Lecture 9

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 (::)**.

Characteristics of a Friend function:

- The function is not in the scope of the class to which it has been declared as a friend.
- It cannot be called using the object as it is not in the scope of that class.
- It can be invoked like a normal function without using the object.
- It cannot access the member names directly and has to use an object name and dot membership operator with the member name.
- It can be declared either in the private or the public part.

C++ friend function Example

Let's see the simple example of C++ friend function used to print the length of a box.

```
#include <iostream>
using namespace std;
class Box
{
    private:
        int length;
    public:
        Box(): length(0) { }
        friend int printLength(Box); //friend function
};
int printLength(Box b)
{
   b.length += 10;
    return b.length;
}
int main()
{
    Box b;
    cout<<"Length of box: "<< printLength(b)<<endl;
    return 0;
}
```

Output:

```
Length of box: 10
```

Let's see a simple example when the function is friendly to two classes.

```
#include <iostream>
using namespace std;
class B;          // forward declarartion.
class A
{
    int x;
    public:
    void setdata(int i)
    {
        x=i;
    }
    friend void min(A,B);         // friend function.
};
class B
{
    int y;
    public:
    void setdata(int i)
    {
        y=i;
    }
    friend void min(A,B);                    // friend function
};
void min(A a,B b)
{
    if(a.x<=b.y)
    cout << a.x << endl;
    else
    cout << b.y << endl;
}
   int main()
{
   A a;
   B b;
   a.setdata(10);
   b.setdata(20);
   min(a,b);
    return 0;
 }
```

Output:

```
10
```

In the above example, min() function is friendly to two classes, i.e., the min() function can access the private members of both the classes A and B.

C++ Friend class

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

Declaration of friend class

```
class class_name
{
      friend class friend_class;// declaring friend class
};

class friend_class
{
};
```

Example:

```
class A
{
      friend class B;// declaring friend class
};


class B
{
};
```

Let's see a simple example of a friend class.

```
#include <iostream>

using namespace std;

class A
{
    int x =5;
    friend class B;           // friend class.
};
class B
{
  public:
    void display(A &a)
    {
        cout<<"value of x is : "<<a.x;
    }
};
int main()
{
    A a;
    B b;
    b.display(a);
    return 0;
}
```

Output:

```
value of x is : 5
```

In the above example, class B is declared as a friend inside the class A. Therefore, B is a friend of class A. Class B can access the private members of class A.