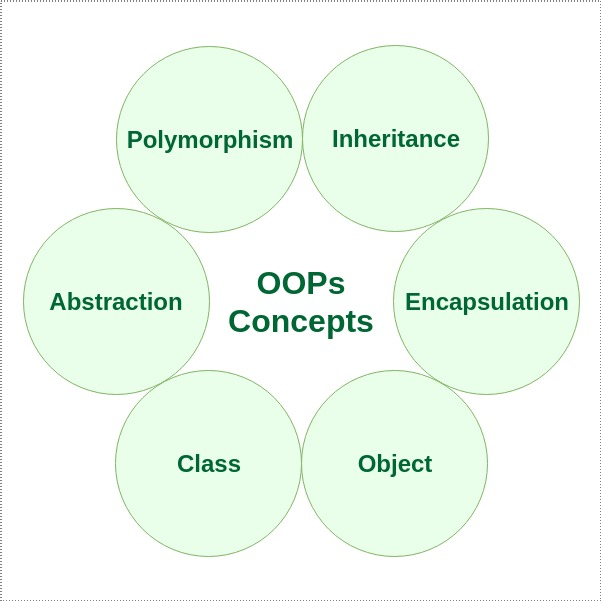
Object-oriented programming – As the name suggests uses [objects](https://www.geeksforgeeks.org/object-oriented-programming-in-cpp/#objects) in programming. Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism, etc in programming. The main aim of OOP 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**](https://www.geeksforgeeks.org/c-classes-and-objects/)**:**

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:**](https://www.geeksforgeeks.org/c-classes-and-objects/)

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.

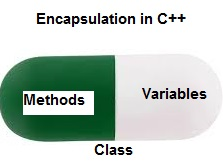
When a program is executed the objects interact by sending messages to one another.

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**](https://www.geeksforgeeks.org/encapsulation-in-c/)**:**

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**](https://www.geeksforgeeks.org/abstraction-in-c/)**:**

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.

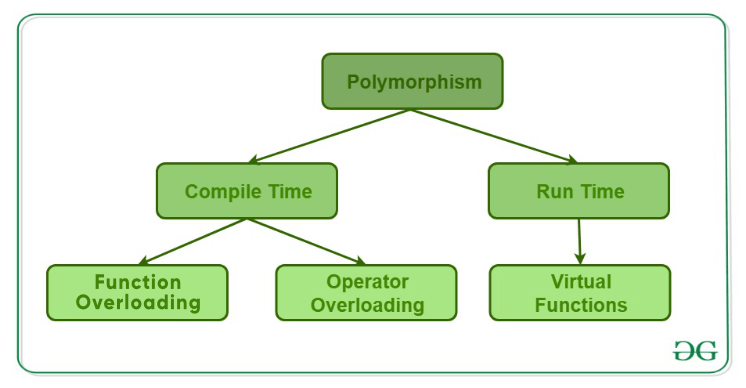
* *Abstraction using Classes*: We can implement Abstraction in C++ using classes. The class helps us to group data members and member functions using available access specifiers. A Class can decide which data member will be visible to the outside world and which is not.
* *Abstraction in Header files*: 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:**](https://www.geeksforgeeks.org/polymorphism-in-c/)

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 posses different behaviour in different situations. This is called polymorphism.

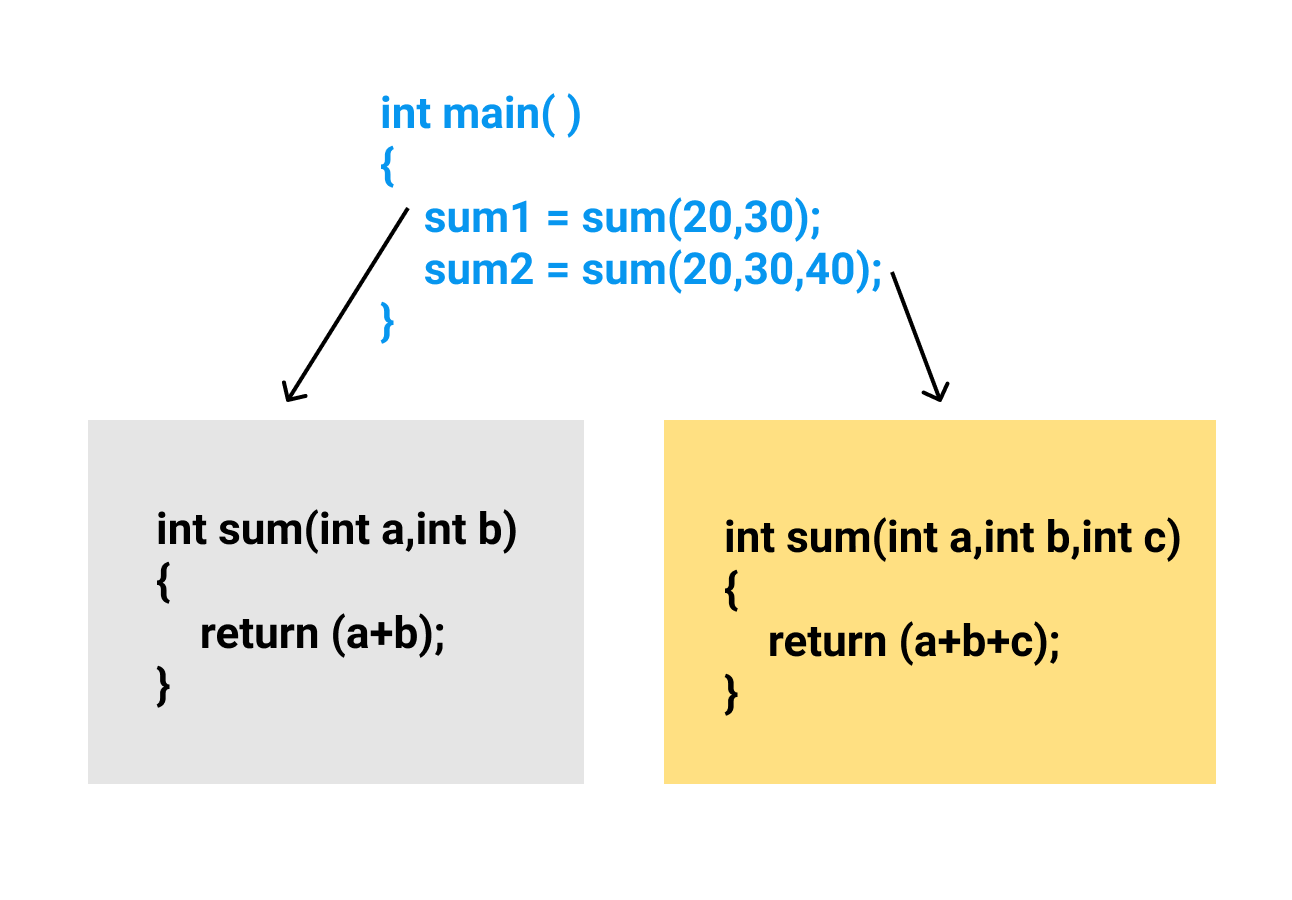
An operation may exhibit different behaviours in different instances. The behaviour depends upon the types of data used in the operation.



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, some times there are 2 integers, some times 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.



**Operator Overloading Example**

// C++ program to demonstrate

// Operator Overloading or

// Compile-Time Polymorphism

#include <iostream>

using namespace std;

class Complex {

private:

int real, imag;

public:

Complex(int r = 0,

int i = 0)

{

real = r;

imag = i;

}

// This is automatically called

// when '+' is used with between

// two Complex objects

Complex operator+(Complex const& obj)

{

Complex res;

res.real = real + obj.real;

res.imag = imag + obj.imag;

return res;

}

void print()

{

cout << real << " + i" <<

imag << endl;

}

};

// Driver code

int main()

{

Complex c1(10, 5), c2(2, 4);

// An example call to "operator+"

Complex c3 = c1 + c2;

c3.print();

}

### **Virtual Function**

A [virtual function](https://www.geeksforgeeks.org/virtual-function-cpp/) is a member function that is declared in the base class using the keyword virtual and is re-defined (Overriden) in the derived class.

#### **Some Key Points About Virtual Functions:**

* Virtual functions are Dynamic in nature.
* They are defined by inserting the keyword “**virtual**” inside a base class and are always declared with a base class and overridden in a child class
* A virtual function is called during Runtime

**Example:**

// C++ Program to demonstrate

// the Virtual Function

#include <iostream>

using namespace std;

// Declaring a Base class

class GFG\_Base {

public:

// virtual function

virtual void display()

{

cout << "Called virtual Base Class function" <<

"\n\n";

}

void print()

{

cout << "Called GFG\_Base print function" <<

"\n\n";

}

};

// Declaring a Child Class

class GFG\_Child : public GFG\_Base {

public:

void display()

{

cout << "Called GFG\_Child Display Function" <<

"\n\n";

}

void print()

{

cout << "Called GFG\_Child print Function" <<

"\n\n";

}

};

// Driver code

int main()

{

// Create a reference of class bird

GFG\_Base\* base;

GFG\_Child child;

base = &child;

// This will call the virtual function

base->GFG\_Base::display();

// this will call the non-virtual function

base->print();

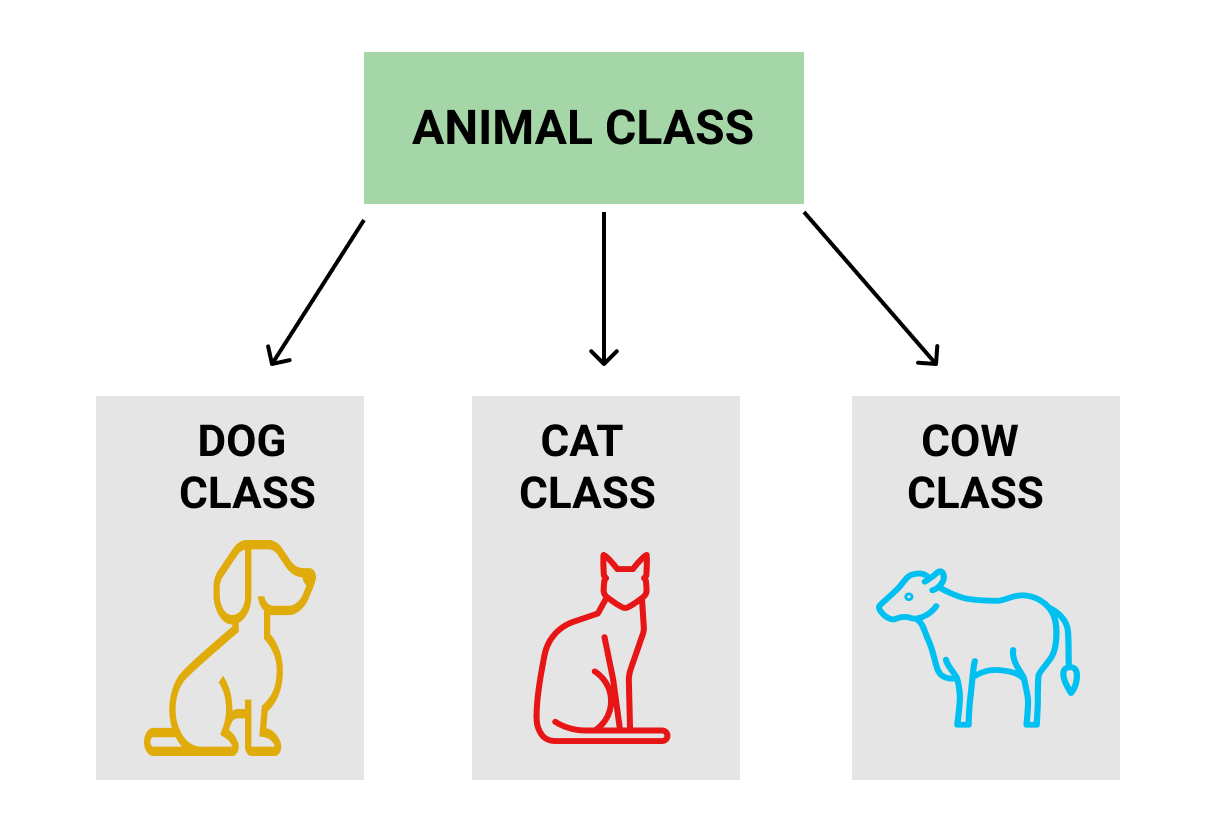
}

[**Inheritance**](https://www.geeksforgeeks.org/inheritance-in-c/)**:**

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.



**Syntax**:

class <derived\_class\_name> : <access-specifier> <base\_class\_name>

{

//body

}

Where

class — keyword to create a new class

derived\_class\_name — name of the new class, which will inherit the base class

access-specifier — either of private, public or protected. If neither is specified, PRIVATE is taken as default

base-class-name — name of the base class

// C++ Implementation to show that a derived class

// doesn’t inherit access to private data members.

// However, it does inherit a full parent object.

class A {

public:

int x;

protected:

int y;

private:

int z;

};

class B : public A {

// x is public

// y is protected

// z is not accessible from B

};

class C : protected A {

// x is protected

// y is protected

// z is not accessible from C

};

class D : private A // 'private' is default for classes

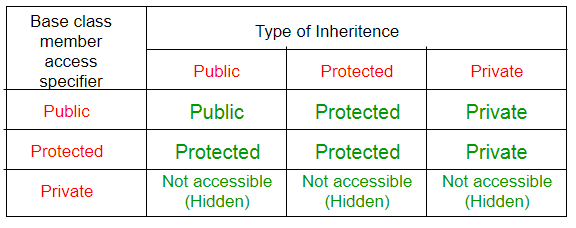
{

// x is private

// y is private

// z is not accessible from D

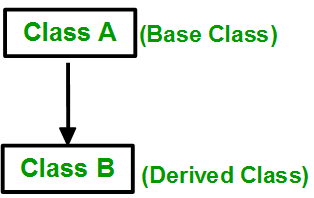
};



### **Types Of Inheritance:-**

1. Single inheritance
2. Multilevel inheritance
3. Multiple inheritance
4. Hierarchical inheritance
5. Hybrid inheritance

**1. Single Inheritance**: In single inheritance, a class is allowed to inherit from only one class. i.e. one subclass is inherited by one base class only.



// C++ program to explain

// Single inheritance

#include<iostream>

using namespace std;

// base class

class Vehicle {

public:

Vehicle()

{

cout << "This is a Vehicle\n";

}

};

// sub class derived from a single base classes

class Car : public Vehicle {

};

// main function

int main()

{

// Creating object of sub class will

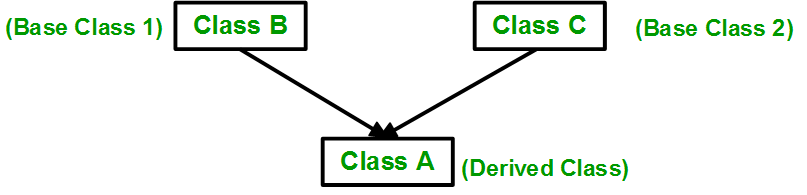
// invoke the constructor of base classes

Car obj;

return 0;

}

**2. Multiple Inheritance:** Multiple Inheritance is a feature of C++ where a class can inherit from more than one class. i.e one **subclass** is inherited from more than one **base class**.



**Syntax**:

class subclass\_name : access\_mode base\_class1, access\_mode base\_class2, ....

{

// body of subclass

};

class B

{

... .. ...

};

class C

{

... .. ...

};

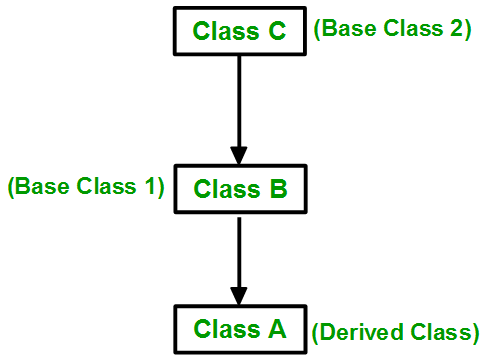
class A: public B, public C

{

... ... ...

};

**3. Multilevel Inheritance**: In this type of inheritance, a derived class is created from another derived class.



Syntax:-

class C

{

... .. ...

};

class B:public C

{

... .. ...

};

class A: public B

{

... ... ...

};

**4. Hierarchical Inheritance**: In this type of inheritance, more than one subclass is inherited from a single base class. i.e. more than one derived class is created from a single base class.



Syntax:-

class A

{

// body of the class A.

}

class B : public A

{

// body of class B.

}

class C : public A

{

// body of class C.

}

class D : public A

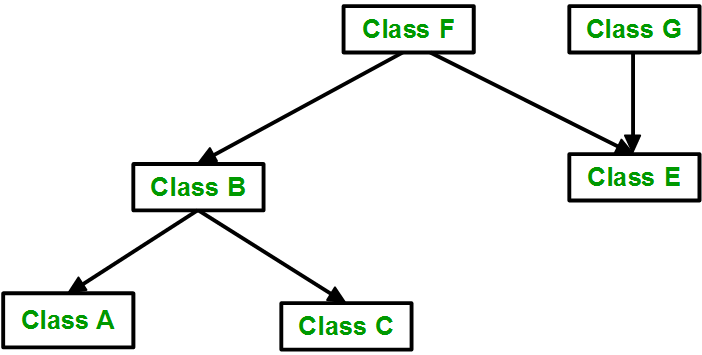
{

// body of class D.

}

**5. Hybrid (Virtual) Inheritance**: Hybrid Inheritance is implemented by combining more than one type of inheritance. For example: Combining Hierarchical inheritance and Multiple Inheritance.

Below image shows the combination of hierarchical and multiple inheritances:



// C++ program for Hybrid Inheritance

#include <iostream>

using namespace std;

// base class

class Vehicle {

public:

Vehicle() { cout << "This is a Vehicle\n"; }

};

// base class

class Fare {

public:

Fare() { cout << "Fare of Vehicle\n"; }

};

// first sub class

class Car : public Vehicle {

};

// second sub class

class Bus : public Vehicle, public Fare {

};

// main function

int main()

{

// Creating object of sub class will

// invoke the constructor of base class.

Bus obj2;

return 0;

}

**Dynamic Binding:**

In dynamic binding, the code to be executed in response to function call is decided at runtime. C++ has [virtual functions](https://www.geeksforgeeks.org/virtual-functions-and-runtime-polymorphism-in-c-set-1-introduction/) 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.

**Encapsulation Example**

// c++ program to explain

// Encapsulation

#include<iostream>

using namespace std;

class Encapsulation

{

private:

// data hidden from outside world

int x;

public:

// function to set value of

// variable x

void set(int a)

{

x =a;

}

// function to return value of

// variable x

int get()

{

return x;

}

};

// main function

int main()

{

Encapsulation obj;

obj.set(5);

cout<<obj.get();

return 0;

}

**Abstraction Example**

// C++ Program to Demonsate the

// wroking of Abstraction

#include <iostream>

using namespace std;

class implementAbstraction {

private:

int a, b;

public:

// method to set values of

// private members

void set(int x, int y)

{

a = x;

b = y;

}

void display()

{

cout << "a = " << a << endl;

cout << "b = " << b << endl;

}

};

int main()

{

implementAbstraction obj;

obj.set(10, 20);

obj.display();

return 0;

}

# **Friend class and function in C++**

**Friend Class** A friend class can access private and protected members of other class in which it is declared as friend. It is sometimes useful to allow a particular class to access private members of other class. For example, a LinkedList class may be allowed to access private members of Node.

A friend class can access both private and protected members of the class in which it has been declared as friend.

class Node {

private:

int key;

Node\* next;

/\* Other members of Node Class \*/

// Now class LinkedList can

// access private members of Node

friend class LinkedList;

};

// Example:

#include<iostream>

using namespace std;

class A

{

int x;

public:

A()

{

x=10;

}

friend class B; //friend class

};

class B

{

public:

void display(A &t)

{

cout<<endl<<"The value of x="<<t.x;

}

};

main()

{

A \_a;

B \_b;

\_b.display(\_a);

return 0;

}

**Friend Function** Like friend class, a friend function can be given a special grant to access private and protected members. A friend function can be:

a) A member of another class

b) A global function

* A friend function is a special function in C++ which in-spite of not being member function of a class has privilege to access private and protected data of a class.
* A friend function is a non member function or ordinary function of a class, which is declared as a friend using the keyword “friend” inside the class. By declaring a function as a friend, all the access permissions are given to the function.
* The keyword “friend” is placed only in the function declaration of the friend function and not in the function definition.
* When friend function is called neither name of object nor dot operator is used. However it may accept the object as argument whose value it want to access.
* Friend function can be declared in any section of the class i.e. public or private or protected.

Syntax:-

Syntax :

class <class\_name>

{

friend <return\_type> <function\_name>(argument/s);

};

// Example: Find the largest of two numbers using Friend Function

#include<iostream>

using namespace std;

class Largest

{

int a,b,m;

public:

void set\_data();

friend void find\_max(Largest);

};

void Largest::set\_data()

{

cout<<"Enter the First No:";

cin>>a;

cout<<"Enter the Second No:";

cin>>b;

}

void find\_max(Largest t)

{

if(t.a>t.b)

t.m=t.a;

else

t.m=t.b;

cout<<"Maximum Number is\t"<<t.m;

}

main()

{

Largest l;

l.set\_data();

find\_max(l);

return 0;

}

### **Merits:-**

* A friend function is able to access members without the need of inheriting the class.
* Friend function acts as a bridge between two classes by accessing their private data.
* It can be used to increase the versatility of overloaded operator.
* It can be declared either in the public or private or protected part of class.

### **Demerits:-**

* Friend functions have access to private members of a class from outside the class which violates the law of the data hiding.
* Friend functions cannot do any run time polymorphism in its members.