305 : OBJECT ORIENTED PROGRAMMING

Amit Patel

Vidyabharti Trust College of BCA, Umrakh

Characteristic (Feature) of OOP

- Class
- Object
- Data Encapsulation
- Data Abstraction
- Inheritance
- Polymorphism
- Message Passing
- Data Binding

```
struct stud
{
    int id;
    char name[30];
};
 struct stud s1,s2;
S1.id
```

Class

- A class in C++ is the building block, that leads to Object-Oriented programming.
- Definition: 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.
- Class will not occupy any memory space and hence it is logical representation of data.
- Class is a data type from which programmer create variable called object.

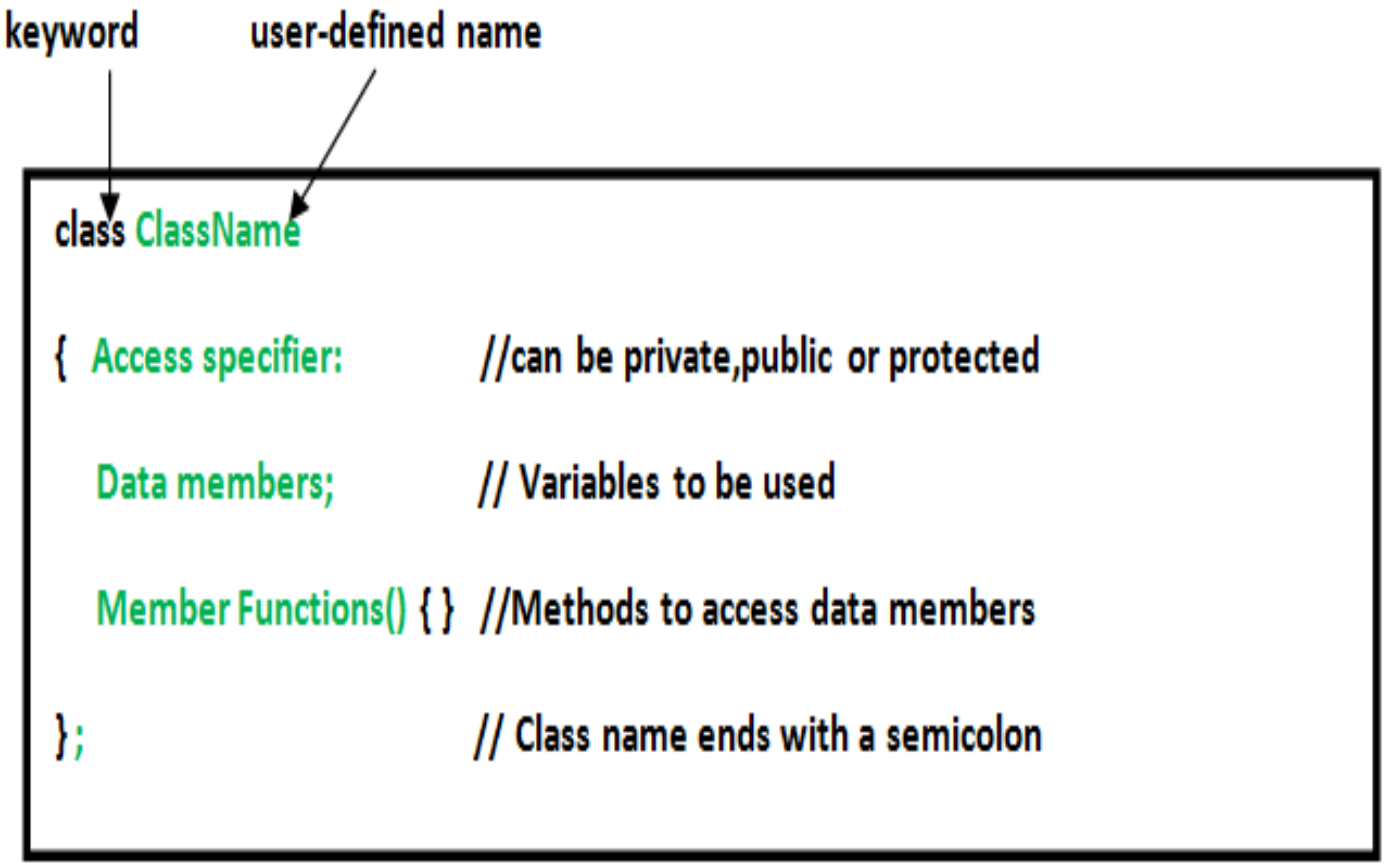
Class - Example

- 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.

Class

- 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 defines the properties and behavior 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 be apply brakes, increase speed etc.

Class - Syntax



Class - Syntax

```
class <classname>
{
    private:
        data member;
        member function;
    public:
        data member;
        member function;
    protected:
        data member;
        member function;
};
```

Class

- “class” keyword is used to create class.
- <classname> is user define name.

Visibility Modifier (Access Specifier)

- It defines the scope of data member and member function.
- There are 3 visibility modifier used in c++
  - public,
  - private and
  - protected
- By default visibility modifier is “private”.

Class

- Class is end with “;”.

Data member

- It is variable declare inside the class.
- Contain the value for specified object.
- For every object Data members have separate allocation.
- Generally data member is declare as **private**.

Member function

- It is function declare inside the class.
- To access data member , we must use member function.
- Generally it is declare with **“public”** visibility modifier.

Class

```
class student
{
    int id;
    char name[30];
   public:
      void getdata();
       void putdata();
};
```


Object

- Objects are the basic run-time entities in an object oriented system.
- They may represent a person, a place or any item that the program has to handle in programming.
- Object has two characteristics: **State** and **behaviours.**
- Class will not occupy any memory space. Hence to work with the data represented by the class you must create a variable for the class, which is called as an object.
- Syntax to create an object of class Employee:

```
student s; // s is object
```


Object

- Objects are the basic run-time entities in an object oriented system.
- They may represent a person, a place or any item that the program has to handle in programming.
- Object has two characteristics: **State** and **behaviours.**
- Class will not occupy any memory space. Hence to work with the data represented by the class you must create a variable for the class, which is called as an object.
- When object is created , memory is allocated.
- Syntax to create an object of class Employee:

```
student s; // s is object

student s1; // s1 is another object
```


Data Encapsulation

- **Definition:** Wrapping up data member and member function together into a single unit (i.e. Class) is called Encapsulation
- It is **implemented using class** in C++ programming.
- Encapsulation means hiding the internal details of an object, i.e. how an object does something.
- Encapsulation is a technique used to protect the information in an object from the other object.
- Encapsulation is like enclosing in a capsule. That is enclosing the related operations and data related to an object into that object.
- Data encapsulation led to the important OOP concept of data hiding.

Data Encapsulation

- Encapsulation prevents clients from seeing its inside view, where the behavior of the abstraction is implemented.
- Encapsulation is like your bag in which you can keep your pen, book etc. It means this is the property of encapsulating members and functions.

Benefits with encapsulation:

- Modularity.
- Information hiding.

Data Encapsulation - Example

- Let's take General example of Mobile Phone and Mobile Phone Manufacturer
- Suppose you are a Mobile Phone Manufacturer and you designed and developed a Mobile Phone design(class), now by using machinery you are manufacturing a Mobile Phone(object) for selling, when you sell your Mobile Phone the user only learn how to use the Mobile Phone but not that how this Mobile Phone works.
- This means that you are creating the class with function and by making object (capsule) of it you are making availability of the functionality of you class by that object and without the interference in the original class.

How to Implement Encapsulation in Programming?

- Encapsulation is combining of the data members and methods(functions) into a single entity, called Class.
- So, Programmatically we implement encapsulation using creating a “class”.

Programming Example

- Here we have two data members id and name, we have declared them as private so that they are not accessible outside the class, this way we are hiding the data.
- The only way to get and set the values of these data members is through the public getter and setter functions.

Programming Example

```
#include<iostream.h>

class student

{

private:

   /* Since we have marked these data members private, any entity outside this class cannot access these data members directly, they have to use getter and  setter functions.    */

    int id;

  char name[30];
```


Programming Example

```
public:
  /* Getter functions to get the value of data members. Since these functions are public, they can be accessed outside the class, thus provide the access to data members through them */
void get( )
 {
    cout<<“Enter id and name:”;
    cin>>id>>name;
 }
```


Programming Example

/* Getter functions to get the value of data members. Since these functions are public, they can be accessed outside the class, thus provide the access to data members through them */

```
  void put( )
  {
      cout<<“Id is :”<<id<<“\nName is :”<<name;
}


};
```


Programming Example

```
int main()
{
student obj;
  clrscr();
  obj.get();
  obj.put();
   return 0;
}
```


Data Abstraction

- Abstraction is the concept of exposing only the required essential characteristics and behavior with respect to a context.
- In object oriented programming language this is implemented automatically while writing the code in the form of class and object.
- Abstraction is one of the feature of Object Oriented Programming, where you show only relevant details to the user and hide irrelevant details.
- In abstraction, we have to focus only on what is to be done instead of how it should be done.
- Abstraction is a thought process; it solves the problem at the design level.

Data Abstraction

- Abstraction is the process of hiding the internal working of a process from its end user.
- In the context of programming, separating the signature of a method from its definition is called abstraction.
- When a programmer wants to perform a task, he looks for a method which can do it for him. If such a method is found, the programmer invokes it without bothering about its internal working.

Data Abstraction: Example

- You can understand with this example, when you send an email to someone you just click send and you get the success message, what actually happens when you click send, how data is transmitted over network to the recipient is hidden from you.
- We can implement Abstraction in C++ using classes. 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 outside world and which is not.

Relationship

Real life analogy to explain Encapsulation, Data Hiding and Abstraction

- A house can be thought of as an application. Drawing room, Bed room, Store room, Kitchen, Bath room etc can be thought of as its classes. According to the functionality, we put different items (data) in different part (classes) of the house such as Bed in bedroom, sofa and TV in drawing room, utensils and stove in kitchen, soap, shampoo, buckets in bathroom etc. This is encapsulation.

Data Abstraction: Example

- Most of the items are accessible to all the members of the house but some valuables such as money and jewelery are kept hidden in the closet i.e. neither all the members of the house nor outsiders have access to them. This is data hiding.

- In our house we use various appliances such as TV, Fridge, Fan, AC etc. Neither we create them nor we have any interest in their internal workings. We simply operate them i.e. their internal working is abstracted from us. This is abstraction.

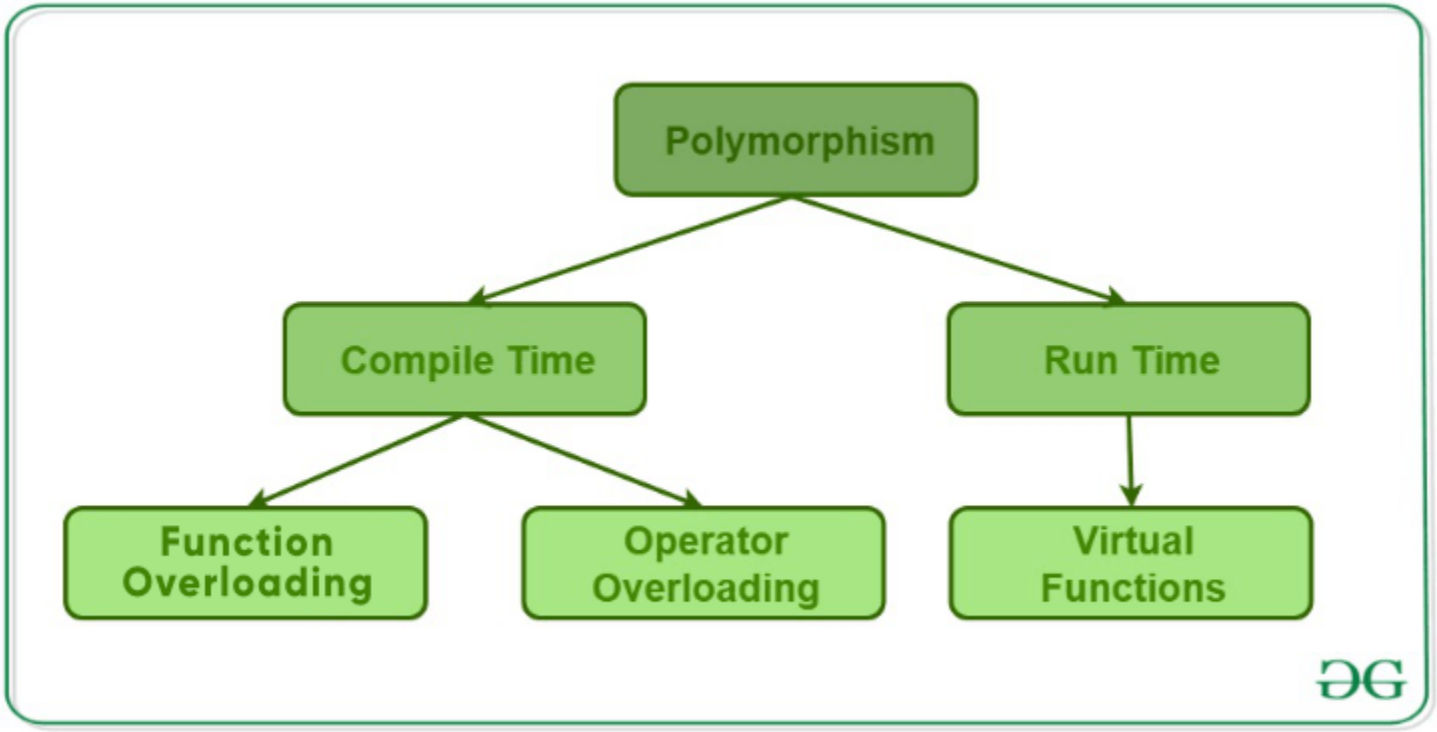
Data Abstraction Vs Encapsulation

- Abstraction focuses on elements that are necessary to build a system whereas, the encapsulation focuses on hiding the complexity of the system.
- The abstraction is performed during the design level of a system. On the other hand, encapsulation is performed the system is being implemented.
- Abstractions main motive is, what is to be done to build a system. Encapsulations main motive is, how it should be done to build a system.
- Abstraction is achieved by encapsulation whereas, the encapsulation is achieved by making the elements of the system private.

Polymorphism

- Polymorphism is derived from 2 greek words: poly and morphs.
- The word "poly" means many and morphs means forms. So polymorphism means many forms.
- The process of representing one Form in multiple forms is known as Polymorphism.
- Here one form represent original form or original method always resides in base class and multiple forms represents overridden method which resides in derived classes.

Polymorphism



Polymorphism : Real Life Example

In Shopping malls behave like Customer

In Bus behave like Passenger

In School behave like Student

At Home behave like Son Sitesbay.com

Polymorphism : Example

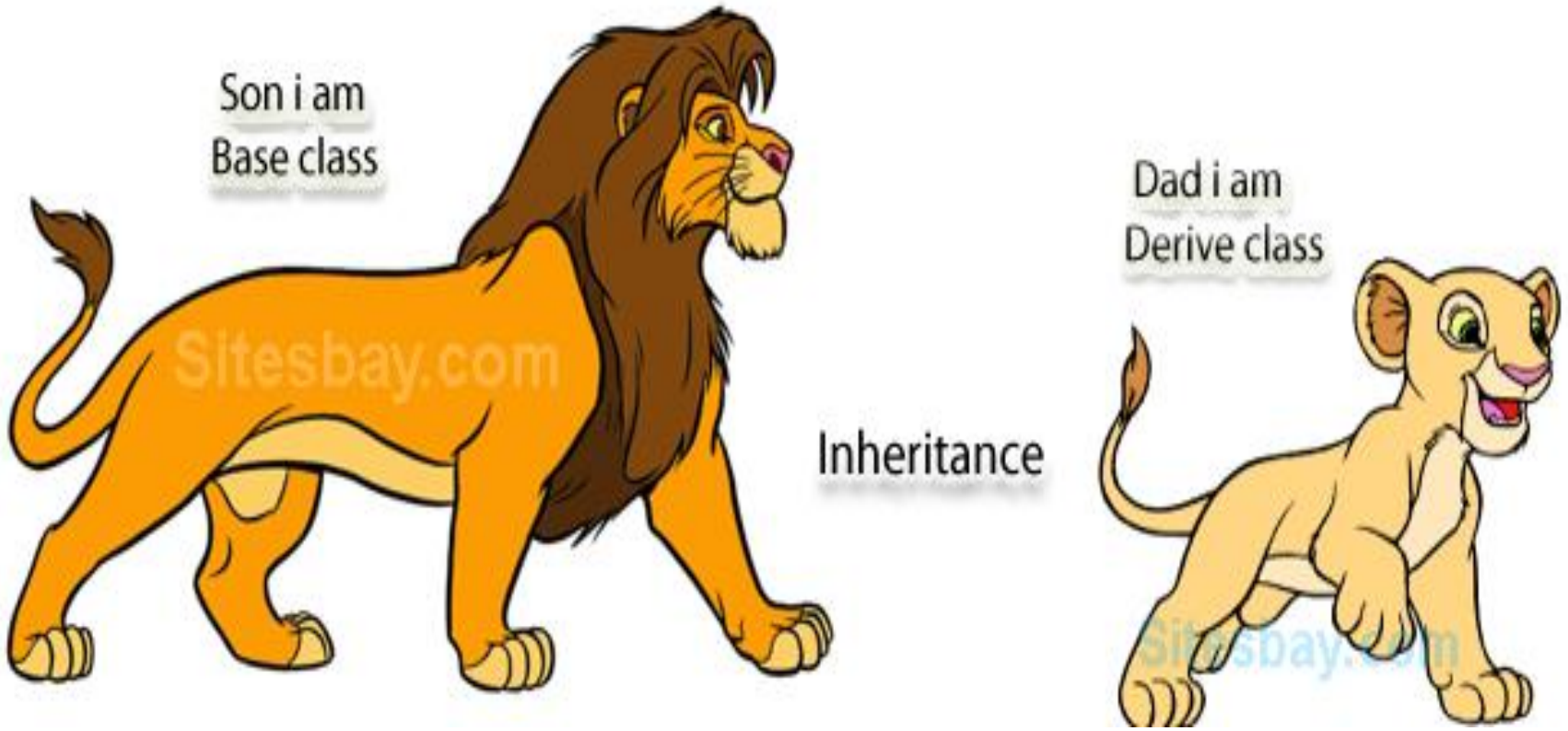
- Suppose if you are in class room that time you behave like a student, when you are in market at that time you behave like a customer, when you at your home at that time you behave like a son or daughter, Here one person have different-different behaviors.

Inheritance

- Inheritance is a process in which one object acquires all the properties and behaviors of its parent object automatically.
- In such way, you can reuse, extend or modify the attributes and behaviors which are defined in other class.
- In C++, the class which inherits the members of another class is called derived class and the class whose members are inherited is called base class.

Inheritance – Real Life Example

- The real life example of inheritance is child and parents, all the properties of father are inherited by his son.



Inheritance – Example

- Lets assume that Human is a class that has properties such as height, weight, colour etc and functionality such as eating(), sleeping(), dreaming(), working() etc.
- Now we want to create Male and Female class, these classes are different but since both Male and Female are humans they share some common properties and behaviours (functionality) so they can inherit those properties and functionality from Human class and rest can be written in their class separately.
- This approach makes us write less code as both the classes inherited several properties and functions from base class thus we didn't need to re-write them.

Data Binding

- Binding refers to the linking of function call to the code to be a executed in response to the call.
- It is associates with Polymorphism and Inheritance.
- There are two types of Binding.
  - 1. Static Binding (Early Binding)
  - 2. Dynamic Binding (Late Binding)

Message Passing

- Message Passing is nothing but sending and receiving of information by the objects same as people exchange information. So this helps in building systems that simulate real life.
- Following are the basic steps in message passing.
  - Creating classes that define objects and its behaviour.
  - Creating objects from class definitions
  - Establishing communication among objects
- In OOPs, Message Passing involves specifying the name of objects, the name of the function, and the information to be sent.

Message Passing

- Objects can communicate with each others by passing message same as people passing message with each other.
- Objects can send or receive message or information.
- For example:

```
student s;

s.get(1,”om”);
```

Here student is object, get is message, 1 and om is information.

Application of C++

- Operating Systems : Be it Microsoft Windows or Mac OSX or Linux - all of them are programmed in C++.
- Browsers : The rendering engines of various web browsers are programmed in C++ simply because if the speed that it offers.
- Libraries: Many high-level libraries use C++ as the core programming language. For instance, several Machine Learning libraries use C++ in the backend because of its speed.
- Graphics : All graphics applications require fast rendering and just like the case of web browsers, here also C++ helps in reducing the latency.

Application of C++

- Banking Applications: Banking applications process millions of transactions on a daily basis and require high concurrency and low latency support. C++ automatically becomes the preferred choice in such applications owing to its speed.
- Cloud/Distributed Systems: Large organizations that develop cloud storage systems and other distributed systems also use C++ because it connects very well with the hardware and is compatible with a lot of machines.

Application of C++

- Databases : Postgres and MySQL - two of the most widely used databases are written in C++ and C, the precursor to C++. These databases are used in almost all of the well-known applications that we all use in our day to day life - Quora, YouTube, etc.
- Embedded Systems: Various embedded systems like medical machines, smartwatches, etc. use C++ as the primary programming language because of the fact that C++ is closer to the hardware level as compared to other high-level programming languages.

Application of C++

- Telephone Switches: Because of the fact that it is one of the fastest programming languages, C++ is widely used in programming telephone switches, routers, and space probes.
- Compilers: The compilers of various programming languages use C and C++ as the backend programming language. This is because of the fact that both C and C++ are relatively lower level languages and are closer to the hardware and therefore are the ideal choice for such compilation systems..

Implementation of Member Function

- Member function can be implemented in
  - Inside the class
  - Outside the class

Inside the class implementation

- When we want to implement function inside the class, not need to use membership label (scope resolution operator and class name).
- We declare and implement function at same place.
- When function is small , one or two liner, its implemented inside the class.

Implementation of Member Function

```
class student
{
      int id;
      char name[30];
 public:
     void get()
     {
          cout<<“Enter student id and name:”;
          cin>>id>>name;
       }
};
```


Implementation of Member Function

Outside the class implementation

- When we want to implement function outside the class, use membership label (combination of scope resolution operator and class name).
- Membership label tells the compiler that function is associated or member of specified class.
- Good practice to implement function outside the class.
- Improve the program readability.

Implementation of Member Function

```
class student
{
      int id;
      char name[30];
 public:
     void get(); //function declaration
};
void student::get()
{
         cout<<“Enter student id and name:”;
          cin>>id>>name;
  }
```


Scope Resolution Operator

- It is symbolize as : :
- There are several use of scope resolution operator in c++.

1. To access a global variable when there is a local variable with same name.
2. To define a function outside a class.
3. To access a class's static variables or function.
4. In case of multiple Inheritance to access override function or variable.

Scope Resolution Operator

- To access a global variable when there is a local variable with same name.

```
#include<iostream>
int x;  // Global x
int main()
{
  int x = 10; // Local x
  cout << "Value of global x is " << ::x;
  cout << "\nValue of local x is " << x;
  return 0;
}
```

Output: Value of global x is 0
Value of local x is 10

Scope Resolution Operator

- To define a function outside a class.

```
#include<iostream>
class A
{
public:
  // Only declaration
   void fun();
};
// Definition outside class using ::
void A::fun()
{
   cout << "fun() called";
}
```


Scope Resolution Operator

```
int main()
{
   A a;
   a.fun();
   return 0;
}
```


Scope Resolution Operator

- To access a class's static variables.

```
class Test
{
    static int x;
public:
    static int y;
    // Local parameter 'x' hides class member
    // 'x', but we can access it using ::
```


Scope Resolution Operator

```
void func(int x)
    {
        // We can access class's static variable even if there is a local
            variable of same name
        cout << "Value of static x is " << Test::x;
        cout << "\nValue of local x is " << x;
    }
};
```


Scope Resolution Operator

```
// In C++, static members must be explicitly defined  like this

int Test::x = 1;

int Test::y = 2;

int main()

{

    Test obj;

    int x = 3 ;

    obj.func(x);

   cout << "\nTest::y = " << Test::y;

    return 0;

}
```


Scope Resolution Operator

Output:

Value of static x is 1

Value of local x is 3

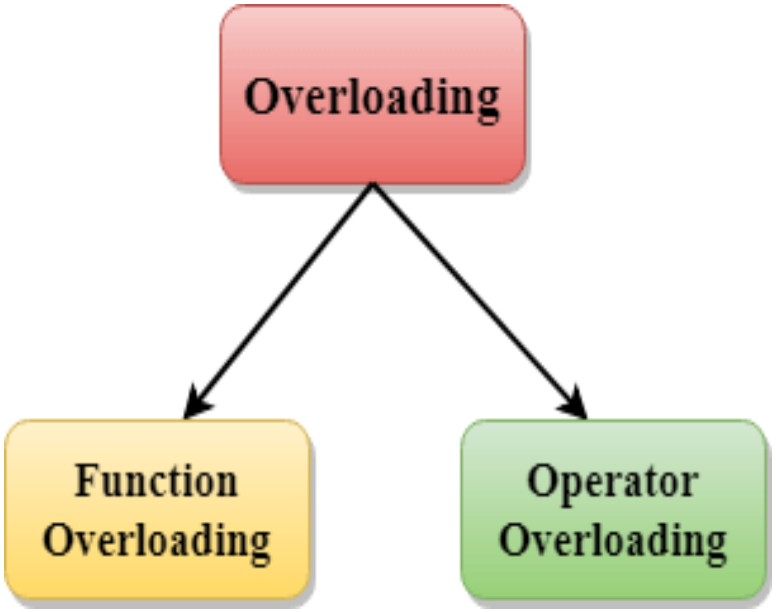
Test::y = 2;

Overloading

- If we create two or more members having the same name but different in number or type of parameter, it is known as C++ overloading.
- In C++, we can overload:
  - Methods (function),
  - constructors, and
  - Operator

Types of Overloading

- It is a part of polymorphism.
- Function and operator overloading is part of compile time polymorphism



Function Overloading

- Function Overloading is defined as the process of having two or more function with the same name, but different in parameters is known as function overloading in C++.
- In function overloading, the function is redefined by using either different types of arguments or a different number of arguments.
- It is only through these differences compiler can differentiate between the functions.
- Advantage of Function overloading is that it increases the readability of the program because you don't need to use different names for the same action.

Function Overloading

Requirements:

- Function names must be same.
- Arguments of same function name must be different
- It cannot differentiate using return type.

Function Overloading

Requirements:

- Function names must be same.
- Arguments of same function name must be different
- It cannot differentiate using return type.

Example: All functions have same name “sum” but all have different parameter so it is function overloading because purpose of both functions are different

- sum(int num1, int num2) // 2 argument with int type
- sum(double num1, double num2) // 2 argument with double

Function Overloading

int sum(int, int)

double sum(int, int)

This is not allowed as the parameter list is same. Even though they have different return types, its not valid.

Function Overloading

```
#include <iostream>

class DemoClass

{

public:

    int demoFunction(int i)

   {

        return i;

    }
```


Function Overloading

```
double demoFunction(double d)
  {
      return d;
   }
};
```


Function Overloading

```
int main(void)
{
    DemoClass obj;
    cout<<obj.demoFunction(100)<<endl;
    cout<<obj.demoFunction(5005.516);
   return 0;
}
```

Output: 100

5005.516

Types of Member function

- Member function is classified in 3 categories
  - Normal Member Function
  - Static Member Function
  - Friend Member Function

Friend Function

- C++ supports the feature of encapsulation in which the data is bundled together with the functions operating on it to form a single unit.
- By doing this C++ ensures that data is accessible only by the functions operating on it and not to anyone outside the class.
- This is one of the distinguishing features of C++ that preserves the data and prevents it from leaking to the outside world.

Friend Function

- A friend function in C++ is a function that is preceded by the keyword “friend”.
- When the function is declared as a friend, then it can access the private and protected data members of the class.
- A friend function is declared inside the class with a friend keyword preceding as shown below.

Friend Function

```
class className
{
   .......
   ......
 friend returnType functionName(arg list);
 };
```

- As shown above, the friend function is declared inside the class whose private and protected data members are to be accessed.

Friend Function

- The function can be defined (implement) anywhere in the code file and we need not use the keyword friend or the scope resolution, operator.

Characteristics

- A friend function can be declared in the private or public section of the class.
- It can be called like a normal function without using the object.
- A friend function is not in the scope of the class, of which it is a friend.

Friend Function

- A friend function is not invoked(call) using the class object as it is not in the scope of the class.
- A friend function cannot access the private and protected data members of the class directly. It needs to make use of a class object and then access the members using the dot operator.
- We have to pass object as a argument in friend function.
- A friend function can be a global function or a member of another class.

Friend Function

```
#include <iostream>
#include <string>
class sample
{
            int length, breadth;
  public:
   void init(int l, int b)
  {
       length=l;
       breadth=b;
 }
  friend void calcArea(sample s); //friend function declaration
};
```


Friend Function

```
//friend function definition : not require to use membership label
void calcArea(sample s)
{
   cout<<"Area = "<<s.length * s.breadth;
 }
int main()
{
      sample s(10,15);
      calcArea(s);
      return 0;
}
```

Output:

Area = 150

Friend Class

- A friend class is a class that can access the private and protected members of a class in which it is declared as friend.
- This is needed when we want to allow a particular class to access the private and protected members of a class.

Friend Class

- A friend class is a class that can access the private and protected members of a class in which it is declared as friend.
- This is needed when we want to allow a particular class to access the private and protected members of a class.

```
class A{
……
friend class B;
};
class B{
……..
};
```


Friend Class

- As depicted above, class B is a friend of class A. So class B can access the private and protected members of class A.
- But this does not mean that class A can access private and protected members of the class B. Note that the friendship is not mutual unless we make it so.
- Similarly, the friendship of the class is not inherited. This means that as class B is a friend of class A, it will not be a friend of the subclasses of class A.

Friend Class

Example:

- In this example we have two classes XYZ and ABC. The XYZ class has two private data members ch and num, this class declares ABC as friend class.
- This means that ABC can access the private members of XYZ, the same has been demonstrated in the example where the function disp() of ABC class accesses the private members num and ch. In this example we are passing object as an argument to the function

Friend Class

```
#include <iostream>

class XYZ

{

private:

   char ch='A';

   int num = 11;

public:

 /* This statement would make class ABC a friend class of XYZ, this means

that ABC can access the private and protected members of XYZ class.    */

   friend class ABC;

};
```


Friend Class

```
class ABC
{
public:
   void disp(XYZ obj){
      cout<<obj.ch<<endl;
      cout<<obj.num<<endl;
   }
};
```


Friend Class

```
int main()
 {
   ABC obj;
   XYZ obj2;
   obj.disp(obj2);
   return 0;
}
```

Output:

```
A
11
```


Static Member Data

- When we declare a normal variable (data member) in a class, different copies of those data members create with the associated objects.
- In some cases when we need a common data member that should be same for all objects, we cannot do this using normal data members. To fulfill such cases, we need static data members.
- It is a variable which is declared with the static keyword, it is also known as class member, thus only single copy of the variable creates for all objects.
- Any changes in the static data member through one member function will reflect in all other object's member functions.

Static Member Data

- Declaration

  static data_type member_name;

- Defining the static data member: It should be defined outside of the class following this syntax:

  data_type class_name :: member_name =value;

Static Member Data : Characteristics

- It is initialized by zero when first object of class is created.
- Only one copy of static data member is created for the entire class and all object share the same copy.
- Its scope is within class but its lifetime is entire program.
- They are used to store the value that are common for all the objects.

Static Member Data : Example

```
#include <iostream.h>
#include <conio.h>
class MyClass
{
        int simple;             // Simple variable
        static int counter;     // Static data member or variable

    public :
        void inc( ) ;    // to increment value of 'counter'
        void print( ) ;    // print value of 'counter'
};
```


Static Member Data : Example

```
int MyClass :: count = 0;    // Definition of static variable

void MyClass :: inc()
{
    counter++ ;
    simple++ ;
}

void MyClass :: print()
{
    count<< "Counter is : " << counter ;
}
```


Static Member Data : Example

```
void main( )
{
   MyClass a1;        // count = 0, simple= garbage
   a1.inc( );         // count = 1, simple = garbage + 1
   a1.print( ) ;

   MyClass a2;        // no new counter, new simple, simple = new garbage
   a2.inc( );      // count  = 2, simple = new garbage + 1
   a2.print( );
   getch() ;
}
```

Static Member Function : Characteristics

- A static function can be access only by other static data member (variables) and function declared in the class.
- A static function is called using class name instead of object name.

Static Member Function : Characteristics

```
#include <iostream.h>
#include <conio.h>
class MyClass
{
        int simple;             // Simple variable
        static int counter;     // Static data member or variable
    public :
        void inc( )    //  to increment value of 'counter'
        {
            counter++ ;
            simple++ ;     // Error ! cannot access non-static member
        }
        void print( ) ;    // print value of 'counter'
};
```


Static Member Function : Characteristics

```
int MyClass :: count = 0;    // Definition of static variable

void MyClass :: print()
{
    count<< "Counter is : " << counter ;
}

int main( )
{
    MyClass a1;        // count = 0, simple= garbage
    MyClass :: inc( );   // called by class name instead of an object.
    a1.print( ) ;
```


Static Member Function : Characteristics

```
 MyClass a2;  // no new counter, new simple, simple = new garbage
    MyClass :: inc( );      // count  = 2, simple = new garbage + 1
    a2.print( );
    getch() ;
}
```