
MODULE - I

Introduction to Object Oriented Programming with C++

Introduction to Object Oriented Programming with C++

Module Description

This module focuses on the overview of objects which is an integral component of object oriented programming. It will further explain the basic concepts, advantages and disadvantages of objects in object oriented programming, C++ data types, and operators, expressions, Decision loops, arrays, strings, structures and Unions.

By the end of this module, students will learn the fundamentals of object oriented programming and C++ syntax to write some sample programs on C++. In addition to these skills, students are also able to intelligently compile and execute the program and debug the errors.

Chapter 1.1

An Overview of Object Oriented Programming

Chapter 1.2

Arrays, String and Structures

Chapter Table of Contents

Chapter 1.1

An Overview of Object Oriented Programming

Aim	1
Instructional Objectives	1
Learning Outcomes	1
1.1.1 Introduction	2
Self-assessment Questions	3
1.1.2 Evolution of Programming Methodologies	4
Self-assessment Questions	6
1.1.3 Procedure Oriented Programming	7
Self-assessment Questions	8
1.1.4 Object Oriented Programming	9
(i) Basics of OOP	11
(ii) Characteristics of OOP	13
(iii) Merits and Demerits of OOP	13
(iv) Procedure Oriented Programming vs. Object Oriented Programming	15
Self-assessment Questions	15
1.1.5 Introduction to C++	16
(i) C++ Data Types	16
(ii) Operators and Keywords	19
(iii) Expressions	24
(iv) Input and Output	24
Self-assessment Questions	27
1.1.6 Decision and Loop	28
(i) Conditional Statements in C++	28
(ii) Looping Statements	32
Self-assessment Questions	40
Summary	41
Terminal Questions	42
Answer Keys	43
Activities	44
Bibliography	45
e-References	45
External Resources	45
Video Links	45



Aim

To explain the fundamentals of Object oriented programming with C++ programming



Instructional Objectives

After completing this chapter, you should be able to:

- Discuss the evolution of programming methodologies
- Compare Procedure oriented programming and Object oriented programming
- Describe the features of OOPS
- Illustrate C++ data types, Operators and expressions
- Design the structure of C++ program and elaborate it
- Design and develop programs using looping and conditional statements



Learning Outcomes

At the end of this chapter, you are expected to:

- Illustrate the programming methodologies of C++
- Identify merits and Demerits of OOP
- Define Object oriented programming with its characteristics
- Elaborate data types of C++
- Design and implement some sample C++ programs
- Elaborate abstraction, encapsulation, inheritance and polymorphism

1.1.1 Introduction

A programming language is a language specifically designed in the form of instructions which can be used to make computer work as per the wish of the programmer. Programming languages are used to create programs that control the behavior of a system, to express algorithms, or can be used as a mode of human communication.

Usually, programming languages have a vocabulary of syntax and semantics for instructing a computer to perform specific tasks. The term 'programming language' usually refers to high-level languages such as BASIC, C, C++, COBOL, FORTRAN, ADA, and PASCAL, to name a few. All these languages have a unique set of keywords and a special syntax for organizing program instructions. These languages are categorised into high level, machine level and assembly level languages based on type of language used in programming.

High-level languages are easy to read and understand, while the computer understands the machine language that consists of only numbers. Different types of central processing unit (CPU) have their unique machine languages.

Assembly language is a type of language that exists in between machine languages and high-level languages. Assembly languages are similar to machine languages, but are much easier to program in because they allow a programmer to substitute names for numbers.

However, irrespective of what language uses, the program written using any programming language has to be converted into machine language so that the computer can understand the same. There are two ways to do this- compile the program or interpret the program.

The determination of the language is dependent on the following factors:

- The type of computer (microcontroller, microprocessor, etc.) on which the program has to be executed.
- The type of program (system program, application program, etc.).
- The expertise of the programmer. That is, the proficiency level of a programmer in a particular language.

For example, FORTRAN is particularly good for processing numerical data but it doesn't lend itself very well to organizing large programs. PASCAL can be used for writing well-structured

and readable programs but it is not flexible as C language. C++ goes one step ahead of C by incorporating powerful object oriented features but is complex and difficult to learn.



Self-assessment Questions

- 1) Programming languages have a vocabulary of _____ and _____ for instructing a computer to perform specific tasks.
- 2) _____ language is used for processing numerical data.
 - a) PASCAL
 - b) FORTAN
 - c) C
 - d) C++
- 3) Programming languages have a unique set of _____ and a special _____ for organizing program instructions.

1.1.2 Evolution of Programming Methodologies

Earlier, the machine language was used to design little and basic programs. Next the Assembly Language was developed which was utilized for planning bigger programs. Both machine language and Assembly language is machine dependent. Next the Procedural Programming Approach was developed which empowered us to compose bigger and hundred lines of code. At that point in 1970, another programming approach called Structured Programming Approach was created for outlining medium estimated programs. In 1980's the extent of programs continued expanding, so another approach known as OOP came into existence.

The software evolution has different phases or layers with each layer representing an improvement over the previous layer. The below Figure 1.1.1 depicts the evolution of OOP.

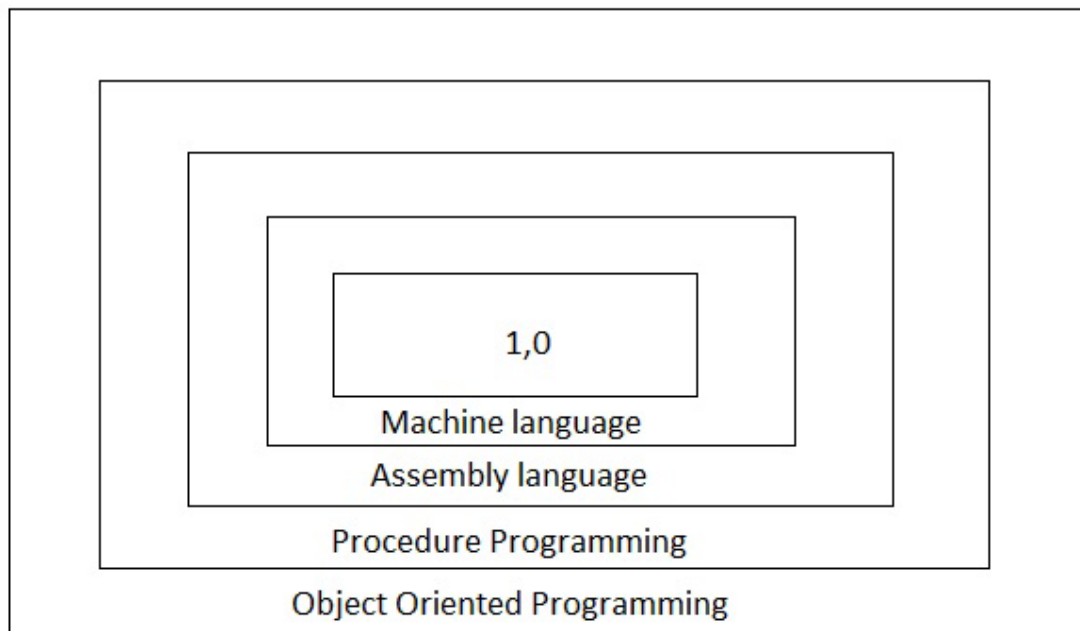


Figure 1.1.1: Evolution of OOP

The various approaches for programming are:

1. Monolithic Programming Approach
2. Procedural Programming Approach
3. Structured Programming Approach
4. Object Oriented Programming Approach

1. Monolithic Programming Approach: In this approach, the program comprises of grouping of statements that change information. All statements of the program are Global all through the entire program. The program control is accomplished using jumps *i.e.*, goto statements. In this approach, code is copied every time since there is no backing for the function. Information is not completely shielded as it can be accessed from any part of the program. So this approach is helpful for planning little and basic programs. The programming languages like ASSEMBLY and BASIC are based on this approach.

- The machine language (0's, 1's) is only understood by the machine. This is low-level language.
- Assembly language is a middle level language and uses mnemonics to write programs. Mnemonics are related to memory which can be understood by people and machine.

Let us look at an example of assembly language code below:

Addition of two numbers in assembly language using 8086

```
ASSUME CS:CODE, DS:DATA
DATA    SEGMENT
        ORG 3000H
        NUM1 DB 22H
        NUM2 DB 32H
        SUM DB 00H
DATA    ENDS
CODE    SEGMENT
        ASSUME CS:CODE, DS:DATA
START:  MOV AX, DATA
        MOV DS, AX
        MOV AL, NUM1
        MOV BL, NUM2
        ADD AL, BL
        MOV SUM, AL
        MOV AH, 4CH
        INT 21H
CODE    ENDS
        END START
```

2. Procedural Programming Approach: This approach follows a top down approach. In this approach, a program is partitioned into functions that play out a particular task. Information is global and all the functions can get hold of the global information. Program flow control is accomplished through function calls and goto statements. This approach keeps away from reiteration of code which is the principle disadvantage of Monolithic Approach. The basic downside of Procedural Programming Approach is that information

is not secured in light of the fact that information is global and can be accessed by any function. This approach is utilized for the most part for medium sized applications. The programming languages: FORTRAN and COBOL are based on this approach.

3. **Structured Programming Approach:** The basic guideline of structured programming approach is to isolate a program in functions and modules. The utilization of modules and functions makes the program more comprehensive (justified). It composes cleaner code and keeps up control over every function. This approach offers significance to functions as opposed to data. It concentrates on the advancement of expansive programming applications. The programming languages: PASCAL and C are based on this approach.
4. **Object Oriented Programming Approach:** The OOP approach appears to expel the disadvantage of ordinary approaches. The basic guideline of the OOP approach is to consolidate both data (information) and functions so that both can work as a single unit. Such a unit is called an Object. This approach secures information too. At present, this approach is utilized for most of applications. The programming languages: C++, PYTHON and JAVA are based on this approach. We can compose lengthy code using this approach.



Self-assessment Questions

- 4) A language understood by the machine in terms of 0's and 1's is called as _____
 - a) High level
 - b) Machine level
 - c) Assembly level
 - d) English
- 5) Which of the following programming approaches is more secure with respect to program data and functions?
 - a) Monolithic
 - b) Object-oriented
 - c) Structural
 - d) Procedural
- 6) Object-oriented programming used to solve _____ problems.
 - a) Mathematical
 - b) Real world
 - c) Mathematical and real world
 - d) Logical

1.1.3 Procedure Oriented Programming

The high level languages such as COBOL, FORTRAN and C are commonly known as procedure oriented languages. In the procedure oriented programming, each task is partitioned into sub-tasks and then solved in a sequence; such as reading the input, processing the necessary computations and printing the result. Flowchart is used to organize actions and shows the flow of control in a program. Flowcharts are diagrams which represent a process or algorithm using different symbols like boxes containing information about steps or a sequence of steps. At a point when a program increases in size, we utilize the rule of partitioning a program into smaller fragments called a procedure or function. In a multifunction program, numerous critical data items are put in global data. So that it might be used by any function when required. Every function may have its own native data.

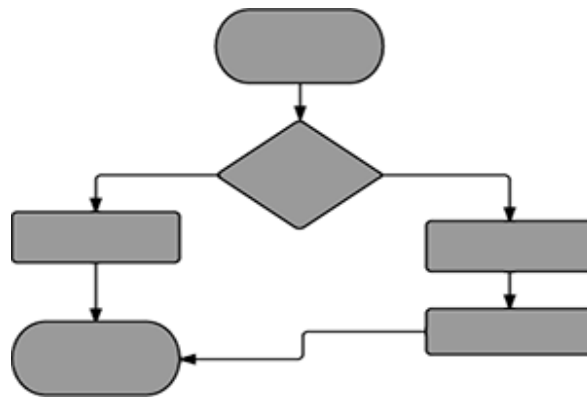


Figure 1.1.2: Flowchart

Example:

```
inti=4;           /* Global definition */

main()
{
    i++;           /* global variable */
    func
}

func()
{
    inti=10;       /* Local declaration */
    i++;           /* Local variable */
}
```

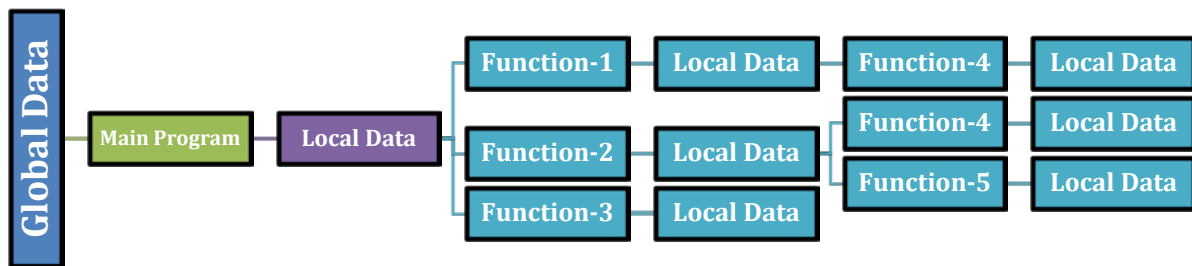


Figure 1.1.3: Relationship between Data and Function

Characteristic of Procedure - Oriented language:

- Procedure oriented programming languages follow a top down approach of problem solving.
- It focuses on which things are to be done as per priority.
- When a program size increases, then it is divided into smaller fragments called functions which reduce the load on the system.
- Since most of the data is made global in a program, the function shares the program data.
- With the above characteristic, it makes easier for data to be moved from one function to another in an efficient manner and also aids the function to transform or convert data from one type to another type.



Self-assessment Questions

- 7) _____programming emphasizes on doing things (algorithms).
 - a) Procedure oriented
 - b) C
 - c) Object oriented
 - d) C++
- 8) Most of the function share _____data.
- 9) Procedure oriented programming design is_____ approach.
 - a) Top down
 - b) Bottom up
 - c) Modular
 - d) System

1.1.4 Object Oriented Programming

Object oriented programming (OOP) is a programming style that is associated with the concept of OBJECTS, which consists of data items and member functions which manipulates these data items.

Objects are instances of classes and are utilized to interface amongst each other to make applications. Instances are the objects of class on which we work with. C++ can be considered to be a C language with classes. In C++ everything rotates around objects and classes, which have their methods & data items to work with.

Consider the following example:

```
Class Student    --- Class Definition
{
    int rollno;
    int marks;
    public:
        void display(int roll)
        {
            cout<<"Rollno: "<<marks;
        }
};

void main()
{
    Student std;    --Object of Class
    std.display();
}
```

Here, Student is a user defined data type called class.

In main() function, an object which is also a user defined data type, std is created from class Student.

The object of class *i.e.*, std of Student becomes the instance of the class.

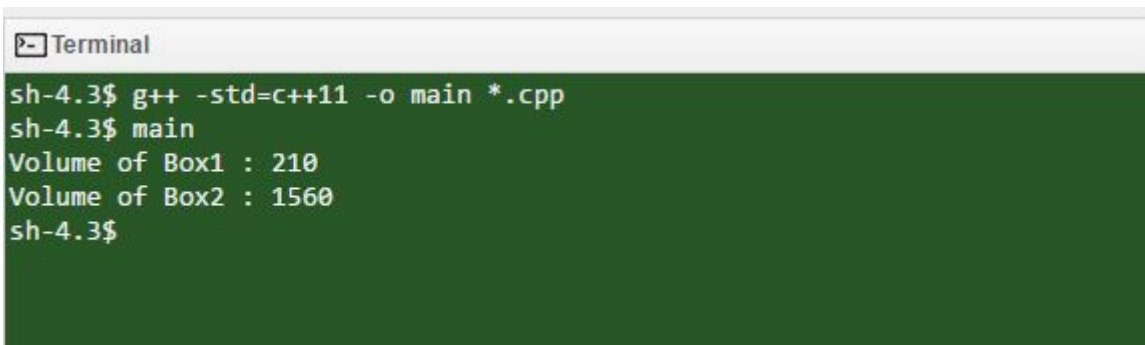
General Structure of C++ program:

```
#include<iostream>
usingnamespacestd;
// main() is where program execution begins.
int main()
{
    cout<<"Hello World";// prints Hello World
    return0;
}
```

The below example explains objects and its instances:

```
1  #include <iostream>
2
3  using namespace std;
4
5  class Box
6  {
7      public:
8          double length;    // Length of a box
9          double breadth;   // Breadth of a box
10         double height;    // Height of a box
11     };
12
13     int main( )
14     {
15         Box Box1;          // Declare Box1 of type Box
16         Box Box2;          // Declare Box2 of type Box
17         double volume = 0.0; // Store the volume of a box here
18
19         // box 1 specification
20         Box1.height = 5.0;
21         Box1.length = 6.0;
22         Box1.breadth = 7.0;
23
24         // box 2 specification
25         Box2.height = 10.0;
26         Box2.length = 12.0;
27         Box2.breadth = 13.0;
28         // volume of box 1
29         volume = Box1.height * Box1.length * Box1.breadth;
30         cout << "Volume of Box1 : " << volume << endl;
31
32         // volume of box 2
33         volume = Box2.height * Box2.length * Box2.breadth;
34         cout << "Volume of Box2 : " << volume << endl;
35         return 0;
36     }
```

Output:



```
Terminal
sh-4.3$ g++ -std=c++11 -o main *.cpp
sh-4.3$ main
Volume of Box1 : 210
Volume of Box2 : 1560
sh-4.3$
```

Let us consider the above program in order to understand objects clearly. Here, Box is considered to be a class, Box1 and Box2 are the objects of the class Box which are used to get length, breadth and height of two different box specifications.

(i) Basics of OOP

Some of the main features of object oriented programming which you will be using in C++ are:

1. Objects
2. Classes
3. Abstraction
4. Encapsulation
5. Inheritance
6. Overloading
7. Exception Handling

They are depicted in the below Figure 1.1.4

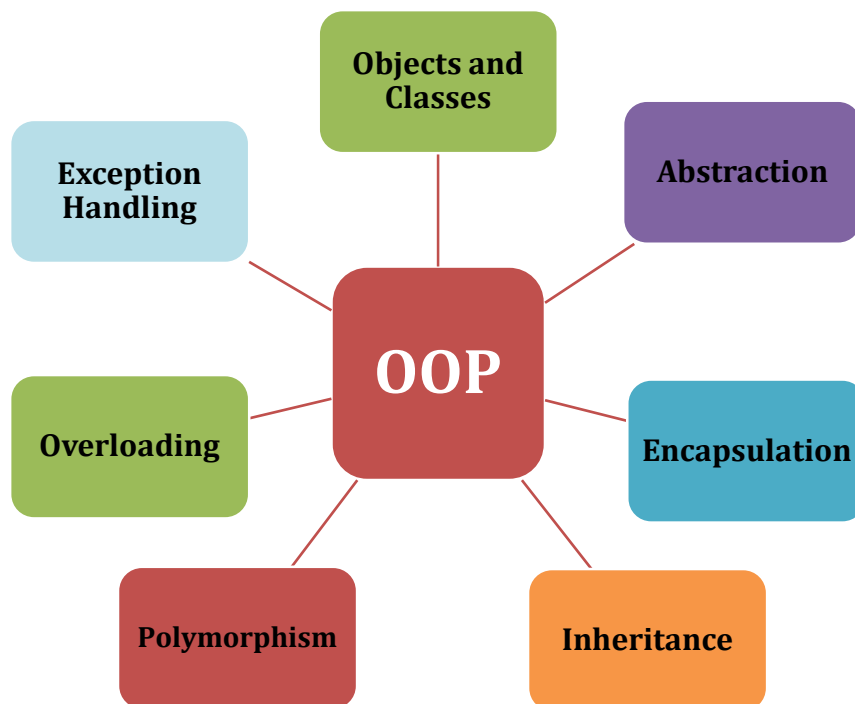


Figure 1.1.4: Features of OOP

1. Objects

Objects are the basic unit of OOP. They are instances of a class that have data members and use various member functions to perform tasks.

It is similar to structures in C language. Class can also be defined as user defined data type but it also contains functions in it. So, class is basically a blueprint for object. It declares and defines what data variables the object will have and what operations can be performed on the class's object.

Consider the following example:

```
'Animal' is a class
'Cow', 'Goat', 'Lion', 'Sheep' etc. are objects.
'ColdDrink' is a class
'Pepsi', 'Limca', 'Maaza', 'CocaCola' etc. are objects

'Fruit' is a class
'Apple', 'Banana', 'Mango' etc. are objects.
'Human' is a class
'Hand', 'Leg', 'Ear', 'Eye' etc. are objects.
```

2. Abstraction

Abstraction refers to showing only the essential features of the application and hiding the details. In C++, classes provide methods to the outside world to access and use the data variables, but the variables are hidden from direct access.

3. Encapsulation

It can also be said data binding. Encapsulation is all about binding the data variables and functions together in class.

4. Inheritance

Inheritance is a way to reuse once written code again and again. The class which is inherited is called base class and the class which inherits is called derived class. So when, a derived class inherits a base class, the derived class can use all the functions which are defined in base class, hence making code reusable.

5. Polymorphism

Polymorphism makes the code more readable. It is a feature, which lets it create functions with same name but different arguments, which will perform differently. That is function with same name, but functioning differently.

6. Overloading

Overloading is a part of polymorphism. When a function or operator is created and defined many times to perform different functions, they are said to be overloaded.

7. Exception Handling

Exception handling is a feature of OOP, to handle unresolved exceptions or errors produced at runtime.

(ii) Characteristics of OOP

- Emphasis on data rather than procedure.
- Programs are divided into entities known as objects.
- Data Structures are designed in such a way that they characterize objects.
- Functions that operate on data of an object are tied together in data structures.
- Data is hidden and cannot be accessed by external functions.
- Objects communicate with each other through functions.
- New data and functions can be easily added whenever necessary.
- Follows bottom up design in program design.

(iii) Merits and Demerits of OOP

OOP offers many benefits to both the developers and the users. The primary advantages and disadvantages are:

Table 1.1.1: Advantages and Disadvantages of OOP

Advantages	Disadvantages
We can eliminate redundant code and extend the use of existing classes through inheritance	It requires more data protection.
We can build programs from the standard working modules that communicate with one another, rather than having to start writing the code from scratch. This leads to saving of development time and higher productivity.	Inadequate for concurrent problems

The principle of data hiding helps the programmer to build secure programs that cannot be invaded by code in other parts of the program.	Inability to work with existing systems.
It is possible to have multiple objects to coexist without any interference.	Compile time and run time overhead.
It is possible to map objects in the problem domain to those objects in the program.	Unfamiliarity causes training overheads
It is easy to partition the work in a project based on objects.	
The data-centered design approach enables us to capture more details of a model in an implementable form.	
Object-oriented systems can be easily upgraded from small to large systems.	
Message passing techniques for communication between objects make the interface descriptions with external systems much simpler.	
Software complexity can be easily managed.	
Object-orientation contributes to the solution of many problems associated with the development and quality of software products. The new technology promises greater programmer productivity, better quality of software and lesser maintenance cost.	

(iv) Procedure Oriented Programming vs. Object Oriented Programming

Table 1.1.2: POP vs. OOP

Procedure Oriented Programming	Object Oriented Programming
Main program is divided into small parts depending on the functions	Main program is divided into small object depending on the problem
There is no perfect way for data hiding	Data hiding prevents illegal access of function from outside of it
Top down process is followed for program design	Bottom up process is followed for program design
Importance is given to the sequence of things to be done	Importance is given to the data
Functions share global data <i>i.e.</i> , data move freely around the system from function to function	The data is private
No access specifier	There are public, private, protected specifiers
Operator cannot be overloaded	Operator can be overloaded
<i>For example</i> , C, Pascal, FORTRAN	<i>For example</i> , C++, Java



Self-assessment Questions

- 10) A class is a user defined data type
a) TRUE
b) FALSE
- 11) An object contains_____ and _____.
- 12) Polymorphism is related to_____.
a) Objects
b) Classes
c) Methods
d) Fields

1.1.5 Introduction to C++

C++ was developed in 1979 by Bjarne Stroustrup at Bell Laboratories in New Jersey. C++ is a further advancement of "C" language. At first it was called "C with Class". The name was changed to C++ in 1983. C++ has been developing to address the issues of its clients. This development has been guided by the experience of clients of varying backgrounds working in an awesome scope of application areas.

(i) C++ Data Types

Data types define the way you use storage in the programs you write. They are used to define types of variables and contents used. Data types can be built in or abstract.

Built in Data Types

These are the data types which are predefined and are wired directly into the compiler. *For example*, int, char etc.

User defined or Abstract data types

These are the type, that user creates as a class. In C++ these are classes whereas in C it was implemented by structures.

The built in data types are listed below:

Table 1.1.3: Built in Data Types

Data Type	Description	Example
Char	for character storage (1 byte)	<code>char a = 'A';</code>
Int	for integral number (2 bytes)	<code>int a = 1;</code>
Float	single precision floating point (4 bytes)	<code>float a = 3.14159;</code>
Double	double precision floating point numbers (8 bytes)	<code>double a = 6e-4; // e is for exponential)</code>

Bool	Boolean (True or False)	<pre> Bool bln=253; //bln will be set to TRUE </pre>
void	Without any Value	<pre> int a = 10 float b = 5 void *a = (void *)&a a = (void *)&b//Generic pointer which can point to any address </pre>
wchar_t	Wide Character	<pre> wchar_t w; w = L'A'; </pre>

Other data types

Enum as Data type

Enumerated type declares a new type-name and a sequence of value containing identifiers which has values starting from 0 and incrementing by 1 every time.

For example,

```
enum seasons { spring, summer, autumn, winter };
```

In the above example, an enum type seasons is declared with 4 enumerators (spring, summer, autumn, winter). By default, the value of first enumerator is 0, second is 1 and so on. Hence, spring is equal to 0, summer is equal to 1 and autumn is 2 and winter is 3.

Modifiers

Specifiers modify the meanings of the predefined built-in data types and expand them to a much larger set. There are four data type modifiers in C++:

1. long
2. short
3. signed
4. unsigned

Syntax for the above modifiers:

modifier-type data-type var-name;

Example: short int i;

Program:

```
#include<iostream>
using namespace std;
int main()
{
    short int i; // a signed short integer
    short unsigned int j; // an unsigned short integer
    j = 50000;
    i = j;
    cout << i << " " << j;
    return 0;
}
```

Output:

-1553650000

Important Points related to modifiers:

- long and short modify the maximum and minimum values that a data type will hold.
- A plain int must have a minimum size of short.
- **Size hierarchy:** short int < int < long int
- Size hierarchy for floating point numbers is : float < double < long double
- **long float** is not a legal type and there are no **short floating point** numbers.
- **Signed** types includes both positive and negative numbers and is the default type.
- **Unsigned**, numbers are always without any sign, that is always positive.

(ii) Operators and Keywords

Operators are special type of functions that takes one or more arguments and produces a new value. *For example*, addition (+), subtraction (-), multiplication (*) etc., are all operators. Operators are used to perform various operations on variables and constants.

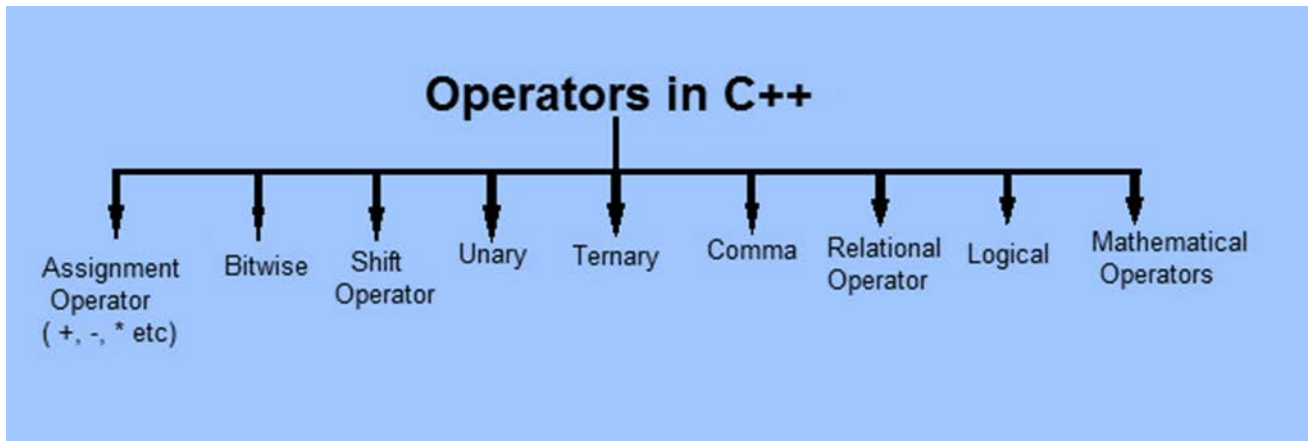


Figure 1.1.4: Operators in C++

Types of operators

1. Assignment Operator
2. Mathematical Operators
3. Relational Operators
4. Logical Operators
5. Bitwise Operators
6. Shift Operators
7. Unary Operators
8. Ternary Operator
9. Comma Operator
10. typedef Operator

1. Assignment Operator (=)

Operator '=' is used for assignment, it takes the right-hand side (called rvalue) and copy it into the left-hand side (called lvalue). Assignment operator is the only operator which can be overloaded but cannot be inherited.

For example,

```
int number=100;
```

2. Mathematical Operators

There are operators used to perform basic mathematical operations. Addition (+), subtraction (-), division (/), multiplication (*) and modulus (%) are the basic mathematical operators. Modulus operator finds the remainder of an integer division and cannot be used with floating-point numbers.

C++ and C also use a shorthand notation to perform an operation and assignment at same time.

For example,

```
int x=10;
x += 4 // will add 4 to 10, and hence assign 14 to X.
x -= 5 // will subtract 5 from 10 and assign 5 to x.
```

3. Relational Operators

These operators establish a relationship between operands. The relational operators are: less than (<), greater than (>), less than or equal to (<=), greater than or equal to (>=), equivalent (==) and not equivalent (!=).

You must notice that assignment operator is (=) and there is a relational operator, for equivalent (==). These two are different from each other, the assignment operator assigns the value to any variable, whereas equivalent operator is used to compare values, like in if-else conditions.

For example,

```
int x = 10; //assignment operator
x=5;       // again assignment operator
if(x == 5) // here we have used equivalent relational operator, for
comparison
{
    cout <<"Successfully compared";
}
```

4. Logical Operators

The logical operators are AND (&&) and OR (||). They are used to combine two different expressions together.

If two statement are connected using AND operator, the validity of both statements will be considered, but if they are connected using OR operator, then either one of them must be valid. These operators are mostly used in loops (especially while loop) and in Decision making.

Syntax:

```
!(6 <= 4)    // evaluates to true because (6 <= 4) would be false
!true       // evaluates to false
( (5 == 5) && (3 > 6) ) // evaluates to false ( true && false )
( (5 == 5) || (3 > 6) ) // evaluates to true ( true || false )
```

5. Bitwise Operators

These are used to change individual bits into a number. They work with only integral data types like char, int and long and not with floating point values.

- Bitwise AND operators &
- Bitwise OR operator |
- And bitwise XOR operator ^
- And, bitwise NOT operator ~

They can be used as shorthand notation too, &= , |= , ^= , ~= etc.

6. Shift Operators

Shift Operators are used to shift Bits of any variable. It is of three types,

1. Left Shift Operator <<
2. Right Shift Operator >>
3. Unsigned Right Shift Operator>>>

Syntax:

Left shift

```
[variable]<<[number of places]
```

Example:

```
Unsigned int x=3;           //3 in binary is 0011
X = x << 3;    // x will be 8
```

Right shift

```
[variable]>>[number of places]
```

Example:

```
Unsigned int x=12;          // 12 in binary is 1100
X = x >> 3;    // x will be 1
```

7. Unary Operators

These are the operators which work on only one operand. There are many unary operators, but increment ++ and decrement -- operators are most used.

Other Unary Operators: address of &, dereference *, new and delete, bitwise not ~, logical not !, unary minus - and unary plus +.

8. Ternary Operator

The ternary if-else ? : is an operator which has three operands.

```
int a = 10;
a > 5 ? cout << "true" : cout << "false"
```

9. Comma Operator

This is used to separate variable names and to separate expressions. In case of expressions, the value of last expression is produced and used.

For example,

```
int a,b,c; // variables declaration using comma operator
a=b++, c++; // a = c++ will be done.
Example:
j=10;
i = (j++; j+100; 999+j)
cout << i;
```

Output:

```
i will be equal to 1010.
j starts with the value 10. j is then incremented to 11. Next, j is
added to 100. Finally, j (still containing 11) is added to 999 which
yields the result 1010.
```

Sizeof operator in C++

Sizeof is a unary operator not a function, it is used to get information about the amount of memory allocated for data types and Objects. It can be used to get size of user defined data types too.

sizeof operator can be used with and without parentheses. If you apply it to a variable you can use it without parentheses.

```
cout << sizeof(double); //Will print size of double
int x = 2;
int i = sizeof x;
```

10. typedef Operator

typedef is a keyword used in C language to assign alternative names to existing types. It is mostly used with user defined data types, when names of data types get slightly complicated. Following is the general syntax for using typedef,

```
typedef existing_name alias_name
```

Lets take an example and see how typedef actually works.

```
typedef unsigned long ulong;
```

The above statement define a term **ulong** for an unsigned long type. Now this **ulong** identifier can be used to define unsigned long type variables.

```
ulong i, j ;
```

typedef and Pointers

typedef can be used to give an alias name to pointers also. Here we have a case in which use of typedef is beneficial during pointer declaration.

In Pointers * binds to the right and not the left.

```
int* x, y ;
```

By this declaration statement, we are actually declaring **x** as a pointer of type int, whereas y will be declared as a plain integer.

```
typedef int* IntPtr ;  
IntPtr x, y, z;
```

But if we use **typedef** like in above example, we can declare any number of pointers in a single statement.

For example,

```
typedef long miles_t;  
typedef long speed_t;  
  
miles_t distance = 5;  
speed_t mhz = 3200;  
  
// The following is valid, because distance and mhz are both actually  
type long  
distance = mhz;
```

(iii) Expressions

"Expression in C++ is a form when we combine operands (variables and constant) and C++ operators. "Expression can also be defined as:

"Expression in C++ is a combination of Operands and Operators. "Operands in C++ program are those values on which we want to perform operation.

There are three types of expressions:

1. Arithmetic expression
2. Relational expression
3. Logical expression

1. Arithmetic expression

An expression in which arithmetic operators are used is called arithmetic expression.

For example,

```
a = a * (c1 + c2);
```

2. Relational expression

An expressions in which relational operators are used is called relational expression.

For example,

```
if ( a <=b)
```

3. Logical expression

An expressions in which logical operators are used is called logical expression.

For example,

```
If ( !(a && b) )
```

(iv) Input and Output

C++ I/O occurs in streams, which are sequences of bytes. If bytes flow from a device like a keyboard, a disk drive, or a network connection etc. to main memory, this is called **input operation** and if bytes flow from main memory to a device like a display screen, a printer, a disk drive, or a network connection, etc. This is called **output operation**.

I/O Library Header Files:

There are following header files important to C++ programs:

Table 1.1.4: I/O Library Header files

Header File	Function and Description
<iostream>	This file defines the cin , cout , cerr and clog objects, which correspond to the standard input stream, the standard output stream, the un-buffered standard error stream and the buffered standard error stream, respectively.
<iomanip>	This file declares services useful for performing formatted I/O with so-called parameterized stream manipulators, such as setw and setprecision .
<fstream>	This file declares services for user-controlled file processing. We will discuss about it in detail in File and Stream related chapter.

The standard output stream (cout):

The predefined object **cout** is an instance of **ostream** class. The **cout** object is said to be "connected to" the standard output device, which usually is the display screen. The **cout** is used in conjunction with the stream insertion operator, which is written as << which are two less than signs

Syntax:

```
cout<<var; //this prints the value of variable which is stored in var.  
cout<<" "; //this prints the text enclosed within " "
```

For example,

```
cout<< "This is my first C++ program.";
```

The above **cout** statement will print This is my first C++ program.

The insertion operator << may be used more than once in a single statement as shown above and **endl** is used to add a new-line at the end of the line.

The standard input stream (cin):

The predefined object **cin** is an instance of **istream** class. The cin object is said to be attached to the standard input device, which usually is the keyboard. The cin is used in conjunction with the stream extraction operator, which is written as >> which are two greater than signs as shown in the following example.

```
#include<iostream>
using namespace std;
int main()
{
    char name[50];
    cout<<"Please enter your name: ";
    cin>> name;
    cout<<"Your name is: "<< name <<endl;
}
```

When the above code is compiled and executed, it will prompt you to enter a name. You enter a value and then hit enter to see the result something as follows:

Please enter your name: cplusplus

Your name is: cplusplus

The C++ compiler also determines the data type of the entered value and selects the appropriate stream extraction operator to extract the value and store it in the given variables.

The stream extraction operator >> may be used more than once in a single statement. To request more than one datum you can use the following:

```
cin>> name >> age;
```

This will be equivalent to the following two statements:

```
cin>> name;
cin>> age;
```

The standard error stream (cerr):

The predefined object cerr is an instance of ostream class. The cerr object is said to be attached to the standard error device, which is also a display screen but the object cerr is un-buffered and each stream insertion to cerr causes its output to appear immediately.

The cerr is also used in conjunction with the stream insertion operator as shown in the following example.

```
#include<iostream>
using namespace std;
int main()
{
    char str[]="Unable to read....";
    cerr<<"Error message : "<<str<<endl;
}
```

When the above code is compiled and executed, it produces the following result:

Error message: Unable to read....



Self-assessment Questions

13) C++ was invented in

- | | |
|---------|---------|
| a) 1979 | b) 1987 |
| c) 1999 | d) 1977 |

14) A unary operator used to calculate the sizes of datatypes.

- | | |
|------------|------------|
| a) Typedef | b) sizeof |
| c) Comma | d) ternary |

15) An operator used for assignment.

- | | |
|------|-------|
| a) = | b) == |
| c) . | d) : |

16) An operator displays data on the monitor.

- | | |
|---------|-----------|
| a) cout | b) cin |
| c) cerr | d) cprint |

1.1.6 Decision and Loop

Decision making is about deciding the order of execution of statements based on certain conditions or repeat a group of statements until certain specified conditions are met.

(i) Conditional Statements in C++

C++ handles decision-making by supporting the following statements:

- if statement
- switch statement
- conditional operator statement
- go to statement

Decision making with if statement

If statement may be implemented in different forms depending on the complexity of conditions to be tested. The different forms are:

- Simple if statement
- If....else statement
- Nested if....else statement
- else if statement

Simple if statement

The general form of a simple if statement is,

```
if( expression )
{
    statement-inside;
}
statement-outside;
```

If the expression is true, then 'statement-inside' it will be executed, otherwise 'statement-inside' is skipped and only 'statement-outside' is executed.

For example,

```
#include<iostream.h>
int main( )
{
```

```
int x,y;
x=15;
y=13;
if (x > y )
{
    cout<< "x is greater than y";
}
}
```

Output: x is greater than y

If...else statement

The general form of a simple if...else statement is,

```
if( expression )
{
    statement-block1;
}
else
{
    statement-block2;
}
```

If the 'expression' is true, the 'statement-block1' is executed, else 'statement-block1' is skipped and 'statement-block2' is executed.

For example,

```
void main( )
{
    int x,y;
    x=15;
    y=18;
    if (x > y )
    {
        cout<< "x is greater than y";
    }
    else
    {
        cout<< "y is greater than x";
    }
}
```

Output: y is greater than x

Nested if....else statement

The general form of a nested if...else statement is,

```
if( expression )
{
    if( expression1 )
    {
        statement-block1;
    }
    else
    {
        statement-block 2;
    }
}
else
{
    statement-block 3;
}
```

if 'expression' is false the 'statement-block3' will be executed, otherwise it continues to perform the test for 'expression 1'. If the 'expression 1' is true the 'statement-block1' is executed otherwise 'statement-block2' is executed.

For example,

```
void main( )
{
    int a,b,c;
    clrscr();
    cout<< "enter 3 number";
    cin>> a >> b >> c;
    if(a > b)
    {
        if( a > c)
        {
            cout<< "a is greatest";
        }
        else
        {
            cout<< "c is greatest";
        }
    }
    else
    {
        if( b> c)
        {
            cout<< "b is greatest";
        }
    }
}
```

```
        else
        {
            printf("c is greatest");
        }
    }
    getch();
}
```

else-if ladder

The general form of else-if ladder is,

```
if(expression 1)
{
    statement-block1;
}
else if(expression 2)
{
    statement-block2;
}
else if(expression 3 )
{
    statement-block3;
}
else
default-statement;
```

The expression is tested from the top (of the ladder) downwards. As soon as the true condition is found, the statement associated with it is executed.

For example,

```
void main( )
{
    int a;
    cout<< "enter a number";
    cin>> a;
    if( a%5==0 && a%8==0)
    {
        cout<< "divisible by both 5 and 8";
    }
    else if( a%8==0 )
    {
        cout<< "divisible by 8";
    }
    else if(a%5==0)
    {
        cout<< "divisible by 5";
    }
    else
```

```
{  
    cout<< "divisible by none";  
}  
getch();  
}
```

(ii) Looping Statements

In any programming language, loops are used to execute a set of statements repeatedly until a particular condition is satisfied.

How it works

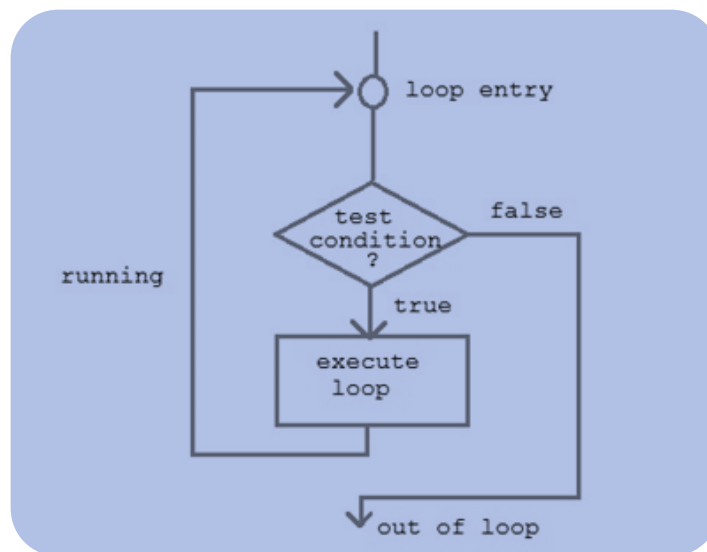


Figure 1.1.5: Looping Statements Working []

The above flowchart in the Figure 1.1.5 explains how looping statements work in a C++ program.

A sequence of statement is executed until a specified condition is true. This sequence of statement to be executed is kept inside the curly braces { } in the program known as loop body. After every execution of loop body, condition is checked, and if it is found to be true the loop body is executed again. When condition check comes out to be **false**, the loop body will not be executed and it will come out of the loop.

There are 3 types of loops in C++ language:

- while loop
- for loop
- do-while loop

while loop

while loop can be addressed as an **entry control** loop. It is completed in 3 steps.

- Variable initialization. (*For example*, int x=0;)
- condition (*For example*, while(x<=10))
- Variable increment or decrement (x++ or x-- or x=x+2)

Syntax:

```
variable initialization ;
while (condition)
{
    statements ;
    variable increment or decrement ;
}
```

Program:

```
1  #include <iostream>
2  using namespace std;
3
4  int main ()
5  {
6      // Local variable declaration:
7      int a = 10;
8
9      // while loop execution
10     while( a < 20 )
11     {
12         cout << "value of a: " << a << endl;
13         a++;
14     }
15
16     return 0;
17 }
```

Output:

```
1  value of a: 10
2  value of a: 11
3  value of a: 12
4  value of a: 13
5  value of a: 14
6  value of a: 15
7  value of a: 16
8  value of a: 17
9  value of a: 18
10 value of a: 19
```

for loop

for loop is used to execute a set of statement repeatedly until a particular condition is satisfied. We can say it an open ended loop. General format is,

```
for(initialization; condition ; increment/decrement)
{
    statement-block;
}
```

In for loop we have exactly two semicolons, one after initialization and second after condition. In this loop we can have more than one initialization or increment/decrement, separated using comma operator. **for** loop can have only one **condition**.

Program:

```
1  #include <iostream>
2  using namespace std;
3
4  int main ()
5  {
6      // for loop execution
7      for( int a = 10; a < 20; a = a + 1 )
8      {
9          cout << "value of a: " << a << endl;
10     }
11
12     return 0;
13 }
```

Output:

```
1  value of a: 10
2  value of a: 11
3  value of a: 12
4  value of a: 13
5  value of a: 14
6  value of a: 15
7  value of a: 16
8  value of a: 17
9  value of a: 18
10 value of a: 19
```

Nested for loop

We can also have nested **for** loop, *i.e.*, one **for** loop inside another **for** loop. Basic syntax is,

```
for(initialization; condition; increment/decrement)
{
    for(initialization; condition; increment/decrement)
    {
        statement ;
    }
}
```

Program:

```
1  #include <iostream>
2  using namespace std;
3
4  int main ()
5  {
6      int i, j;
7
8      for(i=2; i<50; i++) {
9          for(j=2; j <= (i/j); j++)
10             if(!(i%j)) break; // if factor found, not prime
11             if(j > (i/j)) cout << i << " is prime\n";
12         }
13     return 0;
14 }
```

Output:

```
1  2 is prime
2  3 is prime
3  5 is prime
4  7 is prime
5  11 is prime
6  13 is prime
7  17 is prime
8  19 is prime
9  23 is prime
10 29 is prime
11 31 is prime
12 37 is prime
13 41 is prime
14 43 is prime
15 47 is prime
```

do while loop

In some situations it is necessary to execute body of the loop before testing the condition. Such situations can be handled with the help of **do-while** loop. **do** statement evaluates the body of the loop first and at the end, the condition is checked using **while** statement. General format of **do-while** loop is,

```
do
{
    ....
    .....
}
while(condition)
```

Program:

```
1  #include <iostream>
2  using namespace std;
3
4  int main ()
5  {
6      // Local variable declaration:
7      int a = 10;
8
9      // do loop execution
10     do
11     {
12         cout << "value of a: " << a << endl;
13         a = a + 1;
14     }while( a < 20 );
15
16     return 0;
17 }
```

Output:

```
1  value of a: 10
2  value of a: 11
3  value of a: 12
4  value of a: 13
5  value of a: 14
6  value of a: 15
7  value of a: 16
8  value of a: 17
9  value of a: 18
10 value of a: 19
```

Jumping out of loop

Sometimes, while executing a loop, it becomes necessary to skip a part of the loop or to leave the loop as soon as certain condition becomes true, that is jump out of loop. C language allows jumping from one statement to another within a loop as well as jumping out of the loop.

1. Break statement

When break statement is encountered inside a loop, the loop is immediately exited and the program continues with the statement immediately following the loop.

2. Continue statement

It causes the control to go directly to the test-condition and then continue the loop process. On encountering continue, cursor leaves the current cycle of loop, and starts with the next cycle.

3. Go to statement

You insert a label in your code at the desired destination for the go to. The label is always terminated by a colon. The keyword go to, followed by this label name, and then takes you to the label. The following code fragment demonstrates this approach.

```
gotoSystemCrash;
```

Program:

```
1  #include <iostream>
2  using namespace std;
3
4  int main ()
5  {
6      // Local variable declaration:
7      int a = 10;
8
9      // do Loop execution
10     do
11     {
12         cout << "value of a: " << a << endl;
13         a = a + 1;
14         if( a > 15)
15         {
16             // terminate the loop
17             break;
18         }
19     }while( a < 20 );
20
21     return 0;
22 }
```

Output:

```
1  value of a: 10
2  value of a: 11
3  value of a: 12
4  value of a: 13
5  value of a: 14
6  value of a: 15
```

Program:

```
1  #include <iostream>
2  using namespace std;
3
4  int main ()
5  {
6      // Local variable declaration:
7      int a = 10;
8
9      // do Loop execution
10     do
11     {
12         if( a == 15)
13         {
14             // skip the iteration.
15             a = a + 1;
16             continue;
17         }
18         cout << "value of a: " << a << endl;
19         a = a + 1;
20     }while( a < 20 );
21
22     return 0;
23 }
```

Output:

```
1  value of a: 10
2  value of a: 11
3  value of a: 12
4  value of a: 13
5  value of a: 14
6  value of a: 16
7  value of a: 17
8  value of a: 18
9  value of a: 19
```

Program:

```
1  #include <iostream>
2  using namespace std;
3
4  int main ()
5  {
6      // Local variable declaration:
7      int a = 10;
8
9      // do Loop execution
10     LOOP:do
11     {
12         if( a == 15)
13         {
14             // skip the iteration.
15             a = a + 1;
16             goto LOOP;
17         }
18         cout << "value of a: " << a << endl;
19         a = a + 1;
20     }while( a < 20 );
21
22     return 0;
23 }
```

Output:

```
1  value of a: 10
2  value of a: 11
3  value of a: 12
4  value of a: 13
5  value of a: 14
6  value of a: 16
7  value of a: 17
8  value of a: 18
9  value of a: 19
```



a) TRUE b) FALSE

a) 0 b) 1
c) N d) N+1

20) The do-while loop checks the test expression and then executes the statements placed within its body.

a) TRUE b) FALSE

a) Break b) Continue
c) Default d) case

a) TRUE b) FALSE



Summary

- C++ is an Object-Oriented programming language which focuses on classes and objects.
- Procedure-Oriented programming language focuses on dividing the main program into smaller parts called functions. The flow of program is sequential in nature.
- Procedure Oriented programming languages uses functions that share global data to execute the main program.
- Procedure Oriented programming languages follows a top down approach of problem solving and data hiding is not possible in procedure oriented programming languages.
- Object-Oriented programming language focuses on dividing the main program into small objects.
- The use of objects makes data hiding possible in Object-Oriented programming language.
- Object-Oriented programming languages follow a bottom up approach.
- C++ data types are used to define type of variables and the content used in it.
- C++ can handle decision-making with the help of conditional statements such as if, if-else and switch statements.
- C++ is also able to handle looping statements such as for, while and do-while which makes the program to execute a set of statements repeatedly until a particular condition is satisfied.



Terminal Questions

1. Explain various data types in C++ with an example for each.
2. How is Object Oriented Programming superior to Procedural Programming? Explain with evidence.
3. Write a C++ program to show the use of conditional statements in solving a problem.
4. Distinguish between break and continue statements
5. Describe the similarities and differences between a while loop and a do-while loop.
6. Write a C++ program using do-while loop to generate Fibonacci series up to N.



Answer Keys

Self-assessment Questions	
Question No.	Answer
1	Syntax and semantics
2	b
3	Keywords and a special syntax
4	b
5	Middle level
6	c
7	a
8	Global
9	a
10	TRUE
11	Data and methods
12	b
13	a
14	b
15	a
16	a
17	TRUE
18	c
19	goto
20	FALSE
21	Break
22	TRUE



Activities

Activity Type: Online

Duration: 30 Minutes

Description:

1. Develop and execute a C++ program to find the largest of 2 or 3 numbers using if-else statement and nested if-else statement.
2. Develop and execute a program to find the factorial of a number using while and do-while loop.

Bibliography



e-References

- Developer.com, (2016). *The Evolution of Object-Oriented Languages*. Retrieved 7 April, 2016 from <http://www.developer.com/java/other/article.php/3493761/The-Evolution-of-Object-Oriented-Languages.htm>
- ntu.edu.sg, (2016). *C++ Programming Language Object-Oriented Programming (OOP) in C++*. Retrieved 7 April, 2016 from https://www3.ntu.edu.sg/home/ehchua/programming/cpp/cp3_OOP.html
- tutorialspoint.com, (2016). *C++ Data Types*. Retrieved 7 April, 2016 from http://www.tutorialspoint.com/cplusplus/cpp_data_types.htm



External Resources

- Balaguruswamy, E. (2008). *Object Oriented Programming with C++*. Tata McGraw Hill Publications.
- Lippman. (2006). *C++ Primer* (3rd ed.). Pearson Education.
- Robert, L. (2006). *Object Oriented Programming in C++* (3rd ed.). Galgotia Publications References.
- Schildt, H., & Kanetkar, Y. (2010). *C++ completer* (1st ed.). Tata McGraw Hill.
- Strousstrup. (2005). *The C++ Programming Language* (3rd ed.). Pearson Publications.



Video Links

Topic	Link
Looping statements in C++	https://www.youtube.com/watch?v=8Cy7shy-Jjo&nohtml5=False
Operators and expressions in C++	https://www.youtube.com/watch?v=hoqnm5kV_Fo&nohtml5=False
C++ programming	https://www.youtube.com/watch?v=KNFv4DZ4Mp4
Procedure oriented Vs. object oriented programming	https://www.youtube.com/watch?v=NJsbX58hPLY
Basic concepts of OOPs in C++	https://www.youtube.com/watch?v=q5EIJHe0mjw&nohtml5=False



Notes:

