C++ Friend function

If a function is defined as a friend function in C++, then the protected and private data of a class can be accessed using the function.

By using the keyword friend compiler knows the given function is a friend function.

For accessing the data, the declaration of a friend function should be done inside the body of a class starting with the keyword friend.

Declaration of friend function in C++

**class** class\_name

{

**friend** data\_type function\_name(argument/s);            // syntax of friend function.

};

In the above declaration, the friend function is preceded by the keyword friend. The function can be defined anywhere in the program like a normal C++ function. The function definition does not use either the keyword **friend or scope resolution operator**.

#include <iostream>

using namespace std;

class Box {

double width;

public:

friend void printWidth( Box box );

void setWidth( double wid );

};

// Member function definition

void Box::setWidth( double wid ) {

width = wid;

}

// Note: printWidth() is not a member function of any class.

void printWidth( Box box ) {

/\* Because printWidth() is a friend of Box, it can

directly access any member of this class \*/

cout << "Width of box : " << box.width <<endl;

}

// Main function for the program

int main() {

Box box;

// set box width without member function

box.setWidth(10.0);

// Use friend function to print the wdith.

printWidth( box );

return 0;

}

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;

}

C++ virtual function

* A C++ virtual function is a member function in the base class that you redefine in a derived class. It is declared using the virtual keyword.
* It is used to tell the compiler to perform dynamic linkage or late binding on the function.
* There is a necessity to use the single pointer to refer to all the objects of the different classes. So, we create the pointer to the base class that refers to all the derived objects. But, when base class pointer contains the address of the derived class object, always executes the base class function. This issue can only be resolved by using the 'virtual' function.
* A 'virtual' is a keyword preceding the normal declaration of a function.
* When the function is made virtual, C++ determines which function is to be invoked at the runtime based on the type of the object pointed by the base class pointer.

Late binding or Dynamic linkage

In late binding function call is resolved during runtime. Therefore compiler determines the type of object at runtime, and then binds the function call.

**Rules of Virtual Function**

* Virtual functions must be members of some class.
* Virtual functions cannot be static members.
* They are accessed through object pointers.
* They can be a friend of another class.
* A virtual function must be defined in the base class, even though it is not used.
* The prototypes of a virtual function of the base class and all the derived classes must be identical. If the two functions with the same name but different prototypes, C++ will consider them as the overloaded functions.
* We cannot have a virtual constructor, but we can have a virtual destructor
* Consider the situation when we don't use the virtual keyword.

#include <iostream>

**using** **namespace** std;

**class** A

{

**int** x=5;

**public**:

**void** display()

    {

        cout << "Value of x is : " << x<<endl;

    }

};

**class** B: **public** A

{

**int** y = 10;

**public**:

**void** display()

    {

        cout << "Value of y is : " <<y<< endl;

    }

};

**int** main()

{

    A \*a;

    B b;

    a = &b;

   a->display();

**return** 0;

}

**Output:**

Value of x is : 5

In the above example, \* a is the base class pointer. The pointer can only access the base class members but not the members of the derived class. Although C++ permits the base pointer to point to any object derived from the base class, it cannot directly access the members of the derived class. Therefore, there is a need for virtual function which allows the base pointer to access the members of the derived class.