COMPUTER PROGRAMMING
AND
NUMERICAL METHODS

(For B.E./B.Tech Engineering Students)

As per New Revised Syllabus of
APJ Abdul Kalam Technological University

COMPUTER PROGRAMMING
Dr. G. Mathivanan
B.Sc., B.Tech., M.E., Ph.D
Professor,
Dr. V. Maria Anu M.E., Ph.D
Dr. V. Rajalakshmi M.E., Ph.D
R. Sathiavathi M.E., (Ph.D)
AP-Department of Information Technology,
Sathyabama University, CHENNAI - 119

NUMERICAL METHODS
L.N.Narayanan, M.Sc, M.Phil.,
Assistant Professor,
Department of Science & Humanities,
(mathematics Division)
Jerusalem College of Engineering,
Pallikaranai, Chennai-100

AIRWALK PUBLICATIONS
(Near All India Radio)
80, Karneeshwarar Koil Street
Mylapore, Chennai - 600 004.
Ph.: 2466 1909, 94440 81904
Email: aishram2006@gmail.com,
airwalk800@gmail.com
www.airwalkpublications.com
# SYLLABUS

## COMPUTER PROGRAMMING & NUMERICAL METHODS

### Course Plan

<table>
<thead>
<tr>
<th>Module</th>
<th>Contents</th>
<th>Sem. Exam Marks</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>Introduction to Computer programming concept –internal representation of data - Algorithm and flow chart, Basics of procedure oriented and object oriented programming. Introduction to C++: Structure of C++ program; Keywords; Identifiers; Data types – integer, real, character, string, boolean, enumeration, Constant and Variables; Operators – assignment, arithmetic, relational, logical, increment, decrement and conditional operators; Statements – simple &amp; compound, declaration statements. Input and output streams.</td>
<td>15%</td>
</tr>
<tr>
<td>II</td>
<td>Control statements: if, if-else, switch, for, while, do-while, break and continue statements, Arrays – one dimensional &amp; two dimensional; Functions: inline functions, function over loading, Functions with default arguments, recursion.</td>
<td>15%</td>
</tr>
<tr>
<td>III</td>
<td>Basics of Pointers. Function call by value, call by reference. Preparation of programs for evaluation of Factorial of a number, infinite series, Sorting, Searching and Matrix multiplication.</td>
<td>15%</td>
</tr>
<tr>
<td>IV</td>
<td>Introduction to Class and Object- definition, data members, member function. private &amp; public member functions, member access, friend declaration, class objects, predefined classes, initialization. Inheritance-base class and derived class. Simple programs using the above features. (No programming questions for University examination and internals)</td>
<td>15%</td>
</tr>
<tr>
<td>V</td>
<td>Errors and approximations, sources of errors. Solution of linear system of equations: Gauss elimination, Gauss-Jordan and Gauss–Seidel methods. Interpolation: Lagrange and Aitken techniques.</td>
<td>20%</td>
</tr>
</tbody>
</table>
CONTENT

Module - I: Fundamentals of Programming

1.1 Introduction to Programming
   Different Levels of programs ...................................... 1.1
   Programming language paradigms .............................. 1.2
   1. Procedure Oriented languages .......................... 1.2
   2. Object Oriented programming languages ................. 1.2
   3. Special programming languages ......................... 1.3
1.2 Internal Representation of Data ................................ 1.4
1.3 Algorithms and Flowcharts ................................. 1.4
   1.3.1 Algorithms .............................................. 1.4
   1.3.2 Flowchart ............................................... 1.5
1.4 Basics of Procedure Oriented Programming and object
   Oriented Programming ............................................. 1.6
   1.4.1 Procedure-Oriented Programming (POP) ............. 1.6
   1.4.2 Object Oriented Programming (OOPs) ............... 1.7
1.5 Introduction to C++ ............................................. 1.8
1.6 Structure of C++ .................................................. 1.9
1.7 Keywords .......................................................... 1.10
1.8 Identifiers .......................................................... 1.11
1.9 Data Types ........................................................ 1.12
   1.9.1 Built-in Data Types .................................... 1.12
   1.9.2 Enumerated Data Type .................................. 1.14
   1.9.2 Strings ..................................................... 1.16
1.10 Constants and Variables ........................................ 1.17
   1.10.1 Constants ............................................... 1.17
   1.10.2 Variables ............................................... 1.17
1.11 Operators ........................................................ 1.19
   1.11.1 Assignment operator .................................. 1.19
   1.11.2 Arithmetic operators ............................... 1.19
   1.11.3 Relational operations ............................... 1.21
Module - II: Control Statements, Arrays & Functions

2.1 Selection Control Statements ........................................... 2.1
   2.1.1 if statement ................................................. 2.1
   2.1.2 if ... else statement ................................... 2.4
   2.1.3 if_else ladder ............................................. 2.9
   2.1.4 switch ... case statement .................................. 2.10

2.2 Looping Statement .................................................... 2.15
   2.2.1 for statement ............................................... 2.15
   2.2.2 While loop ............................................... 2.19
   2.2.3 do ... while loop ......................................... 2.21

2.3 Unconditional Jump Statements .................................... 2.23
   2.3.1 break statement ........................................... 2.23
   2.3.2 continue statement ....................................... 2.24

2.4 Arrays ................................................................. 2.26
   2.4.1 One dimensional array .................................... 2.27
   2.4.2 Two dimensional arrays .................................... 2.30

2.5 Functions ............................................................. 2.34
   2.5.1 Function declarations ...................................... 2.34
   2.5.2 Function call/calling a function ......................... 2.35
   2.5.3 Function call ............................................... 2.36

2.6 Inline Function .......................................................... 2.37

2.7 Function Overloading .................................................... 2.39

2.8 Functions With Default Arguments .................................. 2.41

2.9 Recursion ............................................................... 2.42
Module - III: Basics of Pointers

3.1 Basics of Pointers
3.2 Function Call by Value
3.3 Call by Reference
3.4 Factorial of a Number
3.5 Infinite Series
3.6 Sorting
3.7 Searching
3.7.1 Linear search
3.7.2 Binary search
3.8 Matrix Multiplication

Module - IV: Introduction to Class and Object

4.1 Class and Object Introduction
4.2 Access Specifier
4.2.1 private
4.2.2 public
4.3 Objects
4.4 Defining Member Functions
4.5 Arrays of Objects
4.6 Friend Functions
4.7 Initialization
4.7.1 Constructors
4.7.2 Parameterized constructors
4.7.3 Destructor
4.8 Features of OOPS
4.9 Inheritance
4.9.1 Single inheritance
4.9.2 Multiple Inheritance
4.9.3 Hierarchial Inheritance
4.9.4 Multilevel Inheritance
4.9.5 Hybrid Inheritance
Module - V: Solution of Equations, Interpolation and Approximation

5.1 Errors and Approximations ........................................ 5.1
5.2 Accuracy of Numbers ............................................. 5.2
5.3 Sources of Errors (Types of Errors) ......................... 5.3
5.4 Error in the Approximation of a Function ... 5.6
5.5 Error in a Series Approximation ......................... 5.8
5.6 Order of Approximation .................................. 5.10
5.7 Propagation of Error ...................................... 5.11
5.8 Solutions of Linear Algebraic Equations ............. 5.12
5.9 Gauss Elimination Method .................................... 5.14
  5.9.1 Gauss - Jordan Method .................................... 5.29
5.10 Iterative Methods ............................................ 5.40
5.11 Gauss-seldel Iterative Method ......................... 5.40
5.12 Interpolation .................................................. 5.58
5.13 Inverse Interpolation ........................................ 5.66
5.14 Fixed Point Representation .................................. 5.69
5.15 Sufficient Condition For Convergence of Iterations .... 5.69
5.15 Aitken Techniques ............................................ 5.75

Module - VI: Cure Fitting, Correlation and Boundary Value Problems in Partial Differential Equations

6.1 Curve Fitting ..................................................... 6.1
  6.1.1 Method of least squares .................................. 6.1
  6.1.2 Non-linear Relationships .................................. 6.2
6.2 Linear Correlation & Measures of Correlation ........ 6.17
  6.2.1 Correlation: ............................................... 6.17
  6.2.2 Scatter Diagram ......................................... 6.17
  6.2.3 Rank correlation ......................................... 6.35
  6.2.4 Repeated Ranks ........................................... 6.40
6.3 Numerical Solution of Partial Differential Equations .... 6.44
6.3.1 Difference equations .......................... 6.48
6.3.2 Classification of partial differential equations of the
second order ........................................... 6.48
6.3.3 Solution of laplace's equation ................... 6.50

6.4 Finite Difference Method .......................... 6.76
  6.4.1 Parabolic equations (one dimensional heat equation) 6.76
  6.4.2 Crank Nicolson difference scheme for solving parabolic
equation ................................................ 6.82
  6.4.3 Hyperbolic equations ........................... 6.91
  6.4.4 Elliptic Equations ............................... 6.98
  6.4.5 Poisson's equation .............................. 6.101
Introduction to Computer programming concept - internal representation of data - Algorithm and flow chart, Basics of procedure oriented and object oriented programming. Introduction to C++: Structure of C++ program; Keywords; Identifiers; Data types - integer, real, character, string, boolean, enumeration, Constant and Variables; Operators - assignment, arithmetic, relational, logical, increment, decrement and conditional operators; Statements - simple & compound, declaration statements. Input and output streams.

1.1 INTRODUCTION TO PROGRAMMING

A programming language is a set of rules that provides a way of telling a computer what to do. It has set of rules to implement an algorithm. An algorithm is sequence of steps to perform an operation.

People communicate in natural languages like English. It has alphabets, words, symbols and grammar rules. Likewise a programming language also has words, symbols and rules of grammar. The grammatical rules are termed as syntax. Each programming language has a different set of syntax rules.

There are many programming languages as people communicate in different languages like French, Hindi, etc. The first programming languages were developed in the 1950s. Since then thousands of programming languages have been developed.

Different Levels of programs

1. **High level programs:** These programs are easily understandable for humans. Examples: C, C++, Java

2. **Low level programs:** These programs are machine understandable. Programs are made up of simple instructions like load, add ... etc.

3. **Executable machine code:** It is like continuous bit stream with 0’s and 1’s. There is no readability.
Programming language paradigms

Programming languages are categorised into two major categories.

1. Procedure oriented languages
   2. Object oriented languages

1. Procedure Oriented languages

   These are traditional programming languages which executes sequence of instructions. Following are the few examples.

(a) FORTRAN - FORmula TRANslation
   It was developed at IBM in the mid 1950’s for scientific and mathematical applications by scientists and engineers.

(b) COBOL - Common Business Oriented Language
   It was developed in 1959 to design business applications.

(c) BASIC - Beginner’s All purpose Symbolic Instruction Code
   It was developed in 1960’s to help students to write simple code so that they can interact with computers.

(d) C Programming
   It was developed in 1970’s, an efficient programming language which is useful in developing high end applications. It is useful in system programming. The Unix operating system is written in C.

2. Object Oriented programming languages

(a) Simula
   Simula stands for simulation Language. It was the first object oriented language developed in 1960’s which used concept of classes and objects.

(b) Smalltalk
   It was developed in 1970’s. Smalltalk is still in use in some of the old generation computers. In smalltalk there was no primitives and control structure.

   Everything is an object and objects communicate by sending messages.
(c) **C++**

It is similar to ‘C’ language with added features like objects, classes, inheritance etc. It is an efficient an widely used object oriented language. It also used developing system softwares and application softwares.

(d) **Java**

An object oriented language that eliminates the problems in C++ like platform dependency, debugging and synchronization. It allows web page developers to create programs for applications. The special feature of Java is platform independence.

3. Special programming languages

(a) **Scripting languages**

It is a programming language designed for integrating and communicating with other programming languages.

Example: Javascript, PHP, ASP, Asp, Perl, Python

(b) **Command languages**

The programming language through which a user communicate with the operating system or an application.

Example: DOS commands

(c) **Text processing languages**

It accepts text files as input, to produce formatted documents as output.

Example: LaTex, Post Script

(d) **HTML - Hypertext Markup Language**

It is used in Internet and WWW (World Wide Web) applications

(e) **XML - extensible Markup Language. It was designed to store and transport data.**

Performance measures of good programming language

- Ease of designing, coding
- Speed
- Debugging
- Maintenance
- Reusability
1.2 INTERNAL REPRESENTATION OF DATA

Inside the computer, all represented data are as 0’s and 1’s. The binary representation is used for internal representation of data. In binary representation only two possible values. They are 0 and 1. It represents two states ON/OFF. Data is stored internally in bits and 8 bits make up 1 byte.

A binary system is based on n bits that can represent \(2^n\) possible values. Using 1 bit 2 possible values are 0 and 1. Using 2 bits 4 possible values are 00, 01, 10 and 11 and so on.

1.3 ALGORITHMS AND FLOWCHARTS

1.3.1 Algorithms

The sequence of steps to be performed in order to solve a given problem by the computer is known as an algorithm. It also refers to logic of the program.

A typical programming task can be divided into two phases.

1. Problem solving phase

It produces an ordered sequence of steps that describe the solution of problem. This sequence of steps is called an algorithm.

2. Implementation phase

Implement the program in some programming language.

Steps in problem solving phase

1. First produce a general algorithm which is called as pseudocode.
   Pseudocode is an information language that helps programmers to develop an algorithm.

2. Refine the pseudocode to produce the step by step detailed algorithm.

Example 1

Write an algorithm to find the largest of two numbers and print the same.

Pseudocode

- Get an input of two numbers say A and B.
- If the first number is greater than second number, print “A is greater” else print “B is greater”
**Detailed algorithm**

Step 1: Read A and B

Step 2: if (A > B) then

    print “A is greater”

else

    print “B is greater”

end if

**1.3.2 Flowchart**

Flowchart is a graphical or symbolic representation of an algorithm. It is the diagrammatic representation of step-by-step solution to a given problem. Table 1.1 shows the list of symbols used in the flowchart.

<table>
<thead>
<tr>
<th>S.No</th>
<th>Name</th>
<th>Symbol</th>
<th>Use in Flowchart</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Oval</td>
<td><img src="symbol.png" alt="Oval" /></td>
<td>Denotes the beginning or end of the program.</td>
</tr>
<tr>
<td>2</td>
<td>Parallelogram</td>
<td><img src="symbol.png" alt="Parallelogram" /></td>
<td>Used for input (read) and output (write) operations.</td>
</tr>
<tr>
<td>3</td>
<td>Rectangle</td>
<td><img src="symbol.png" alt="Rectangle" /></td>
<td>Denotes process to be carried out.</td>
</tr>
<tr>
<td>4</td>
<td>Diamond</td>
<td><img src="symbol.png" alt="Diamond" /></td>
<td>Denotes decision to be made.</td>
</tr>
<tr>
<td>5</td>
<td>Flow Line symbol</td>
<td><img src="symbol.png" alt="Flow Line" /></td>
<td>Used to show the flow of control in the program.</td>
</tr>
</tbody>
</table>
### Basics of Procedure Oriented Programming and Object Oriented Programming

#### 1.4.1 Procedure-Oriented Programming (POP)

In the procedure oriented approach, the problem is viewed as a sequence of jobs, to be done such as reading, calculating and printing. A set of functions can be written to accomplish these tasks. The primary focus is on functions. Any given procedure might be called at any point during a program execution.

Examples of procedure oriented programming languages are C, FORTRAN, Pascal.

---

**Table: Connectors in Flowcharts**

<table>
<thead>
<tr>
<th>S.No</th>
<th>Name</th>
<th>Symbol</th>
<th>Use in Flowchart</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Connector symbol</td>
<td><img src="image" alt="connector symbol" /></td>
<td>A connector symbol is represented by a circle with a letter or digit inside to specify the link.</td>
</tr>
</tbody>
</table>
Characteristics of procedure – oriented programming are

- It focuses on functions rather than data.
- A program is divided into a number of functions and each function has clearly defined purpose.
- Most of the functions share global data.
- Data can move through the program freely from function to function.

1.4.2 Object Oriented Programming (OOPs)

The core of the pure objected oriented programming is to create an object, that has certain properties and methods. In OOPs, programmers define not only the data type of a data structure, but also the type of operations (functions) that can be applied to the data structure. The main idea behind the object oriented approach is to combine process (function) and data into a unit called an object. Hence, it focuses on object rather than procedure.

Examples of object oriented languages are Java, C++, Smalltalk and Python.

Characteristics of Object Oriented Programming languages

- Emphasis on data rather than procedure.
- Programs are divided into entities known as objects.
- Functions that operate on data of an object are tied together in data structure.
- Objects may communicate with each other through functions.

Difference between procedure-oriented programming and object-oriented programming is given in the table 1.2.

<table>
<thead>
<tr>
<th>S.No</th>
<th>Procedure oriented programming</th>
<th>Object oriented programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>The program is divided into small modules called functions.</td>
<td>The program is divided into number of parts called objects.</td>
</tr>
<tr>
<td>2.</td>
<td>Importance is given to functions rather than data.</td>
<td>Importance is given to the data rather than procedures or functions.</td>
</tr>
<tr>
<td>3.</td>
<td>Does not have any access specifier.</td>
<td>It has access specifiers named as public, private and protected.</td>
</tr>
</tbody>
</table>
### 1.5 Introduction to C++

C++ is an object oriented programming language, which was developed by Bjarne Stroustrup at AT & T Bell Lab in New Jersey, USA in the early 1980’s. C++ is an extension of C language with major addition of class constructs from Simula 67 language. Stroustrup initially called a new language as ‘C with Classes’. In 1983, the name was changed to C++. C++ is a superset of C.

\[
C++ = C + \{ \text{classes, objects, Data encapsulation and data hiding, polymorphism, inheritance and dynamic binding} \}
\]

This means that C++ contains all the features of C. In addition, C++ contains the following new features:

- Classes
- Objects
- Data encapsulation and data hiding
- Polymorphism
- Inheritance
- Dynamic Binding
- Message passing

<table>
<thead>
<tr>
<th>S.No</th>
<th>Procedure oriented programming</th>
<th>Object oriented programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.</td>
<td>Most of the functions share global data.</td>
<td>Data is hidden and cannot be accessed by external functions.</td>
</tr>
<tr>
<td>5.</td>
<td>It follows top-down approach.</td>
<td>It follows bottom-up approach.</td>
</tr>
<tr>
<td>6.</td>
<td>Does not have any proper way for hiding data; hence it is less secure.</td>
<td>Data hiding provides more security.</td>
</tr>
<tr>
<td>7.</td>
<td>Overloading is not possible.</td>
<td>Overloading is possible in the form of function overloading and operator overloading.</td>
</tr>
<tr>
<td>8.</td>
<td>Examples are FORTRAN, Pascal, C</td>
<td>Examples of OOP are Smalltalk, C++</td>
</tr>
</tbody>
</table>
1.6 STRUCTURE OF C++

A typical C++ program contains the following section.

<table>
<thead>
<tr>
<th>Documentation section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preprocessor section</td>
</tr>
<tr>
<td>Global Declaration section</td>
</tr>
<tr>
<td>Class declaration and method definition section</td>
</tr>
<tr>
<td>Main function</td>
</tr>
<tr>
<td>Method definition section</td>
</tr>
</tbody>
</table>

1. Documentation section

It contains the comment lines, comments can appear anywhere in a program that is used to explain a point about a code. The comment lines begins with /* and ends with */ or a single set of // is given in the beginning of the line. Comment lines are not executable.

2. Preprocessor section

It is used to link system library files, for defining the macros conditional inclusion. It provides the special instructions to the preprocessor that tells how to prepare the program for compilation. For example, #include<iostream.h>,

    #include<stdio.h> etc.,

3. Global declaration section

In this section, we declare some variables before starting of the main program or outside the main program. These variables are globally declared and used by the main function or sub function. These variables are known as global variable.

4. Class declaration and member function definition section

This section can be considered as sub section for the global declaration section. Class declaration and all methods of that class are defined here. The detailed explanation of this section given in chapter 4.
5. Main function

This is the entry point for all the function. Every method is called indirectly through main function. Class objects are created and the methods are called. The detailed explanation is given in chapter 4.

6. Method definition section

This is the optional section. This method was generally used in C programming.

Program 1.1

// An example for simple C++ program
#include<iostream.h>
void main ()
{
    cout <<“Welcome to C++”;
}

Output:
Welcome to C++

An example with classes and objects are discussed in chapter 4.

1.7 KEYWORDS

Keywords are reserved words, reserved identifiers which cannot be used as names for the program variable and other user defined program elements.

Table 1.3 gives the complete set of C++ keywords. Most of them are common to C and C++.

<table>
<thead>
<tr>
<th>Table 1.3 C++ Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>asm</td>
</tr>
<tr>
<td>auto</td>
</tr>
<tr>
<td>break</td>
</tr>
<tr>
<td>case</td>
</tr>
</tbody>
</table>
1.8 IDENTIFIERS

Identifiers are names given to variables, functions, arrays, classes, etc. created by the programmer. They are the basic requirement for any programming language.

Rules of identifiers

1. C++ is case-sensitive so that uppercase letters and lower case letters are different.
2. The name of identifier cannot begin with a digit. However, underscore can be used as first character while declaring the identifier.
3. Only alphabetic characters, digits and underscore (_) are permitted for declaring identifier.
4. Keywords cannot be used as identifier.

Example

```cpp
int x; // valid statement, identifier of integer type
int number1; // valid statement, identifier of integer type
float Inum; // Invalid statement
```

Program 1.2

// Addition of two numbers
```cpp
#include <iostream.h>

void main()
{
}
```
```cpp
int a, b, c; // a, b and c are integer variables
cout << "Enter two numbers" endl;
cin >> a >> b;
c = a + b;
cout << "The result is" << c;
}
```

Output:

Enter two numbers
20 30
The result is 50

1.9 DATA TYPES

In any programming language, we need to use various variable to store various information. C++ provides rich set of built in data types and user defined data types.

1.9.1 Built-in Data Types

C++ compiler support all the built-in data types are given in the table 1.4

<table>
<thead>
<tr>
<th>Type</th>
<th>Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>bool</td>
</tr>
<tr>
<td>Character</td>
<td>char</td>
</tr>
<tr>
<td>Integer</td>
<td>int</td>
</tr>
<tr>
<td>Real/Floating point</td>
<td>float</td>
</tr>
<tr>
<td>Double floating point</td>
<td>double</td>
</tr>
</tbody>
</table>

The built in types can be modified by using one or more of the following type modifiers.

- Signed
- Unsigned
- Short
- Long
The following table 1.5 shows the typical size and range of built in data type.

<table>
<thead>
<tr>
<th>SNo</th>
<th>Data type</th>
<th>Size (in Bytes)</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>bool</td>
<td>1 set</td>
<td>True or false</td>
</tr>
<tr>
<td>2</td>
<td>char</td>
<td>1</td>
<td>− 127 to 127 or 0 to 255</td>
</tr>
<tr>
<td>3</td>
<td>unsigned char</td>
<td>1</td>
<td>0 to 255</td>
</tr>
<tr>
<td>4</td>
<td>signed char</td>
<td>1</td>
<td>− 127 to 127</td>
</tr>
<tr>
<td>5</td>
<td>int</td>
<td>2</td>
<td>− 32768 to 32767</td>
</tr>
<tr>
<td>6</td>
<td>unsigned int</td>
<td>2</td>
<td>0 to 65535</td>
</tr>
<tr>
<td>7</td>
<td>signed int</td>
<td>2</td>
<td>− 31768 to 32767</td>
</tr>
<tr>
<td>8</td>
<td>short int</td>
<td>2</td>
<td>− 31768 to 32767</td>
</tr>
<tr>
<td>9</td>
<td>long int</td>
<td>4</td>
<td>− 2147483648 to 214783647</td>
</tr>
<tr>
<td>10</td>
<td>float</td>
<td>4</td>
<td>3.4 E − 38 to 3.4 E + 38</td>
</tr>
<tr>
<td>11</td>
<td>double</td>
<td>8</td>
<td>1.7 E − 308 to 1.7 E + 308</td>
</tr>
<tr>
<td>12</td>
<td>long double</td>
<td>10</td>
<td>3.4 E − 4932 to 1.1 E + 4932</td>
</tr>
</tbody>
</table>

Program 1.3

// program to find area of circle
#include <iostream.h>

void main()
{
    int r;
    float pi = 3.14, area;
    cout << "Enter the value of r";
    cin >> r;
    area = pi * r * r;
    cout << "The area of circle is" << area;
}
1.9.2 Enumerated Data Type

An enumerated data type is an user-defined type which provides a way of attaching names to the number.

An enumerated type declares an optional type name and a set of zero or more identifiers that can be used as values of the type.

Each enumerator is a constant whose type is the enumeration, enum is a keyword used to create a enumeration type.

Syntax

```cpp
enum enum_name { list_of_name } var_list;
```
where enum is a keyword and enum_name is a name of the enumerated type, list_of_names refers to list of constants, var_list refers to variable created for given enumerator. For example

```
enum color {red,green,blue};
c = red;
```

Here, the name of the enumerator is color, the variable c is the type of color. Finally c is assigned to the value “red”.

By default, the enumerators are assigned integer values starting with O for the first enumerator, 1 for the second, and so on.

We can override the default by explicitly assigning integer values to the enumerators. For example,

```
enum color { red = 5, green = 8, blue };
```

In the above line, red is 5, green is 8 and then blue is 9. i.e. the subsequent initialized enumerators are larger by one than their predecessors.

**Program 1.5**

// Program for enumeration datatype

```
#include <iostream.h>
enum position { off, on };
void main()
{
    position x;
    x=off;
    cout << "Position is" << x;
}
```

**Output**

Position is 0
1.9.2 Strings

Strings are actually one-dimensional array of characters terminated by a null character ‘\0’. Thus a null-terminated string contains the characters that comprise the string followed by a null.

The following declaration and initialization creates a string consisting of the word “Hello”. To hold the null character at the end of the array, the size of the character array containing the string is one more than the number of characters in the word “Hello”.

```c
char greeting [6] = { 'H', 'e', 'l', 'l', 'o', '\0' };
```

(Or)

```c
char greeting [ ] = "Hello";
```

The memory representation of the above defined string is as follows:

<table>
<thead>
<tr>
<th>index</th>
<th>variable</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>H</td>
<td>0×6378</td>
</tr>
<tr>
<td>1</td>
<td>e</td>
<td>0×6379</td>
</tr>
<tr>
<td>2</td>
<td>l</td>
<td>0×6380</td>
</tr>
<tr>
<td>3</td>
<td>l</td>
<td>0×6381</td>
</tr>
<tr>
<td>4</td>
<td>o</td>
<td>0×6382</td>
</tr>
<tr>
<td>5</td>
<td>\0</td>
<td>0×6382</td>
</tr>
</tbody>
</table>

The C++ compiler automatically places the ‘\0’ at the end of the string when it initializes the array.

Program 1.6

// Program for string data type
#include <iostream.h>

void main()
{

char greeting[6] = { 'H', 'e', 'l', 'l', 'o', '\0' };

cout << “Greeting message” << greeting;
}

Output

Greeting message Hello
1.10 CONSTANTS AND VARIABLES

1.10.1 Constants

Constants refers to fixed values that cannot be changed during the execution of a program.

Examples

456 // Integer constant
45.6 // Floating point constant
‘A’ // Character constant
“C++” // String constant

1.10.2 Variables

A variable is an entity whose value can be changed during the execution of the program, a variable can hold a value during the execution of the program. Definition of the variable is associated with the memory location of the variable name.

Syntax

Datatype varname1, varname2, varnameN;

For example,

int a; // a is an integer variable
float total; // total is a float variable

Operands can be variables, expression or a literal value where operation instructions are specified by operators.

Variable Initialization

A variable can be assigned with the value during the definition or execution of the program. The assignment operator (=) is used in both cases.

Syntax of variable initialization during the definition:

datatype variable_name = constant_value;
datatype variable_name (constant_value);
For example,

```
int x = 100; // (or) int x (100);
float c = 185.86; // (or) float c (185.86);
```

Syntax of variable initialization during the runtime is as follows.
```
variable_name = expression;
```

The expression can be a constant, variable_name or a combination of variables and constants connected using operators (i.e. mathematical expression).

**Example**
```
a = 10;
a = b;
a = x + y - 3;
```

**Program 1.7**

*Program for variable initialization*

```c++
#include <iostream.h>
void main()
{
    int x,y; // variable declaration
    int z = 500; // variable definition and initialization
    // initialization during run time
    x = z;
    y = x + z;
    cout << "x =" << x << "n";
    cout << "y =" << y << "n";
    cout << "z =" << z << "n";
}
```

**Output**
```
x = 500
y = 1000
z = 500
```
1.11 OPERATORS

An operator is a symbol that “operates” on one or more expressions, producing a value that can be assigned to a variable. C++ operators are special characters which instruct the compiler to perform the (assigned) designated operation on the operands.

List of operators

1. Assignment operations
2. Arithmetic operators
3. Relational operators
4. Logical operators
5. Incremental and decrement operators
6. Conditional operators

1.11.1 Assignment operator

The equal (=) sign is used for assigning a value to a variable. The Syntax is:

```plaintext
variable = expression;
```

The left hand side should be a variable and the right hand side should be a valid expression.

Example

```plaintext
a = 586; // rvalue is constant
b = z+10*a;// rvalue is expression
c = sqrt(5); // rvalue is function
```

1.11.2 Arithmetic operators

Arithmetic operators are used to perform mathematical calculations like addition, subtraction, multiplication, division and modulus. Table 1.6 shows the list of arithmetic operators.
Table 1.6 Arithmetic operators

<table>
<thead>
<tr>
<th>S.No</th>
<th>Operator</th>
<th>Operation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>+</td>
<td>Addition</td>
<td>A + B</td>
</tr>
<tr>
<td>2</td>
<td>–</td>
<td>Subtraction</td>
<td>A – B</td>
</tr>
<tr>
<td>3</td>
<td>*</td>
<td>Multiplication</td>
<td>A * B</td>
</tr>
<tr>
<td>4</td>
<td>/</td>
<td>Division</td>
<td>A/B</td>
</tr>
<tr>
<td>5</td>
<td>%</td>
<td>Modulus</td>
<td>A % B</td>
</tr>
</tbody>
</table>

Rules for Arithmetic operators

1. C++ allows only one variable on left hand side of = operator.
   For example,
   
   c = a * b is legal, but a * b = c is not legal.

2. Arithmetic operations are performed on the ASCII values of the characters and not on characters themselves.

3. Operators must be explicitly written.

Program 1.8

```c
// Program to use arithmetic operator
#include <iostream.h>

void main()
{
    int x, y, add, sub, mul, mod;
    float div;
    cout << "Enter the value of x and y":
    cin >> x >> y;
    add = x + y;
    sub = x - y;
    mul = x * y;
    div = x/y;
    mod = x % y;
}```
cout << "Addition is:" << add;
cout << "Subtraction is:" << sub;
cout << "Multiplication is:" << mul;
cout << "Division is:" << div;
cout << "Remainder is:" << mod;
}

Output
Enter the value of x and y 40 20
Addition is: 60
Subtraction is: 20
Multiplication is: 800
Division is: 2
Remainder is: 0

1.11.3 Relational operations
Relational operators are used to compare two or more operands. Operands can be variables, constants or expression. Table 1.7 shows the relational operators in C++.

<table>
<thead>
<tr>
<th>S.No</th>
<th>Operators</th>
<th>Operation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>&gt;</td>
<td>is greater than</td>
<td>x &gt; y</td>
</tr>
<tr>
<td>2.</td>
<td>&lt;</td>
<td>is less than</td>
<td>x &lt; y</td>
</tr>
<tr>
<td>3.</td>
<td>&gt;=</td>
<td>is greater than or equal to</td>
<td>x &gt;= y</td>
</tr>
<tr>
<td>4.</td>
<td>&lt;=</td>
<td>is less than or equal to</td>
<td>x &lt;= y</td>
</tr>
<tr>
<td>5.</td>
<td>==</td>
<td>is equal to</td>
<td>x == y</td>
</tr>
<tr>
<td>6.</td>
<td>!=</td>
<td>is not equal to</td>
<td>x != y</td>
</tr>
</tbody>
</table>
Program 1.9

// Program to find greatest of two numbers
#include <iostream.h>

void main()
{
    int a, b;
    cout "Enter the value of a and b":;
    cin >> a >> b;
    if (a > b)
    {
        cout << "a is greater"
    }
    else
    {
        cout << "b is greater"
    }
}

Output
Enter the value of a and b: 30 50
b is greater

1.11.4 Logical operators

Logical operators used to test more than one condition and make decision. It also used to combine the results of two or more conditions. Table 1.8 shows the logical operators.
Table 1.8 Logical operator

<table>
<thead>
<tr>
<th>S.No</th>
<th>Operators</th>
<th>Operation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>&amp;&amp;</td>
<td>logical AND</td>
<td>$(m &gt; 10) \land (n &lt; 8)$</td>
</tr>
<tr>
<td>2.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td></td>
<td></td>
<td>logical NOT</td>
</tr>
</tbody>
</table>

Program 1.10

// Program to find greatest of 3 numbers

#include <iostream.h>

void main()
{
    int x, y, z;
    cout "Enter the value of x, y and z"; << endl;
    cin >> x >> y >> z;
    if ((x > y) && (x > z))
    {
        cout < "x is greater than y and z";
    } else if(y > z)
    {
        cout < "y is greater than x and z";
    } else
    {
        cout < "z is greater than x and y";
    }
}

Output

Enter the value of x, y and z
38 50 62

z is greater x and y
1.11.5 Increment and decrement operators

Increment operators are used to increase the value of the variable by one and decrement operators are used to decrease the value of the variable by one.

++ is the increment operator

-- is the decrement operator

Syntax

Increment operator:

```cpp
++ var_name; //preincrement
```

(or)

```cpp
var_name++; //postincrement
```

Decrement operator:

```cpp
-- var_name; //predecrement
```

(or)

```cpp
var_name--; //post decrement
```

Difference between increment and decrement operators in C++ is given in the table 1.9

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pre increment operator (++i)</td>
<td>value of i is increment before assigning it to variable i</td>
</tr>
<tr>
<td>post increment operator (i++)</td>
<td>value of i is incremented after assigning it to variable i</td>
</tr>
<tr>
<td>pre decrement operator (--i)</td>
<td>value of i is decremented before assigning it to variable i</td>
</tr>
<tr>
<td>post decrement operator (i--)</td>
<td>value of i is decremented after assigning it to variable i</td>
</tr>
</tbody>
</table>
Program 1.11

//program for increment and decrement operator
#include <iostream.h>
void main()
{
    int x=10;
    int c;
    c = x++; // post increment
    cout <<"value of c is" << c;
    cout <<"value of x is" << x;
    c = ++ x; // pre increment
    cout <<"value of c is" << c;
    c = --x; // pre decrement
    cout <<"value of c is" << c;
    c = x--; // post decrement
    cout << "value of c is" << c;
}

Output
    Value of c is 10
    Value of x is 11
    Value of c is 12
    Value of c is 11
    Value of c is 11

1.11.6 Conditional operator/Ternary operator

Conditional operator checks the condition and executes the statement depending on the result of condition.
Syntax

Condition? Exp1: Exp2

The ‘?:’ operator acts as ternary operator. It evaluates the condition, if it is true then exp1 is evaluated, else exp2 is evaluated.

For example,

\[ X = (a > b) \? a : b; \]

Assume that \(a = 50, b = 30\).

Given condition is \((a > b)\), since the condition is true, value of \(a\) is assigned to \(x\)

Program 1.12

//program to find smallest of two numbers

#include <iostream.h>

void main()
{

    int x, a = 50, b = 30;
    x = (a < b) \? a : b;
    cout << "The value of x is" << x;
}

Output

The value of x is 30

C++ Operator precedence:

The following table 1.10 lists the precedence of C++ operators. Operators are listed top to bottom, in descending precedence.
## Table 1.10 Operator precedence in C++

<table>
<thead>
<tr>
<th>Precedence</th>
<th>Operator</th>
<th>Description</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>::</td>
<td>Scope resolution</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>x++ x--</td>
<td>Post fix increment/decrement</td>
<td>L → R</td>
</tr>
<tr>
<td></td>
<td>type() type { }</td>
<td>Functional cast</td>
<td></td>
</tr>
<tr>
<td></td>
<td>x( )</td>
<td>Function call</td>
<td></td>
</tr>
<tr>
<td></td>
<td>x[ ]</td>
<td>Subscript</td>
<td></td>
</tr>
<tr>
<td></td>
<td>. →</td>
<td>Member access</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>++x --x</td>
<td>Pre increment/decrement</td>
<td>R → L</td>
</tr>
<tr>
<td></td>
<td>+x -x</td>
<td>Unary plus and minus</td>
<td></td>
</tr>
<tr>
<td></td>
<td>! ~</td>
<td>Logical NOT and bitwise NOT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(type)</td>
<td>C - style cast</td>
<td></td>
</tr>
<tr>
<td></td>
<td>*x</td>
<td>Indirection</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&amp;x</td>
<td>Address-of</td>
<td></td>
</tr>
<tr>
<td></td>
<td>size of</td>
<td>Size-of</td>
<td></td>
</tr>
<tr>
<td></td>
<td>new new [ ]</td>
<td>Dynamic memory allocation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>delete delete [ ]</td>
<td>Dynamic memory eallocation</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>.* → *</td>
<td>Pointer-to-member</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>a * b a/b a % b</td>
<td>Multiplication, division and remainder</td>
<td>L → R</td>
</tr>
<tr>
<td>6</td>
<td>a + b a - b</td>
<td>Addition and subtraction</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>&lt;&lt; &gt;&gt;</td>
<td>Bitwise left shift and right shift</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>&lt; &lt;=</td>
<td>For relational operator</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&gt; &gt;=</td>
<td>&lt;, &lt;=, &gt; and &gt;=</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>== !=</td>
<td>For relational operator</td>
<td></td>
</tr>
<tr>
<td></td>
<td>= and #</td>
<td>= and #</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>a &amp; b</td>
<td>Bitwise AND</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>^</td>
<td>Bitwise XOR (exclusive or)</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td>Bitwise OR (inclusive or)</td>
</tr>
<tr>
<td>13</td>
<td>&amp; &amp;</td>
<td>Logical AND</td>
<td></td>
</tr>
</tbody>
</table>
### Precedence and Associativity

<table>
<thead>
<tr>
<th>Precedence</th>
<th>Operator</th>
<th>Description</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>a ? b : c</td>
<td>Ternary condition</td>
<td></td>
</tr>
<tr>
<td></td>
<td>=</td>
<td>Assignment</td>
<td>R → L</td>
</tr>
<tr>
<td></td>
<td>+= -=</td>
<td>Shorthand Assignment</td>
<td></td>
</tr>
<tr>
<td></td>
<td>*= /= %=</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;&lt;= &gt;&gt;=</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&amp; = ∧ = !</td>
<td>=</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>,</td>
<td>Comma</td>
<td>L → R</td>
</tr>
</tbody>
</table>

Note:

R → L (Right to Left)

L → R (Left to Right)

Operator precedence determines which operator is performed first in an expression with more than one operators with different precedence. For example 5 + 10 * 20 is calculated as 5 + (10 * 20) and not as (5 + 10) * 20.

Associativity is used when two operators of same precedence appear in an expression. Associativity can be either Left to Right or Right to Left. For example ‘*’ and ‘/’ have same precedence and their associativity is Left to Right, so the expression “100/10 * 10” is treated as “(100/10) * 10”.

Precedence and Associativity are two characteristics of operators that determine the evaluation order of subexpressions in absence of brackets.

Example for evaluating an expression

Let X = 2, Y = 5 then the value of the expression

(((Y - 1) / X) * (X + Y)) is calculated as:-

(Y - 1) = (5 - 1) = 4 = A1

(A1/X) = (4/2) = 2 = A2

(X + Y) = (2 + 5) = 7 = A3

(A2 * A3) = (2 * 7) = 14

The evaluations are made according to the priority value.
Program 1.13
// Example for operator precedence
#include <iostream.h>

void main()
{
    int x = 20;
    int y = 10;
    int z = 15;
    int a = 5;
    int e;
    e = (x + y) * z / d; // (30 * 15) / 5
    cout << "value of (x + y) * z/d is" << e;
    e = ((x + y) * c) / d; // (30 * 15) / 5
    cout << "value of ((x + y) * c) / d is" << e;
}

Output
Value of (x + y) * z/d is 90
Value of ((x + y) * c) / d is 90

1.12 SIMPLE STATEMENTS

The body of C++ functions (including the main function) are made up of statements. These can either be simple statements that do not contain other statements, or compound statements that have other statements inside of them.

The simplest kind of statement in C++ is an expression (followed by a semicolon, the terminator for all simple statements). Its value is computed and discarded.
Example

\[ x = 2; \ // \ an \ assignment \ statement \]
\[ 2 + 3; \ // \ has \ no \ effect - \ it \ will \ be \ discarded \ by \ smart \ compilers \]
\[ \text{puts} ("hello"); \ // \ a \ statement \ containing \ a \ function \ call \]

**Compound statements**

Any block of the code is written or embedded inside the pair of the curly braces. No semicolon is needed at the end of a block.

**Program 1.14**

//program for statements in c++

```cpp
#include <iostream.h>
void main()
{
    int num = 50; // simple statement
    if (num > 0)
    {
        cout << \\
        "Number is positive"; // compound statement
        cout << \\
        "This is the example of compound statement";
    }
}
```

**Output**

Number is positive
This is the example of compound statement

In the above program, if the condition specified inside the ‘if block’ is true then code block written inside the pair of curly braces will be executed.

**1.13 INPUT AND OUTPUT STREAM (I/O Stream)**

In C++, the input and output occurs in streams, which are sequences of bytes. If the bytes flow from devices like keyboard to main memory, this is called as input operation. If the bytes flow from main memory to a device like a display screen, this is called as output operation.
Input stream

- The standard input stream used in C++ is cin. cin is a predefined object available in istream class. The cin object is attached to the standard input device, which is usually a keyboard. The object is used along with stream extraction operator, which is written as >> (Two greater than signs).

For example,

```cpp
int a;

cin >> a; // Reads an integer and assign it to a
```

Output stream

The standard output stream used in C++ is cout.

1. cout is the predefined object available in ostream class.
2. The cout object is attached to the standard output device, which is usually a monitor/display screen.
3. This object is used along with the stream insertion operator, which is written as <<. (Two less than signs).

For example,

```cpp
cout << "Hello Friends" // The given string is printed.
```

Program 1.15

//program to find multiplication of two numbers
#include <iostream.h>

void main()
{
    int x,y result;
    cout << "Enter the value of x and y\n";
    cin >> x;
    cin >> y;
    result = x*y;
    cout << "The multiplication of two number is" << result;
}
Output
Enter the value of x and y
5  8
The multiplication of two number is 40

Questions
1. What is a program?
2. What is an algorithm?
3. Draw a flowchart and explain the different symbols used in the flowchart.
4. Write an algorithm to find maximum of three numbers.
5. Compare object oriented programming with procedure oriented programming.
6. Explain the structure of C++ program with an example.
7. Explain the various datatypes supported by C++.
8. What are operators? Explain in detail with suitable examples.
9. Write notes on the following: (a) Keywords (b) Identifiers
10. Explain conditional operator with an example.
11. What do you mean by statements, what are the types of statements.
12. Write the code to display the statement “End of the page”.
13. What is a header file? How is it related to library file.
14. Arrange in order of precedence (highest first) the following kind of operators - logical, unary; arithmetic, assignment, relational, conditional.
15. Write a short notes on 1. Input stream 2. Output stream
16. Write a program to convert Fahrenheit to Celsius. Use floating point numbers.
17. What do you mean by enumerated data type in C++.