoga Adebukola.

2. **ENVIRONMENT DIVISION**: This is used to describe the environment in which the program will run. The purpose of the ENVIRONMENT DIVISION is to isolate in one place all aspects of the program that are dependent upon a specific computer, device or encoding sequence. The idea behind this is to make it easy to change the program when it has to run on a different computer or one with different peripheral devices. In the ENVIRONMENT DIVISION, aliases are assigned to external devices, files or command sequences. Other environment details, such as the collating sequence, the currency symbol and the decimal point symbol may also be defined here.

3. **DATA DIVISION**: As the name suggests, provides descriptions of the data-items processed by the program. The DATA DIVISION has two main sections: the FILE SECTION and the WORKING-STORAGE SECTION. Additional sections, such as the LINKAGE SECTION (used in subprograms) and the REPORT SECTION (used in Report Writer based programs) may also be required. The FILE SECTION is used to describe most of the data that is sent to, or comes from, the computer's peripherals. The WORKING-STORAGE SECTION is used to describe the general variables used in the program.

Below is a sample program fragment

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SequenceProgram.
AUTHOR. Onashoga Adebukola.

DATA DIVISION.
WORKING-STORAGE SECTION.
01  Num1          PIC 9  VALUE ZEROS.
01  Num2          PIC 9  VALUE ZEROS.
01  Result        PIC 99 VALUE ZEROS.
```

4. **PROCEDURE DIVISION**: This contains the code used to manipulate the data described in the DATA DIVISION. It is here that the programmer describes his algorithm. The PROCEDURE DIVISION is hierarchical in structure and consists of sections, paragraphs, sentences and statements. Only the section is optional. There must be at least one paragraph, sentence and statement in the PROCEDURE DIVISION. Paragraph and section names in the PROCEDURE DIVISION are chosen by the programmer and must conform to the rules for user-defined names.

Sample Program

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SequenceProgram.
AUTHOR. Onashoga Adebukola.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 Num1 PIC 9 VALUE ZEROS.
01 Num2 PIC 9 VALUE ZEROS.
01 Result PIC 99 VALUE ZEROS.

PROCEDURE DIVISION.
CalculateResult.
   ACCEPT Num1.
   ACCEPT Num2.
   MULTIPLY Num1 BY Num2 GIVING Result.
   DISPLAY "Result is = ", Result.
   STOP RUN.
```

Some COBOL compilers require that all the divisions be present in a program while others only require the IDENTIFICATION DIVISION and the PROCEDURE DIVISION. For instance the program shown below is perfectly valid when compiled with the Microfocus NetExpress compiler.

Minimum COBOL program

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  SmallestProgram.
PROCEDURE DIVISION.
DisplayGreeting.
   DISPLAY "Hello world".
   STOP RUN.
```

4.0 CONCLUSION

COBOL is one of the simplest and most interesting programming language though it is not so common in other areas of programming application but a deeper look will help you take advantage of its usefulness especially if you choice to move into the business world.

5.0 SUMMARY

This unit has briefly introduced you to the concept behind COBOL and the full meaning of the acronym “COBOL”, reasons why it succeeded in the area of computer application in business. You have learnt the simple introduction into rules for user-defined names, division syntax and the COBOL hierarchy.

6.0 TUTOR MARKED ASSIGNMENT

1. What is the full meaning of COBOL? and state area where it is used.

2. Briefly explain the hierarchical structure of programming.

3. List 7 data type in COBOL and their sample declaration.

7.0 FURTHER READING