

CONTENT:

1. **Introduction**
2. **Class**
3. **Objects**
4. **Encapsulation**
5. **Abstraction**
6. **Polymorphism**
7. **Inheritance**
8. **Dynamic Binding**
9. **Message Passing**

Object-oriented programming – As the name suggests uses objects in programming. Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism, etc in programming. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

Characteristics of an Object Oriented Programming language

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 brand but all of them will share some common properties like all of them will have 4 wheels, Speed Limit, Mileage range etc. So here, Car is the class and wheels, speed limits, mileage are their properties.

- A **Class** is a user-defined data-type which has data members and member functions.
- **Data members** are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions define the properties and behaviour of the objects in a Class.

In the above example of class Car, the data member will be speed limit, mileage etc and member functions can apply brakes, increase speed etc.

We can say that a Class in C++ is a blue-print representing a group of objects which shares some common properties and behaviours.

Object:

An Object is an identifiable entity with some characteristics and behaviour. 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.

```
class person
{
    char name[20];
    int id;
public:
    void getdetails(){}
};

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

Object take up space in memory and have an associated address like a record in pascal or structure or union in C.

When a program is executed the objects interact by sending messages to one another.

Each object contains data and code to manipulate the data. Objects can interact without having to know details of each other's data or code, it is sufficient to know the type of message accepted and type of response returned by the objects.

Encapsulation:

In normal terms, Encapsulation is defined as wrapping up of 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.

Consider a real-life example of encapsulation, in a company, there are different sections like the accounts section, finance section, sales section etc. The finance section handles all the financial transactions and keeps records of all the data related to finance. Similarly, the sales section handles all the sales-related activities and keeps records of all the sales. Now there may arise a situation when for some reason an official from the finance section needs all the data about sales in a particular month. In this case, he is not allowed to directly access the data of the sales section. He will first have to contact some other officer in the sales section and then request him to give the particular data. This is what encapsulation is. Here the data of the sales section and the employees that can manipulate them are wrapped under a single name "sales section".

Encapsulation also leads to *data abstraction or hiding*. As using encapsulation also hides the data. In the above example, the data of any of the section like sales, finance or accounts are hidden from any other section.

Abstraction:

Data abstraction is one of the most essential and important features of object-oriented programming in C++. 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.

Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of the car or applying brakes will stop the car but he does not know about how on pressing accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of accelerator, brakes etc in the car. This is what abstraction is.

- **Abstraction using Classes:** We can implement Abstraction in C++ using classes. The class helps us to group data members and member functions using available access specifiers. A Class can decide which data member will be visible to the outside world and which is not.
- **Abstraction in Header files:** One more type of abstraction in C++ can be header files. For example, consider the pow() method present in math.h header file. Whenever we need to calculate the power of a number, we simply call the function pow() present in the math.h header file and pass the numbers as arguments without knowing the underlying algorithm according to which the function is actually calculating the power of numbers.

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 characteristic. Like a man at the same time is a father, a husband, an employee. So the same person possess different behaviour in different situations. This is called polymorphism.

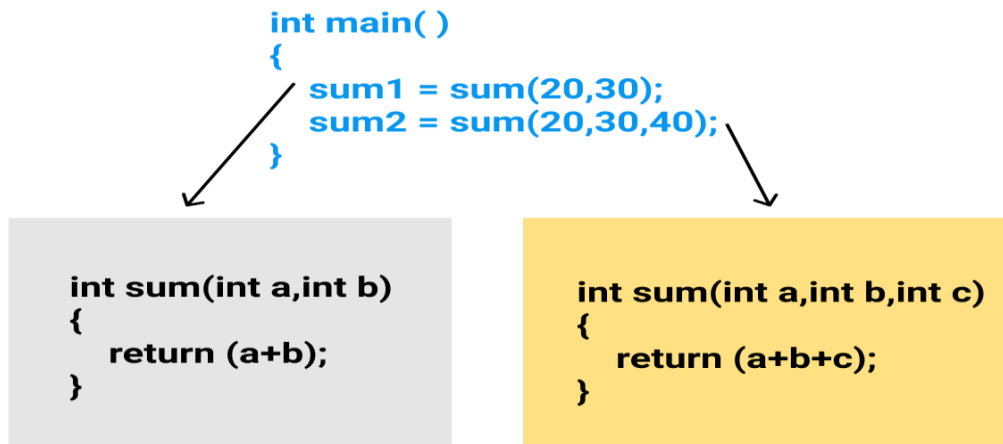
An operation may exhibit different behaviours in different instances. The behaviour depends upon the types of data used in the operation.

C++ supports operator overloading and function overloading.

- **Operator Overloading:** The process of making an operator to exhibit different behaviours 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.

Example: Suppose we have to write a function to add some integers, sometimes there are 2 integers; sometimes there are 3 integers. We can write the Addition Method with the same name having different parameters; the concerned method will be called according to parameters.

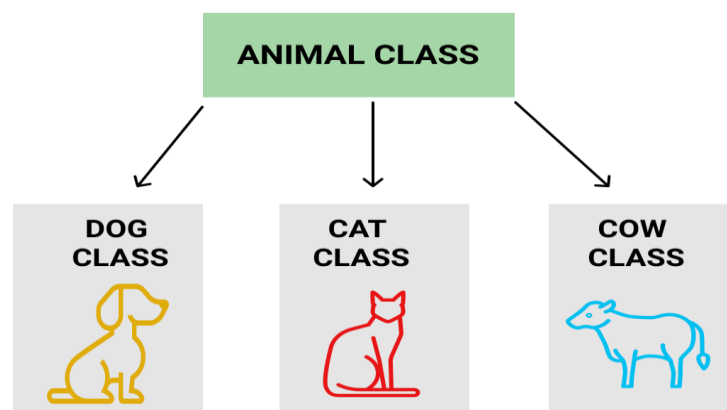


Inheritance:

The capability of a class to derive properties and characteristics from another class is called Inheritance. Inheritance is one of the most important features of Object-Oriented Programming.

- **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 sub class is called Base Class or Super class.
- **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.



Dynamic Binding: In dynamic binding, the code to be executed in response to function call is decided at runtime. C++ has virtual functions to support this.

Message Passing: Objects communicate with one another by sending and receiving information to each other. A message for an object is a request for execution of a procedure and therefore will invoke a function in the receiving object that generates the desired results. Message passing involves specifying the name of the object, the name of the function and the information to be sent.

Functions

A function is a set of statements that take inputs, do some specific computation and produces output. The idea is to put some commonly or repeatedly done task together and make a function so that instead of writing the same code again and again for different inputs, we can call the function.

Example:

Below is a simple C program to demonstrate functions.

```
#include <stdio.h>

// An example function that takes two parameters 'x' and 'y'
// as input and returns max of two input numbers
int max(int x, int y)
{
    if (x > y)
        return x;
    else
        return y;
}

// main function that doesn't receive any parameter and
// returns integer.
int main(void)
{
    int a = 10, b = 20;

    // Calling above function to find max of 'a' and 'b'
    int m = max(a, b);

    printf("m is %d", m);
    return 0;
}
```

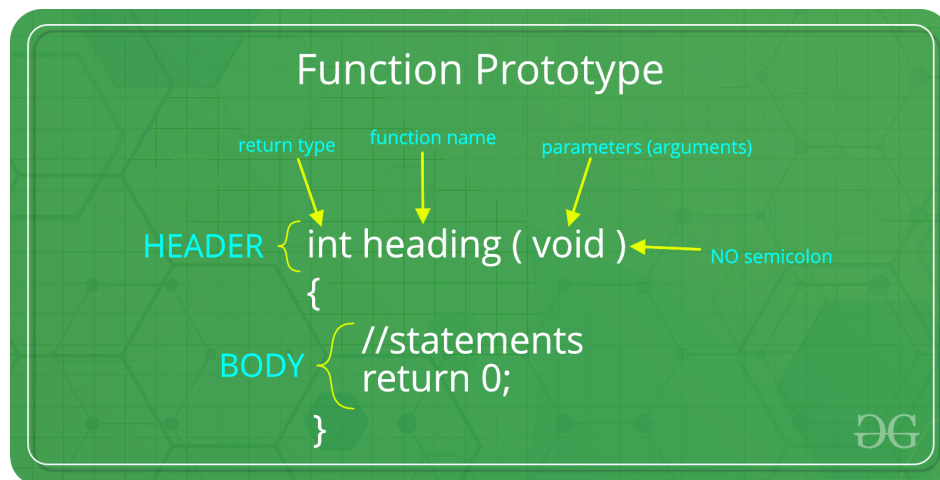
Output: m is 20

Why do we need functions?

- Functions help us in reducing code redundancy. If functionality is performed at multiple places in software, then rather than writing the same code, again and again, we create a function and call it everywhere. This also helps in maintenance as we have to change at one place if we make future changes to the functionality.
- Functions make code modular. Consider a big file having many lines of codes. It becomes really simple to read and use the code if the code is divided into functions.
- Functions provide abstraction. For example, we can use library functions without worrying about their internal working.

Function Declaration

A function declaration tells the compiler about the number of parameters function takes, data-types of parameters and return type of function. Putting parameter names in function declaration is optional in the function declaration, but it is necessary to put them in the definition. Below are an example of function declarations. (parameter names are not there in below declarations)



- 1) `int max(int, int);` // A function that takes two integers as parameters and returns an integer
- 2) `int *swap(int*,int);` // A function that takes a int pointer and an int variable as parameters and returns an pointer of type int
- 3) `char *call(char b);` // A function that takes a charas parameters and returns an reference variable
- 4) `int fun(char, int);` // A function that takes a char and an int as parameters and returns an integer

It is always recommended to declare a function before it is used

In C, we can do both declaration and definition at the same place, like done in the above example program.

C also allows to declare and define functions separately, this is especially needed in case of library functions. The library functions are declared in header files and defined in library files. Below is an example declaration.

Parameter Passing to functions

1) The parameters passed to function are called ***actual parameters***.

For example, in the above program 10 and 20 are actual parameters.

2) The parameters received by function are called ***formal parameters***.

For example, in the above program x and y are formal parameters.

There are two most popular ways to pass parameters.

Pass by Value: In this parameter passing method, values of actual parameters are copied to function's formal parameters and the two types of parameters are stored in different memory locations. So any changes made inside functions are not reflected in actual parameters of caller.

Pass by Reference Both actual and formal parameters refer to same locations, so any changes made inside the function are actually reflected in actual parameters of caller.

In C++, parameters are always passed by value. Parameters are always passed by value in C++.

For example. in the below code, value of x is not modified using the function fun().

```
#include <iostream>
using namespace std;
void fun(int x) {
    x = 30;
}
int main() {
    int x = 20;
    fun(x);
    cout << "x = " << x;
    return 0;
}
```

Output: x = 20

However, in C, we can use pointers to get the effect of pass by reference. For example, consider the below program. The function fun() expects a pointer ptr to an integer (or an address of an integer). It modifies the value at the address ptr. The dereference operator * is used to access the value at an address. In the statement '*ptr = 30', value at address ptr is changed to 30. The address operator & is used to get the address of a variable of any data type. In the function call statement 'fun(&x)', the address of x is passed so that x can be modified using its address.

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

void fun(int *ptr)
{
    *ptr = 30;
}

int main() {
    int x = 20;
    fun(&x);
    cout << "x = " << x;

    return 0;
}
```

The main purpose of C++ programming is to add object orientation to the C programming language and classes are the central feature of C++ that supports object-oriented programming and are often called user-defined types.

A class is used to specify the form of an object and it combines data representation and methods for manipulating that data into one neat package. The data and functions within a class are called members of the class.

Class Definitions

When you define a class, you define a blueprint for a data type. This doesn't actually define any data, but it does define what the class name means, that is, what an object of the class will consist of and what operations can be performed on such an object.

A class definition starts with the keyword class followed by the class name; and the class body, enclosed by a pair of curly braces. A class definition must be followed either by a semicolon or a list of declarations. For example, we defined the Box data type using the keyword class as follows –

```

class Box {
public:
    double length; // Length of a box
    double breadth; // Breadth of a box
    double height; // Height of a box
};

```

The keyword public determines the access attributes of the members of the class that follows it. A public member can be accessed from outside the class anywhere within the scope of the class object. You can also specify the members of a class as private or protected which we will discuss in a sub-section.

Define C++ Objects

A class provides the blueprints for objects, so basically an object is created from a class. We declare objects of a class with exactly the same sort of declaration that we declare variables of basic types. Following statements declare two objects of class Box –

```

Box Box1;    // Declare Box1 of type Box
Box Box2;    // Declare Box2 of type Box

```

Both of the objects Box1 and Box2 will have their own copy of data members.

Accessing the Data Members

The public data members of objects of a class can be accessed using the direct member access operator (.). Let us try the following example to make the things clear

```

include <iostream>

#using namespace std;

class Box {
public:
    double length; // Length of a box
    double breadth; // Breadth of a box
    double height; // Height of a box
};

int main() {
    Box Box1;    // Declare Box1 of type Box
    Box Box2;    // Declare Box2 of type Box
    double volume = 0.0; // Store the volume of a box here

    // box 1 specification
    Box1.height = 5.0;
    Box1.length = 6.0;
    Box1.breadth = 7.0;
}

```

```

// box 2 specification
Box2.height = 10.0;
Box2.length = 12.0;
Box2.breadth = 13.0;

// volume of box 1
volume = Box1.height * Box1.length * Box1.breadth;
cout << "Volume of Box1 : " << volume <<endl;

// volume of box 2
volume = Box2.height * Box2.length * Box2.breadth;
cout << "Volume of Box2 : " << volume <<endl;
return 0;
}

```

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

```

Volume of Box1 : 210
Volume of Box2 : 1560

```

It is important to note that private and protected members can not be accessed directly using direct member access operator (.). We will learn how private and protected members can be accessed.

Classes and Objects in Detail

So far, you have got very basic idea about C++ Classes and Objects. There are further interesting concepts related to C++ Classes and Objects which we will discuss in various sub-sections listed below –

C++ Class Member Functions

A member function of a class is a function that has its definition or its prototype within the class definition like any other variable. It operates on any object of the class of which it is a member, and has access to all the members of a class for that object.

Let us take previously defined class to access the members of the class using a member function instead of directly accessing them –

```

class Box {
public:
    double length;    // Length of a box
    double breadth;   // Breadth of a box
    double height;    // Height of a box
    double getVolume(void); // Returns box volume
};

```

Member functions can be defined within the class definition or separately using scope resolution operator: –. Defining a member function within the class definition declares the function inline, even if you do not use the inline specifier. So either you can define Volume() function as below –

```
class Box {
public:
    double length;    // Length of a box
    double breadth;   // Breadth of a box
    double height;    // Height of a box

    double getVolume(void) {
        return length * breadth * height;
    }
};
```

If you like, you can define the same function outside the class using the scope resolution operator (::) as follows –

```
double Box::getVolume(void)
{
    return length * breadth * height;
}
```

Here, only important point is that you would have to use class name just before :: operator. A member function will be called using a dot operator (.) on a object where it will manipulate data related to that object only as follows –

```
Box myBox;        // Create an object
```

```
myBox.getVolume(); // Call member function for the object
```

Let us put above concepts to set and get the value of different class members in a class –

```
#include <iostream>

using namespace std;

class Box {
public:
    double length;    // Length of a box
    double breadth;   // Breadth of a box
    double height;    // Height of a box
```

```

    // Member functions declaration
    double getVolume(void);
    void setLength( double len );
    void setBreadth( double bre );
    void setHeight( double hei );
};

// Member functions definitions
double Box::getVolume(void) {
    return length * breadth * height;
}

void Box::setLength( double len ) {
    length = len;
}
void Box::setBreadth( double bre ) {
    breadth = bre;
}
void Box::setHeight( double hei ) {
    height = hei;
}

// Main function for the program
int main() {
    Box Box1;           // Declare Box1 of type Box
    Box Box2;           // Declare Box2 of type Box
    double volume = 0.0; // Store the volume of a box here

    // box 1 specification
    Box1.setLength(6.0);
    Box1.setBreadth(7.0);
    Box1.setHeight(5.0);

    // box 2 specification
    Box2.setLength(12.0);
    Box2.setBreadth(13.0);
    Box2.setHeight(10.0);

    // volume of box 1
    volume = Box1.getVolume();
    cout << "Volume of Box1 : " << volume << endl;

    // volume of box 2
    volume = Box2.getVolume();
    cout << "Volume of Box2 : " << volume << endl;
    return 0;
}

```

```
}
```

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

```
Volume of Box1 : 210
Volume of Box2 : 1560
```

Class Access Modifiers

Data hiding is one of the important features of Object Oriented Programming which allows preventing the functions of a program to access directly the internal representation of a class type. The access restriction to the class members is specified by the labeled public, private, and protected sections within the class body. The keywords public, private, and protected are called access specifiers.

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.

```
class Base {
    public:
        // public members go here
    protected:

    // protected members go here
    private:
        // private members go here

};
```

The public Members

A public member is accessible from anywhere outside the class but within a program. You can set and get the value of public variables without any member function as shown in the following example –

```
#include <iostream>
Using namespace std;
class Line {
    public:
        double length;
        void setLength( double len );
        double getLength( void );
};

// Member functions definitions
double Line::getLength(void) {
    return length ;
}
```

```

}

void Line::setLength( double len) {
    length = len;
}

// Main function for the program
int main() {
    Line line;

    // set line length
    line.setLength(6.0);
    cout << "Length of line : " << line.getLength() << endl;

    // set line length without member function
    line.length = 10.0; // OK: because length is public
    cout << "Length of line : " << line.length << endl;

    return 0;
}

```

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

```

Length of line : 6
Length of line : 10

```

The private Members

A private member variable or function cannot be accessed, or even viewed from outside the class. Only the class and friend functions can access private members.

By default all the members of a class would be private, for example in the following class width is a private member, which means until you label a member, it will be assumed a private member –

```

class Box {
    double width;

    public:
        double length;
        void setWidth( double wid );
        double getWidth( void );
};

```

Practically, we define data in private section and related functions in public section so that they can be called from outside of the class as shown in the following program.

```

#include <iostream>

using namespace std;

class Box {
public:
    double length;
    void setWidth( double wid );
    double getWidth( void );

private:
    double width;
};

// Member functions definitions
double Box::getWidth(void) {
    return width ;
}

void Box::setWidth( double wid ) {
    width = wid;
}

// Main function for the program
int main() {
    Box box;

    // set box length without member function
    box.length = 10.0; // OK: because length is public
    cout << "Length of box : " << box.length <<endl;

    // set box width without member function
    // box.width = 10.0; // Error: because width is private
    box.setWidth(10.0); // Use member function to set it.
    cout << "Width of box : " << box.getWidth() <<endl;

    return 0;
}

```

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

```

Length of box : 10
Width of box : 10

```

The protected Members

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.

You will learn derived classes and inheritance in next chapter. For now you can check following example where I have derived one child class SmallBox from a parent class Box.

Following example is similar to above example and here width member will be accessible by any member function of its derived class SmallBox.

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

class Box {
    protected:
        double width;
};

class SmallBox:Box { // SmallBox is the derived class.
    public:
        void setSmallWidth( double wid );
        double getSmallWidth( void );
};

// Member functions of child class
double SmallBox::getSmallWidth(void) {
    return width ;
}

void SmallBox::setSmallWidth( double wid ) {
    width = wid;
}

// Main function for the program
int main() {
    SmallBox box;

    // set box width using member function
    box.setSmallWidth(5.0);
    cout << "Width of box : " << box.getSmallWidth() << endl;

    return 0;
}
```

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

Width of box : 5