## MCA-108 Object Oriented Programming using C++ L-T-P: 3-1-0

**Prerequisite:** Basic knowledge of object oriented programming.

**UNIT I: (10 Hours)**
Introduction to object oriented programming, user defined types, polymorphism, and encapsulation. Getting started with C++ - syntax, data-type, variables, strings, functions, exceptions and statements, namespaces and exceptions operators. Flow control, functions, recursion. Arrays and pointers, structures.

**UNIT II: (10 Hours)**
Abstraction Mechanisms: Classes, private, public, constructors, destructors, member functions, static members, references etc. class hierarchy, derived classes.
Inheritance: simple inheritance, polymorphism, object slicing, base initialization, virtual functions.

**UNIT III: (12 Hours)**
Prototypes, linkages, operator overloading, ambiguity, friends, member operators, operator function, I/o operators etc. Memory management: new, delete, object copying, copy constructors, assignment operator, this Input/Output.
Exception handling: Exceptions and derived classes function exception declarations, Unexpected exceptions.
Exceptions when handling exceptions, resource capture and release etc.

**UNIT – IV: (8 Hours)**
Templates and Standard Template library: template classes declaration, template functions, namespaces, string, iterators, hashes, iostreams and other type.
Design using C++ design and development, design and programming, role of classes.

**Text Book:**
1. E. Balaguruswamy, Object-Oriented Programming in C++. 4 ed, Tata McGraw-Hill.
2. Ashok N. Kamthane, Object-Oriented Programming with ANSI & Turbo C++, Pearson Education.

**Reference Books:**
1. Herbert Schildt, The Complete Reference C++. 4 ed, Tata McGraw-Hill.
2. Bjarne Stroustrup, Programming: Principles and Practice Using C++. 4 ed, AddisonWesley.

---

**Course Outcomes:**
1. Familiar to map real world problems into the Programming language using objects.
2. Can solve the problems in systematic way using class and method paradigms.
3. Efficiently implement linear, nonlinear data structures and various searching and sorting techniques.
4. Efficiently implement Exception handling techniques.

# CONTENTS


1. INTRODUCTION
   - 1.1 Features
   - 1.2 Advantages
   - 1.3 Uses of OOP
2. INPUT AND OUTPUT IN C++
   - 2.1 pre-defined streams
   - 2.2 stream classes
   - 2.3 Typecasting
3. C++ DECLARATION
   - 3.1 Parts of C++ program
   - 3.2 Token
   - 3.3 Data types in C++
     - 3.3.1 Basic data types
     - 3.3.2 Derived data type
     - 3.3.3 User defined data type
     - 3.3.4 Void data type
   - 3.4 Operators in C++
4. FUNCTIONS IN C++
   - 4.1 Passing Arguments
     a) Call by value(pass by value)
     b) Call by address(pass by address)
     c) Call by reference(pass by reference)
   - 4.2 Default Arguments
   - 4.3 Inline functions
   - 4.4 Function overloading
   - 4.5 Library function
   - 4.6 Const Argument
5. CLASSES & OBJECTS
   - 5.1 Introduction
   - 5.2 Declaring objects
   - 5.3 Defining member function
   - 5.4 Declaring member function outside the class
   - 5.5 Static member variable
   - 5.6 Static member function
   - 5.7 Static object
   - 5.8 Object as function arguments
     - 5.8.1 pass-by value
     - 5.8.2 pass-by reference
     - 5.8.3 pass by address
   - 5.9 Friend function
   - 5.10 Friend classes
   - 5.11 Constant member function

5.12 Recursive member function
5.13 Member function & non-member function
5.14 Overloading member function
5.15 Overloading main() function
5.16 Indirect recursion

6. CONSTRUCTOR AND DESTRUCTOR
6.1 Constructor with Arguments
6.2 Constructor overloading
6.3Constructor with default Argument
6.5 Copy Constructor
6.6Const Object
6.7 Destructor
6.8 Qualifier & Nested classes
6.9 Anonymous object
6.10 Private Constructor and Destructor
6.11 main() as constructor & destructor
6.12 Local vs Global Object

7. INHERITANCE
7.1 Types of Inheritance
7.1.1 Single Inheritance
7.1.2 Multiple Inheritance
7.1.3 Hierarchical Inheritance
7.1.4 Multilevel Inheritance
7.1.5 Hybrid Inheritance
7.1.6Multipath Inheritance
7.2 Container class
7.3 Abstract class
7.4 Common constructor
7.5Pointer & Inheritance
7.6 Overloading Member function

8. OPERATOR OVERLOADING
8.1 Introduction
8.2 Overloading Unary operator
8.3 Overloading binary operator using friend function
8.4 Overloading increment & decrement operator
8.5 Overloading special operator
8.6 Overloading function call operator
8.7 Overloading class member access operator
8.8 Overloading comma operator
8.9 Overloading stream operator
8.10 Overloading extraction & insertion operator

9. POINTERS AND ARRAY
9.1 Void pointer
9.2 Wild pointer

9.3 Class pointer
9.4 Pointer to derived & base classes
9.5 Binding , polymorphism
9.6 Virtual function
9.7 Pure virtual function
9.8 Object slicing
9.9 VTABLE & VPTR
9.10 New & delete operator
9.11 Dynamic object
9.12 heap
9.13 Virtual destructor
10.EXCEPTION HANDLING
10.1 Introduction
10.2 Exception handling mechanism
10.3 Multiple catch statements
10.4 Catching multiple exception
10.5 Rethrowing an exception
10.6 Specifying exception
10.7 Exceptions in constructor & destructors
10.8 Controlling uncaught exception
10.8.1 Terminate function
10.8.2 Set-terminate function
10.9 Exception & inheritance
11.TEMPLATES
11.1 Introduction
11.2 Need Of Template
11.3 Normal Function Template
11.4 Member Template Function
11.5 Working Of Function Templates:
11.6 Overloading Of Template Functions :
11.7 Exception Handling With Class Template:
11.8 Class Templates With Overloaded Operators :
11.9 Class Template And Inheritance:
11.10 Difference Between Templates And Macros:
11.11 Guidelines For Templates
12.NAMESPACES
12.1 namespace Scope:
12.2 namespace Declaration:
12.3 Accessing elements of a name space:
12.4 Nested namespace:
12.5 Anonymous namespaces:
12.6 Function In namespace:
12.7 Classes in namespace:
12.8 namespace Alias:

12.9 explicit Keyword :
12.10 mutable keyword:
12.11 Manipulating Strings:
12.12 Manipulating String Objects:
12.13 Relational Operator:
12.14 Accessing Characters In String:
13. STANDARD TEMPLATE LIBRARY(STL)
13.1 Component of STL
13.1.1 Containers
13.1.2 Algorithm
13.1.3 Iterator
13.2 Types of containers
13.2.1 Sequence Containers
13.2.2 Associative Container
13.2.3 Derived Container
13.3 Algorithm
13.4 Iterator

# INTRODUCTION

- C++ is an object oriented programming language & is an extension of c.
- Bjarne Stroustrup at AT & Bell Lab,USA developed it.
- He called the language C with classes.

## 1.1 Feature

1) Data Abstraction
   Abstraction directs to the procedure of representing essential features without including the background details.
2) Encapsulation
   The packing of data and functions in to a single component is known as encapsulation.
3) Inheritance
   It is the method by which objects of one class get the properties of objects of another class.
4) Polymorphism
   It allows the same function to act differently in different classes.
5) Dynamic Binding
   Binding means connecting one program to another program that is to be executed in reply to the call. It is also known as late binding.
6) Reusability
   Object oriented technology allows reusability of the classes by executing them to other classes using inheritance.
7) Delegation
   When an object of one class is used as a data member in another class such composition of object is known as delegation.
8) Genericity
   This feature allows declaration of variables without specifying exact data type. The compiler identifies the data type at run time.

## 1.2 Advantages

- OOPS can be comfortably upgraded.
- Using inheritance redundant program, codes can be eliminated & use of previously defined classes may be continued.
- OOP languages have standard class library.

## 1.3 Uses of OOP

- Object-Oriented DBMS
- Office automation software
- AI and expert systems
- CAD/CAM software
- Network programming
- System software

## INPUT & OUTPUT IN C++

## 2.1 pre-defined streams

| | |
|---|---|
| cin | • Standard input, usually keyboards, corresponding to stdin in C.<br>• It handles input from input devices. |
| Cout | • Standard output, usually screen, corresponding to stdout in C.<br>• It passes data to output devices such as monitor and printer. |
| clog | • It control error messages that are passed from buffer to the standard error device |
| cerr | • Standard error output, usually screen, corresponding to stderr in C.<br>• it controls the unbuffered output data.<br>• It catches the error & passes to standard error device monitor. |

### 13.1 stream classes

**Input stream**

- It does read operation through keyboard.
- It uses cin as object.
- The cin statement is used to read data through the input device & uses the extraction operator " >> ".

Syntax: cin>> varname;

```
int a;    float b;
cin>>a>>b;
```

**Output stream**

- It display contents of variables on the screen.
- It uses insertion operation " << " before variable name.

Syntax :

```
cout << variable;
 int a; float b;
 cout <<a << b;
```

## 2.3 Typecasting

- It refers to the conversion of data from one basic type to another by applying external use of data type keywords

Program:

Write a program to display from A-Z using typecasting.

```
# include<iostream.h>
#include<conio.h>
void main()
{
int i;
for(i=65;i<=91;i++)
cout<<(char)i;
}
```

**get()** – it is used to read a single character from the keyboard.

**Puts()** – it is used to display a single character on the screen.

```
 cin.get(ch);
 cin.put(ch);
```

**getline()** – it is used to read the string including whitespace. The object cin calls the function by
syntax cin.getline(variable,size);
cin.getline(name,30);
**read()** – it is used to read text through the keyboard.
Syntax cin.read(variable,size);
**write()** – it is used to display the string on screen.
Syntax cout.write(variable,size);

# C++ DECLARATION

## 3.1 Parts of C++ program

| |
|---|
| Include files |
| Class definition or declaration |
| Class function definition |
| main() function(compulsory) |

**Include files** It controls the header files for function declaration.Each header file has an extension.
e.g # include " iostream.h"

**Class declaration** A class is defined in this section.It contains data variable, data member, function prototype, function definition.

**Class function definition** This contains definition of function.

**main()** C++ programs always start execution from main().

## 3.2 Token

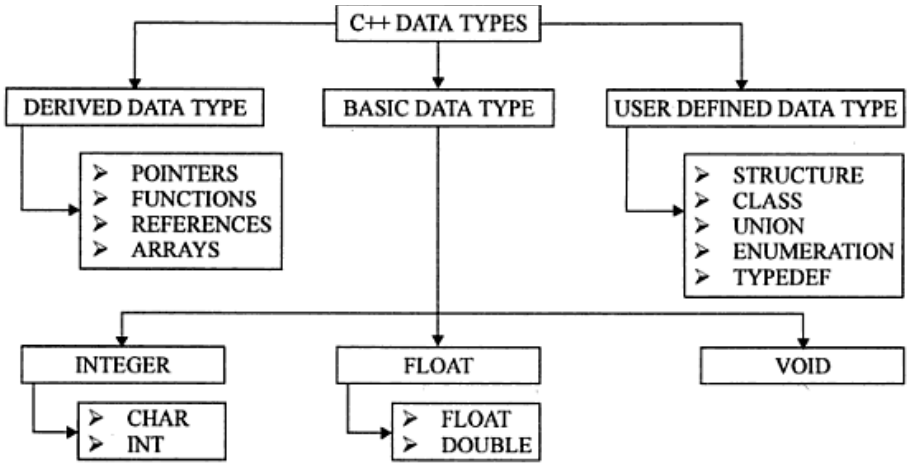
The smallest individual units in a program is known as token.

- Keywords
- Identifiers
- Constants
- Strings
- Operators

## 3.3 Data types in C++

- Basic data types
- Derived data types
- User defined data type
- Void data type

### 3.3.1 Basic data types




C++ data types

Basic data types supported by C++ with size and range

| DATA TYPE | SIZE IN BYTES | RANGE |
|---|---|---|
| char | 1 | -128 to 127 |
| unsigned char | 1 | 0 to 255 |
| signed char | 1 | -128 to 127 |
| int | 2 | -32768 to 32767 |
| unsigned int | 2 | 0 to 65535 |
| signed int | 2 | -32768 to 32767 |
| short int | 2 | -32768 to 32767 |
| unsigned short int | 2 | 0 to 65535 |
| signed short int | 2 | -32768 to 32767 |
| long int | 4 | -2147483648 to 2147483647 |
| signed long int | 4 | -2147483648 to 2147483647 |
| unsigned long int | 4 | 0 to 4294967295 |
| float | 4 | 3.4E-38 to 3.4E+38 |
| double | 8 | 1.7E –308 to 1.7E+308 |
| long double | 10 | 3.4E –4932 to 1.1E +4932 |
| enum | 2 | -32768 to 32767 |
| bool | 1 | true / false |

### 3.3.2 Derived data type

**a) Pointer**

- A pointer is a memory variable that stores a memory address.
- Pinter can have any name i.e legal for other variable and is declared in the same fashion like other variable but it is always denoted by '*' operator.

```
int *a;
float *f;
```

**b) Function**

- A function is a self-contained block or subprogram of one or more statement that perform a specific task when call

```
main( )
{
fun A( );
fun B( );
}
```

**c) Array**

- Array is a collection of elements of homogenous data type.

  **Program**

  Write a program to print array element using pointer.

```
# include<iostream.h>
#include<conio.h>
void main()
{
int a[5],*p;
int i;
clrscr();
cout<<"enter the array element ";
```

```
for(i=0;i<5;i++)
cin>>a[i];
p=a;
cout<<"elements are";
for(i=0;i<5;i++)
cout<<*(p+i);
getch();
}
```

**d) Reference**

- It is used to define a reference variable.
- This variable prepares an alias for a previously defined variable.

```
int qty=10;
int &qt=qty;
```

syntax : data type & reference variable name=variable name;

### 3.3.3 User defined data type

**a) Structure & classes**

```
struct student
{
char nm[30];
char sex[10];
int roll;
}
```

**Classes**

```
class student
{
 char nm[30];
 char sex[10];
 int roll;
 void input(void);
};
```

**b) Union**

```
union number
{
char c;
int I;
}
union number num;
```

**Anonymous union**

- it doesn't contain tag name.
- Elements of such union can be accessed without using tag name.

```
void main()
 {
union
{
```

```
int r;
float f;
};
F=3.1;r=2;
Cout<<r<<f;
}
```

**c) Enumerated data type**

Keyword: enum

enum logical {true,false}

### 3.3.4 Void data type

It also known as empty data type..

```
void display()
{
Cout<<"Hello";
}
```

## 3.4 Operators in C++

1) Insertion operator(<<)
2) Extraction operator(>>)
3) Scope access /resolution operator(::)
4) New(Memory allocation operator)
5) Delete(Memory release operator)
6) Comma operator(,)
7) Reference operator(&)
8) Dereferencing operator(*)

## FUNCTIONS IN C++

# 4.1 Passing Arguments

- The main objective of passing arguments to function is message passing.
- It is also known as communication between two function i.e coller & collee function.

## 4.1.1 Call by value(pass by value)

- Value of actual argument is passed to the formal argument & operation is done on the formal argument.
- Any change in formal argument doesn't effect the actual argument because formal argument are photocopy of actual arguments .

*Program - Write a program to swap two value*

```
#include< iostream.h>
void main()
{
int x,y;
void swap(int,int);
cout<<"enter value of x and y";
cin>>x>>y;
swap(x,y);
}
void swap(int a, int b)
{
int k;
k=a; a=b; b=k;
cout<<"a="<<a<<"b="<<b;
}
```

## 4.1.2 Call by address(pass by address)

- Instead of passing value address are passed. Function operates on addresses rather than values.
- The formal arguments are pointers to the actual argument.hence,changes made in the arguments are permanents.

*Program - Write a program to swap two number*

```
#include<iostream.h>
void main()
{
int x,y;
void swap(int * ,int *);
cout<<"enter value of x and y";
cin>>x>>y;
swap(&x, &y);
cout<<"x="<<x<<"y="<<y;
}
void swap(int *p, int *q)
```

```
{
int k;
k=*p; *p=*q; *q=k;
}
```

### 4.1.3 Call by reference(pass by reference)

*Program - Write a program to swap two value*

```
#include<iostream.h>
void main()
{
int x,y;
void swap(int & , int &);
cout<<"enter value of x and y";
cin>>x>>y;
swap(x, y);
cout<<"x="<<x<<"y="<<y;
}
void swap(int &p, int &q)
{
int k;
k=p;
 p=q;
q=k;
}
```

## 4.2 Default Arguments

- C++ compiler allows the programmer to assign default values in function prototype.
- When the function is called with less parameter or without parameter the default values are used for operation.

```
#include <iostream.h>
void main()
{
int sum(int a,int b=10;int c=15,int d=20);
int a=2;int b=3; int c=4; int d=5;
cout<<sum(a,b,c,d);
cout<<sum(a,b,c);
cout<<sum(a,b);
cout<<sum(a);
cout<<cum(b,c,d);
}
int sum(int j,int k, int l, int m)
{
 return(j+k+l+m);
}
```

## 4.3 Inline functions

- C++ provides a mechanism called inline function . When a function is declared as inline the compiler copies the code of the function in calling function i.e function body is inserted in place of function call during compilation.
- Passing of control between coller and collee function is avoided.

*Program - Write a program to find square of a number.*

```
#include <iostream.h>
inline float square(float j)
{     return(j*j);   }
void main()
{
Int p,q;
cout<<"enter a number:" ;
cin>>p;
q=square(p);
cout<<q;
}
```

## 4.4 Function overloading

- Defining multiple function with same name is known as function overloading or function polymorphism.
- The overloading function must be different in its argument list & with different data type

*Program - Write a program to addition of int & float number using function overloading.*

```
#include<iostream.h>
int add(int,int);
float add(float,float,float);
int  main()
{
Clrscr();
float fa,fb,fc,fsum;
int ia,ib,ic,isum;
cout<<"enter integer value:";
cin>>ia>>ib>>ic;
cout<<"enter float value";
cin>>fa>>fb>>fc;
isum=add(ia,ib,ic);
cout<<isum;
fsum=add(fa,fb,fc);
cout<<fsum;
return 0;
}
add(int j, int k, int l)
{
```

```
return(j+k+l);
}
float add(float a, float  b,float c)
{
return(a+b+c);
}
```

## 4.5 Library function

**a) ceil, ceill and floor, floor1**

- The function **ceil, ceill** round up given float number.
- The function **floor,floor1** round down  float number.
- They are defined in **math.h** header file

```
#include<iostream.h>
#include<math.h>
void main()
{
float num=3.2;
float d,u;
d=float(num);
u=ceil(num);
cout<<num<<u<<d;
}
```

**b) Modf and modf1**

- The function **modf** breaks double into integer and fraction elements
- The  function **modf1** breaks long double into integer and fraction elements.

```
void main()
{
double f,I;
double num=211.57;
f=modf(num,&i);
cout<<num<<i<<f;
}
```

**c) abs, fabs and labs**

- The function **abs()** returns the absolute value of integer.
- The **fabs()** returns the absolute value of a floating point number.
- The **labs()** returns the absolute value of a long number.

Declaration:

```
Int abs(int n);
double fabs(double n);
long int labs(long int n);
```

**d) norm**

- The function is defined in complex.h header file and it is used to calculate the square of the absolute value.

Program

Write a program to calculate the square of the complex number using norm() function.

```
#include<iostream.h>
#include<complex.h>
#include<conio.h>
int main(0
{
 double x=-12.5;
cout<<norm(x);
return 0;
}
```

**e) complex(), real(), imag() and conj()**

**complex():** - is defined in complex.h header file & it create complex numbers.

**real()** :- it returns real part of the complex number .

**imag()** ;- it returns imaginary part of the complex number.

**Conj()** :- it returns complex conjugate of a complex number.

Program

Write a program to addition, subtraction and multiplication of two complex number.

```
#include<iostream.h>
#include<conio.h>
class complex
{
public:
        int real,imag;
        void in()
        {
        cout<<"enter real part :";
        cin>>real;
        cout<<"enter imaginary part:";
        cin>>imag;
        }
};
void main()
{
clrscr();
complex a,b,c;
a.in();
b.in();
cout<<"\n addition of two complex number :";
c.real=a.real+b.real;
c.imag=a.imag+b.imag;
cout<<c.real<<"+"<<c.imag<<"i";
cout<<"\n subtraction of two complex number :";
c.real=a.real-b.real;
c.imag=a.imag-b.imag;
```

```
cout<<c.real<<"+"<<c.imag<<"i";
cout<<"\n multiplicaton of two complex number :";
c.real=a.real*b.real-a.imag*b.imag;
c.imag=a.imag*b.real+a.real*b.imag;
cout<<c.real<<"+"<<c.imag<<"i";
getch();
}
```

## 4.6 Const Argument

- The const variable can be declared using “const” keyword.
- This keyword makes the value of the variable stable.
- It assign an initial value to a variable that can’t be changed later on by the program.