OBJECT ORIENTED PROGRAMMING WITH C++(BCS306B)

Module – 1
Chapter 1 :

An Overview of C++

SYLLABUS:

An overview of C++: What is object-Oriented Programming? Introducing C++ Classes, The General Form of a C++ Program.
Classes and Objects: Classes, Friend Functions, Friend Classes, Inline Functions, Parameterized Constructors, Static Class Members,
When Constructors and Destructors are Executed, The Scope Resolution Operator, Passing Objects to functions, Returning Objects,
Object Assignment.

What is C++

C++ is a high-level, object oriented, general-purpose programming language created by Danish computer scientist Bjarne Stroustrup.

1. Operating Systems

Mac OS X has large amounts of code written in C++. Most of the software from Microsoft like Windows, Microsoft Office, IDE Visual Studio, and Internet Explorer are also written in C++.

2. Games

It is used in 3D games and multiplayer networking.

3. GUI Based Applications

C++ is also used to develop GUI-based and desktop applications. Most of the applications from Adobe such as Photoshop, Illustrator, etc. are developed using C++.

4. Web Browsers

Mozilla Firefox is completely developed from C++. Google applications like Chrome and Google File System are partly written in C++.

5. Embedded Systems

Various embedded systems that require the program to be closer to hardware such as smartwatches, medical equipment systems, etc., are developed in C++.

6. Banking Applications

Infosys Finacle is a popular banking application developed using C++.

7. Compilers

The compilers of many programming languages are developed in C and C++.

8. Database Management Software

The world's most popular open-source database MySQL, is written in C++.

Differences between C and C++

	C Language	C++ Language
1.	C Is a Procedure Oriented Language	C++ is an object oriented programming language.
2.	C makes use of top down approach of problem solving.	C++ makes use of bottom up approach of problem solving.
3.	The input and output is done using scanf and printf statements.	The input and output is done using cin and cout statements.
4.	The I/O operations are supported by stdio.h header file.	The I/O operations are supported by iostream.h header file.
5.	C does not support inheritance, polymorphism, class and object concepts.	C++ supports inheritance, polymorphism, class and object concepts.
6.	The data type specifier or format specifier (%d, %f, %c) is required in printf and scanf functions.	The format specifier is not required in cin and cout functions.
7.	Data hiding is not possible.	Data hiding can be done by making it private.
8.	Data reusability is not possible.	Data reusability is possible.

```c
// C program to add two numbers
#include <stdio.h>

int main()
{
    int A, B, sum = 0;
    // Ask user to enter the two numbers
    printf("Enter two numbers A and B : \n");
    // Read two numbers from the user || A = 2, B = 3
    scanf("%d%d", &A, &B);
    // Calculate the addition of A and B
    // using '+' operator
    sum = A + B;
    // Print the sum
    printf("Sum of A and B is: %d", sum);
    return 0;
}
```

* Single-line comments start with two forward slashes (//)
* Multi-line comments start with /* and ends with */.

```cpp
// C++ program to add two numbers
#include<iostream>
using namespace std;
int main()
{
int A, B, sum;
cout << "Please enter the First Number : "<<endl;
cin >> A;
cout << "Please enter the Second Number : "<<endl;
cin >> B;
sum = A + B;
cout << "Sum of Two Numbers " << A <<" and " << B << " = 
" << sum;
return 0;
}
```

> **C++ manipulator endl** function is used to insert a new line character and flush the stream.

- ➢ **iostream** stands for standard input-output stream. This header file contains definitions of objects like cin, cout, cerr, etc.
- ➢ The **std** is a short form of standard, the **std** namespace contains the built-in classes and declared functions.
- ➢ Single-line comments start with two forward slashes (//)
- ➢ Multi-line comments start with /* and ends with */.
- ➢ cin uses the insertion operator(>>) while cout uses the extraction operator(<<).

```cpp
// C++ Program to show the syntax/working of Objects as a
// part of Object Oriented programming

#include <iostream>
using namespace std;

class person {
    char name[20];
    int id;

public:
    void getdetails() {}
};

int main()
{
    person p1; // p1 is a object
    return 0;
}
```

➢ public - members are accessible from outside the class.
➢ private - members cannot be accessed (or viewed) from outside the class.
➢ protected - members cannot be accessed from outside the class,
➢ however, they can be accessed in inherited classes.

Q : Differentiate between Procedure Oriented Programming and Object Oriented Programming.

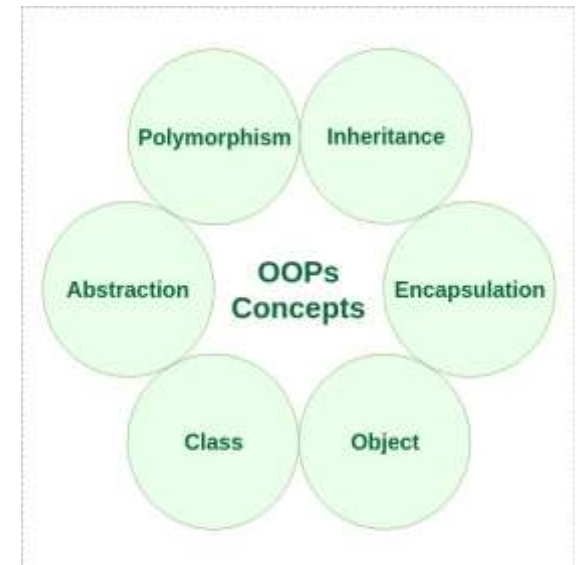| Procedural-Oriented Programming	Object-Oriented Programming
It is known as POP.	It is known as OOP.
Procedural programming languages are not as faster as object-oriented.	The object-oriented programming languages are faster and more effective.
Procedural uses procedures, modules, procedure calls.	Object-oriented uses objects, classes, messages.
Top-down approach.	Bottom-up approach.
Access modifiers are not supported.	Access modifiers are supported.
Functions are preferred over data.	Security and accessibility are preferred.
If the size of the problem is small, POP is preferred.	If the size of the problem is big, OOP is preferred.
C++, C#, Java, Python, etc. are the examples of OOP languages.	C, BASIC, COBOL, Pascal, etc. are the examples POP languages.

1.1 What is Object-oriented programming ?

- The word **object-oriented** is the combination of two words i.e. **object** and **oriented**.

- The **object-oriented programming** is basically a computer programming design methodology that models software design around data or objects rather than functions and logic.

- The main aim of OOP is to bind together the data and the functions that operate on them.

There are some basic concepts that act as the building blocks of OOPs i.e.

- Class
- Objects
- Encapsulation
- Abstraction
- Polymorphism

- Inheritance
- Dynamic Binding
- Message Passing

Q: Explain the salient features of object oriented programming languages.

1. CLASS

- The building block of C++ that leads to Object-Oriented programming is a Class.

- It is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object.

- For Example: Consider the Class of Cars. There may be many cars with different names and brands but all of them will share some common properties like all of them will have 4 wheels, Speed Limit, Mileage range, etc. So here, the Car is the class, and wheels, speed limits, and mileage are their properties.

2. OBJECT

- An Object is an identifiable entity with some characteristics and behavior. An Object is an instance of a Class.

- When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.

3. ENCAPSULATION

- Encapsulation is defined as wrapping up data and information under a single unit.

- In Object-Oriented Programming, Encapsulation is defined as binding together the data and the functions that manipulate them.

Encapsulation in C++

Methods   Variables

Class
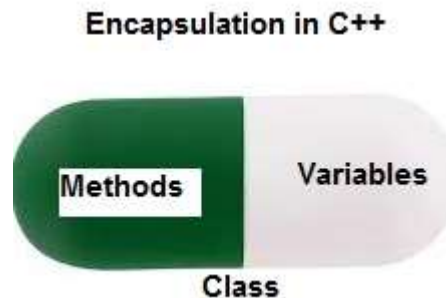
4. ABSTRACTION

- Abstraction means displaying only essential information and hiding the details.

- Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.

5. POLYMORPHISM

- The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form. A person at the same time can have different characteristics.

- *Operator Overloading*: The process of making an operator exhibit different behaviors in different instances is known as operator overloading.

- *Function Overloading*: Function overloading is using a single function name to perform different types of tasks. Polymorphism is extensively used in implementing inheritance.

6. INHERITANCE

- The capability of a class to derive properties and characteristics from another class is called Inheritance.

- **Sub Class**: The class that inherits properties from another class is called Sub class or Derived Class.

- **Super Class**: The class whose properties are inherited by a sub-class is called Base Class or Superclass.

- **Reusability**: Inheritance supports the concept of "reusability", i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

Example: Dog, Cat, Cow can be Derived Class of Animal Base Class.
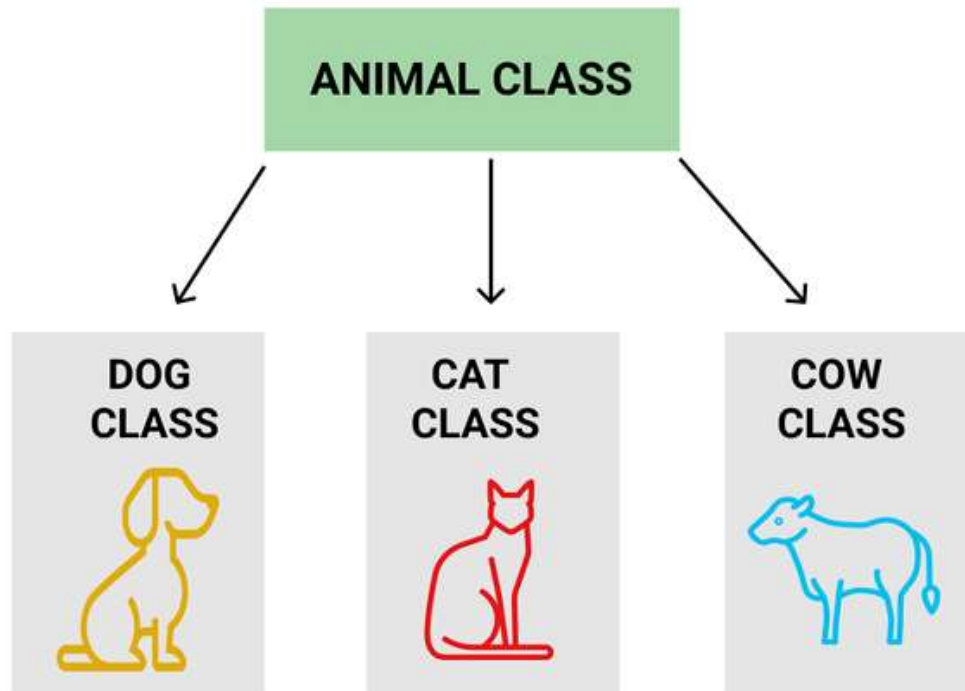
7. DYNAMIC BINDING

- In dynamic binding, the code to be executed in response to the function call is decided at runtime.

- C++ has virtual functions to support this. Because dynamic binding is flexible, it avoids the drawbacks of static binding, which connected the function call and definition at build time.
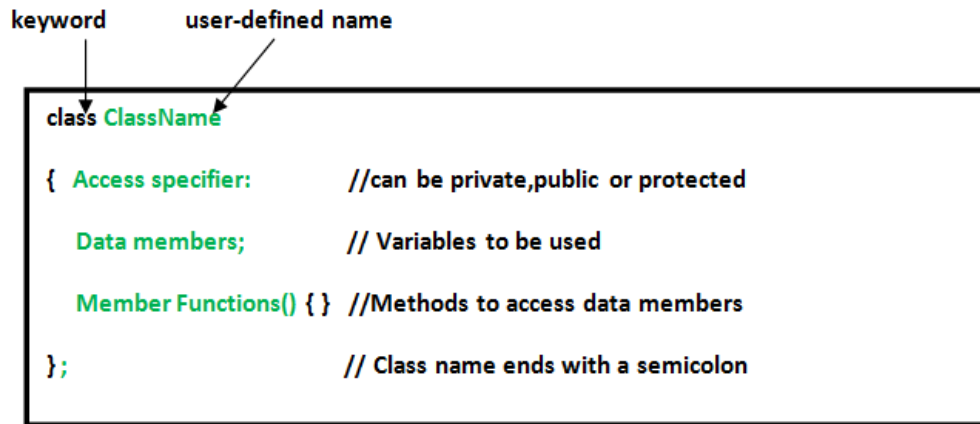
8. MESSAGE PASSING

- Objects communicate with one another by sending and receiving information.

- A message for an object is a request for the execution of a procedure and therefore will invoke a function in the receiving object that generates the desired results.

1.2 INTRODUCING C++ CLASSES

- **Class in C++** is the building block that leads to Object-Oriented programming. It is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class.

- A C++ class is like a blueprint for an object.

- An **Object** is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.

❑ Defining Class

A class is defined in C++ using the keyword class followed by the name of the class. The body of the class is defined inside the curly brackets and terminated by a semicolon at the end.

```
keyword          user-defined name

class ClassName
{  Access specifier:        //can be private,public or protected

   Data members;            // Variables to be used

   Member Functions() { }  //Methods to access data members

};                          // Class name ends with a semicolon
```

❑ Declaring Objects

When a class is defined, only the specification for the object is defined; no memory or storage is allocated. To use the data and access functions defined in the class, you need to create objects.

Syntax

ClassName ObjectName;

❑ **Accessing member functions**: The data members and member functions of the class can be accessed using the dot('.') operator with the object. For example, if the name of the object is *obj* and you want to access the member function with the name *printName()* then you will have to write *obj.printName()*.

❑ **Accessing Data Members**

The public data members are also accessed in the same way given however the private data members are not allowed to be accessed directly by the object. Accessing a data member depends solely on the access control of that data member.

There are three access modifiers: **public, private, and protected.**

```cpp
// C++ program to demonstrate accessing of data members
#include <iostream.h>
using namespace std;

class AIT {
    // Access specifier
public:
    // Data  Members
    string CSE;
    // Member Functions()
    void printname() { cout << "CSE name is:" << CSE; }
};
int main()
{

    // Declare an object of class geeks
    AIT obj1;
    // accessing data member
    obj1.CSE = "3RD A & B";
    // accessing member function
    obj1.printname();
    return 0;

}
```

1.3 GENERAL FORM OF C++ PROGRAM

Although individual styles will differ, most C++ programs will have this general form:

```
#includes
base-class declarations
derived class declarations
nonmember function prototypes
int main( )
{
//...
}
nonmember function definitions
```

In most large projects, all **class** declarations will be put into a header file and included with each module. But the general organization of a program remains the same.

Module – 1
Chapter 2 :
Classes and Objects

1.4 CLASSES :

Classes are created using the keyword **class**. A class declaration defines a new type that links code and data. This new type is then used to declare objects of that class.

```
class class-name {
            private data and functions
            access-specifier:
            data and functions
            access-specifier:
            data and functions
            // ...
            access-specifier:
            data and functions
            } object-list;
```

The *object-list* is optional. If present, it declares objects of the class.

Create a Class :

To create a class, use the class keyword:

Example

Create a class called "MyClass":

```cpp
class MyClass {        // The class
  public:              // Access specifier
    int myNum;         // Attribute (int variable)
    string myString;   // Attribute (string variable)
};
```

Create an Object :

❑ To create an object of MyClass, specify the class name, followed by the object name.

❑ To access the class attributes (myNum and myString), use the dot syntax (.) on the object:

Example :

Create an object called "myObj" and access the attributes:

```cpp
class MyClass {          // The class
  public:                // Access specifier
    int myNum;           // Attribute (int variable)
    string myString;     // Attribute (string variable)
};

int main() {
  MyClass myObj;   // Create an object of MyClass

  // Access attributes and set values
  myObj.myNum = 15;
  myObj.myString = "Some text";

  // Print attribute values
  cout << myObj.myNum << "\n";
  cout << myObj.myString;
  return 0;
}
```

❏ **ACCESS-SPECIFIER** is one of these three C++ keywords:

      i) **private**   **ii) public**   **iii) protected**

▪ By default, functions and data declared within a class are **private** to that class and may be accessed only by other members of the class.

▪ The **public** access specifier allows functions or data to be accessible to other parts of your program.

▪ The **protected** access specifier is needed only when inheritance is Involved.

▪ A **protected** member variable or function is very similar to a private member but it provided one additional benefit that they can be accessed in child classes which are called derived classes.

▪ Once an access specifier has been used, it remains in effect until either another access specifier is encountered or the end of the class declaration is reached.

- A class can have multiple public, protected, or private labeled sections. Each section remains in effect until either another section label or the closing right brace of the class body is seen.

- The default access for members and classes is private.

```cpp
class Base {
            public: // public members go here
            protected: // protected members go here
            private: // private members go here
                };
```

1.5 FRIEND FUNCTION

❑ Data hiding is a fundamental concept of object-oriented programming. It restricts the access of private members from outside of the class.

❑ Similarly, protected members can only be accessed by derived classes and are inaccessible from outside. For example,

```cpp
class MyClass {
     private: int member1;
              }
int main()
     {
     MyClass obj;//Error! Cannot access private members
      obj.member1 = 5;
     }
```

- However, there is a feature in C++ called **friend functions** that break this rule and allow us to access member functions from outside the class.

❑ **friend Function in C++**

- A **friend function** can access the **private** and **protected** data of a class. We declare a friend function using the friend keyword inside the body of the class.

```
class className {

            ... .. ...

            friend returnType functionName(arguments);

            ... .. ...

            }
```

```cpp
#include <iostream>
using namespace std;

class demo
{
  private:
  int a=5;
  friend void display(demo);//friend function declaration
};

void display(demo d)            //friend function defination
{
  cout<<"a: "<<d.a<<endl;    //access of private data
}

int main()
{
  demo d;
  display(d);                       //calling a friend function
return 0;
}
```

1.6 FRIEND CLASSES

We can also use a friend Class in C++ using the friend keyword. For example,

```cpp
class ClassB;
class ClassA {
            // ClassB is a friend class of ClassA
            friend class ClassB; ... .. ...
               }


class ClassB {
            ... .. ...
            }
```

- When a class is declared a friend class, all the member functions of the friend class become friend functions.

- Since ClassB is a friend class, we can access all members of ClassA from inside ClassB.

- However, we cannot access members of ClassB from inside ClassA. It is because friend relation in C++ is only granted, not taken.

```cpp
#include <iostream>
using namespace std;
class test1
{
  int a,b;
  public:
  void get()
  {
 cout<<"enter 2 integers";
 cin>>a>>b;
  }
  friend class test2;
//defining friend class
};

class test2
{
    test1 t1;
    public:
    void put()
  {
      t1.get();
      cout<<"a: "<<t1.a<<endl;
        cout<<"b: "<<t1.b<<endl;
  }
};

int main()
{
    test2 t2;
    t2.put();
}
```

1.7 INLINE FUNCTIONS

In C++, we can declare a function as inline. This copies the function to the location of the function call in compile-time and may make the program execution faster.

To create an inline function, we use the inline keyword. For example

```
inline returnType functionName(parameters) { // code }
```

Notice the use of keyword inline before the function definition
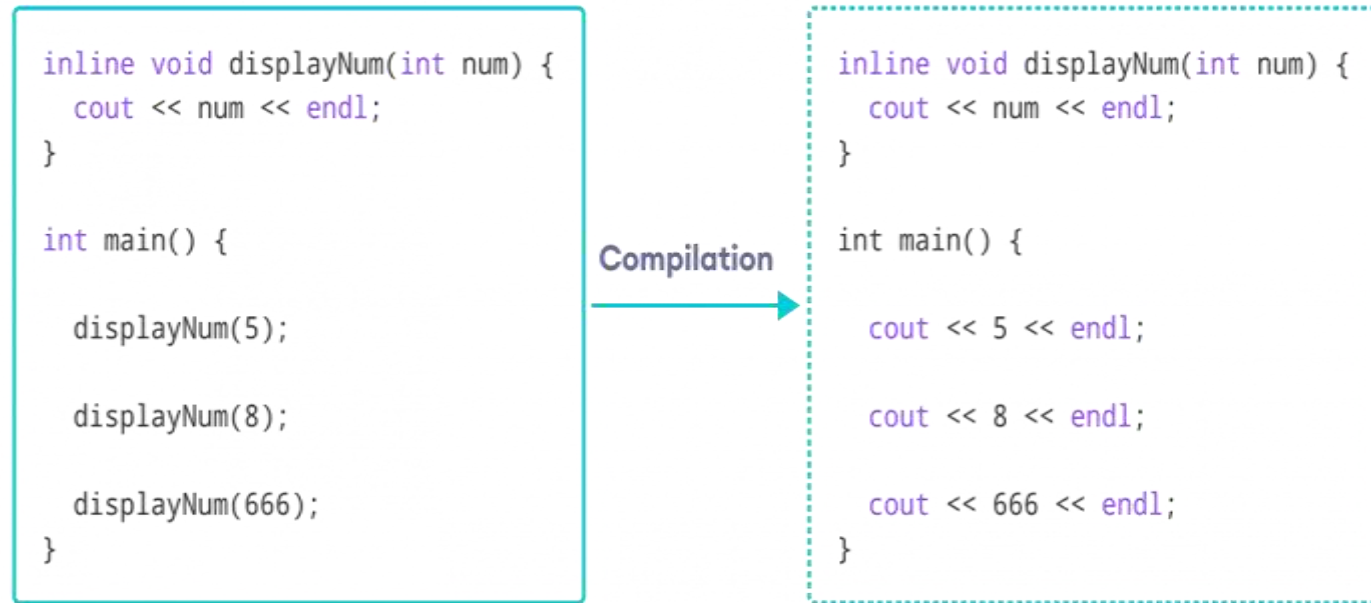
```cpp
#include <iostream>
using namespace std;
inline void displayNum(int num)
{    cout << num << endl;}
int main() {
        // first function call
        displayNum(5);
         // second function call
        displayNum(8);
        // third function call
        displayNum(666);
        return 0;
}
Output: 5 8 666
```

Here is how this program works:

```cpp
inline void displayNum(int num) {
  cout << num << endl;
}

int main() {

  displayNum(5);

  displayNum(8);

  displayNum(666);
}
```

Compilation →

```cpp
inline void displayNum(int num) {
  cout << num << endl;
}

int main() {

  cout << 5 << endl;

  cout << 8 << endl;

  cout << 666 << endl;
}
```

Working of inline functions in C++

Here, we created an inline function named displayNum() that takes a single integer as a parameter.

We then called the function 3 times in the main() function with different arguments. Each time displayNum() is called, the compiler copies the code of the function to that call location.

1.8 Parameterized Constructors :

A constructor is a special type of member function that is called automatically when an object is created.

In C++, a constructor has the same name as that of the class and it does not have a return type. For example,

```
class Wall {
        public:
        Wall() // create a constructor
        { // code }
        };
```

Here, the function Wall() is a constructor of the class Wall. constructor

• has the same name as the class,

• does not have a return type, and

• is public

In C++, a constructor with parameters is known as a parameterized constructor.

This is the preferred method to initialize member data.

```cpp
#include <iostream>
using namespace std;
 class A  {
  private:
          int num1, num2 ;
  public:
      A(int n1, int n2) {
                  num1 = n1;
                  num2 = n2;       }
  void display() {
    cout<<"num1 = "<< num1 <<endl;
    cout<<"num2 = "<< num2 <<endl;
                  }};
```

```cpp
int main()

{

    A obj(3,8);

    obj.display();

    return 0;

}
```

Output :
```
      num1 = 3
      num2 = 8
```

1.9 STATIC CLASS MEMBERS

a. Static data members

- Static data members are class members that are declared using **static** keywords.

- Only one copy of that member is created for the entire class and is shared by all the objects of that class, no matter how many objects are created.

- It is initialized before any object of this class is created, even before the main starts.

- It is visible only within the class, but its lifetime is the entire program.

Syntax:

```
static data_type data_member_name;
```

```cpp
#include <iostream>
using namespace std;

class Circle
{
  private:
    float _radius;
    static float _pi;
  public:
    Circle(float radius)
    {
      _radius = radius;
    }
 float Area()
  {
return _pi * _radius *
_radius;
  }
};

float Circle::_pi = 3.14 f;
 // static data definition
int main()
{
  Circle c1(4);
cout<< " Area = "<< c1.Area();
  return 0;
}
```

OUTPUT :

Area = 50.24

b. Static Member Functions

- A static member function is a member function of a class that can be called without creating an object of the class.

- It is defined using the "static" keyword, and it is typically declared within the class definition but outside of any member function.

- A static member function can only access static data members of the class, and it cannot access non-static data members or member functions of the class.

```
class ClassName {
  // other members
  static return_type function_name(parameter list);
 };
```

- Where "ClassName" is the name of the class, "return_type" is the return type of the function, "function_name" is the name of the function, and "parameter list" is the list of parameters passed to the function.

```cpp
#include <iostream>
using namespace std;

class Test
{
  public:
    static void Display();
    // static member function declaration
};
void Test::Display()
{
  cout << "This is a static member function";
}
int main()
{
  Test::Display();
  return 0;
}
```

OUTPUT :
This is a static member function

1.10 WHEN CONSTRUCTORS AND DESTRUCTORS ARE EXECUTED

- A <u>constructor</u> is a particular member function having the same name as the class name. It calls automatically whenever the object of the class is created.

- <u>Destructors</u> have the same class name preceded by (~) tilde symbol. It removes and destroys the memory of the object, which the constructor allocated during the creation of an object.

- A constructor is a function that initializes the object of the class and allocates the memory location for an object,

- Destructor also has the same name as the class name, denoted by tilted ~ symbol, known for destroying the constructor, deallocates the memory location for created by the constructor.

- One class can have more than one constructor but have one destructor.

Syntax:

The syntax of the constructor in C++ are given below.

```
class class_name
{
   ……….
    public

class_name([parameter list])
        {
                …………….
        }
};
```

In the above-given syntax, class_name is the constructor's name, the public is an access specifier, and the parameter list is optional.

Syntax:

The syntax of destructor in C++ are given below.

```
class class_name
{
    …………….;
    …………….;
public:
    xyz();   //constructor
    ~xyz();   //destructor
};
```

Here, we use the tilde symbol for defining the destructor in C++ programming.

The Destructor has no argument and does not return any value, so it cannot be overloaded.

There some features which distinguish between a constructor and a normal member function

- Constructor has same name as the class itself

- Constructors do not have any return type

- A constructor is automatically called when an object is created.

- If we do not specify a constructor while defining a class, the C++ compiler generates a default constructor for us without any parameters and an empty body.

The destructor is usually called when:-

- The program finished execution.

- When a scope containing local variable ends.

- When you call the delete operator.

This program illustrates when constructors and destructors are executed:

```cpp
#include <iostream.h>
#include<conio.h>
 class MyClass
 {
     public:
      int x;
      MyClass();    // constructor
      ~MyClass();   // destructor
 };

 // Implement MyClassconstructor.
        MyClass::MyClass()
      {
              x = 10;
      }
```

```cpp
// Implement MyClass destructor.

 MyClass::~MyClass()
        {
                cout<< "Destructing ...\n";
        }

void main()
        {
                clrscr();
                MyClass ob1;
                MyClass ob2;
                cout <<ob1.x<< " " <<ob2.x<<"\n";
                getch();
        }
```

Output:
 10 10

S.No.	Constructors	Destructors
1.	The constructor initializes the class and allots the memory to an object.	If the object is no longer required, then destructors demolish the objects.
2.	When the object is created, a constructor is called automatically.	When the program gets terminated, the destructor is called automatically.
3.	It receives arguments.	It does not receive any argument.
4.	A constructor allows an object to initialize some of its value before it is used.	A destructor allows an object to execute some code at the time of its destruction.
5.	It can be overloaded.	It cannot be overloaded.
6.	When it comes to constructors, there can be various constructors in a class.	When it comes to destructors, there is constantly a single destructor in the class.
7.	They are often called in successive order.	They are often called in reverse order of constructor.

1.11 THE SCOPE RESOLUTION OPERATOR

❑ It is used to access the global variables or member functions of a program when there is a local variable or member function with the same name.

❑ It is used to define the member function outside of a class.

❑ The operator is represented as the double colon (::) symbol.

❑ For example, when the global and local variable or function has the same name in a program, and when we call the variable, by default it only accesses the inner or local variable without calling the global variable. In this way, it hides the global variable or function.

❑ To overcome this situation, we need to use the scope resolution operator to fetch a program's hidden variable or function.

Example to Understand Scope Resolution Operator in C++:

```cpp
#include <iostream>
using namespace std;

class Rectangle
{
    private:
        int length;
        int breadth;
    public:
    Rectangle (int l, int b)
        {
            length = l;
            breadth = b;
        }
    int area ()
    {
      return length * breadth;
    }
int perimeter ();
};
```

```cpp
int Rectangle::perimeter()
{
   return 2*(length + breadth);
}

int main()
{
 Rectangle r(8, 3);
 cout << "Area is :"<< r.area() <<endl;
 cout << "Perimeter is :"<<
 r.perimeter();
}
```

Output:

Area is : 24
Perimeter is :22

- To make you understand these two methods, please have a look at the below Rectangle class. Here, the Rectangle class has two private data members i.e. length and breadth.

- The Rectangle class also has one parameterized constructor to initialize the length and breadth data members. But here, from the below class let us keep the focus on the two methods i.e. area and perimeter.

1.12 Passing Objects to functions :

In C++ programming, we can pass objects to a function in a similar manner as passing regular arguments.

Syntax: **function_name (object_name)**

```cpp
#include<iostream>
using namespace std;
class A {
   public:
           int age = 20;
           char ch = 'A';

   // function that has objects as parameters

   void display(A &a) {
       cout << "Age = " << a.age << endl;
       cout << "Character = " << a.ch << endl;
     }
};
```

```
int main()
{
    A obj;
    obj.display(obj);
    return 0;
}
```

Output
Age = 20
Character = A

Pass by Value

This creates a shallow local copy of the object in the function scope. Things you modify here won't be reflected in the object passed to it.

Pass by Reference

This passes a reference to the object to the function. Things you modify here will be reflected in the object passed to it. No copy of the object is created.

An object can be passed to a function just like we pass structure to a function.

Here in class A we have a function display() in which we are passing the object of class A.

Similarly we can pass the object of another class to a function of different class.

1.13 Returning Objects

A function may return an object to the caller

```cpp
#include <iostream>
using namespace std;

class Student
{
   public:
     double marks1, marks2;
};

// function that returns object of Student
Student createStudent() {
    Student student;

   // Initialize member variables of Student
    student.marks1 = 96.5;
    student.marks2 = 75.0;
```

```cpp
// print member variables of Student
    cout << "Marks 1 = " << student.marks1 << endl;
    cout << "Marks 2 = " << student.marks2 << endl;

    return student;
}

int main() {
    Student student1;

    // Call function
    student1 = createStudent();

    return 0;
}
```

Output
Marks 1 = 96.5
Marks 2 = 75.0

```cpp
#include<iostream>

class Student {...};

Student createStudent() {
    Student student;
    ... ...
    return student;
}

int main() {
    ... ...
    student1 = createStudent();
    ... ...
}
```

function call

In this program, we have created a function createStudent() that returns an object of Student class.

We have called createStudent() from the main() method.

Call function student1 = createStudent();

Here, we are storing the object returned by the createStudent() method in the student1.

1.14 Object Assignment

Assuming that both objects are of the same type, you can assign one object to another. This causes the data of the object on the right side to be copied into the data of the object on the left. For example, this program displays **99**:

```cpp
#include <iostream>
using namespace std;
class myclass
{
    int i;
        public:
    void seti(int n)    { i=n; }
    int geti()    { return i; }
};
```

```cpp
int main()
{
    myclass ob1, ob2;
    ob1.seti(99);
    ob2 = ob1;      // assign data from ob1 to ob2
cout << "This is ob2's i: " << ob2.geti();
return 0;
}
```

Output:

This is ob2's I : 99