LECTURE-10

Friendly Functions

We know private members cannot be accessed from outside the class. That is a non-member function can't have an access to the private data of a class. However, there could be a case where two classes manager and scientist, have been defined we should like to use a function income tax to operate on the objects of both these classes.

In such situations, C++ allows the common function lo be made friendly with both the classes, there by following the function to have access to the private data of these classes. Such a function need not be a member of any of these classes. To make an outside function "**friendly**" to a class, we have to simply declare this function as a friend of the classes as shown below:

```
class ABC
{
        ---------
        ---------
public:
        --------
        ----------
friend void xyz(void);
};
```

The function declaration should be preceded by the keyword friend, The function is defined else where in the program like a normal C++ function. The function definition does not use their the keyword friend or the scope operator:: . The functions that are declared with the keyword friend are known as friend functions. A function can be declared as a friend in any no of classes. A friend function, as though not a member function, has full access rights to the private members of the class.

A friend function processes certain special characteristics:

- It is not in the scope of the class to which it has been declared as friend.
- Since it is not in the scope of the class, it cannot be called using the object of that class. It can be invoked like a member function without the help of any object.
- Unlike member functions.

```
#include<iostream>
using namespace std;
class AB
{
	int x=77;
	friend void print(AB obj);
};
void print(AB obj)
{
	cout<<obj.x;
}
main()
{
	AB obj;
	print(obj);
}
```

The use of friendly functions in Object-Oriented Programming (OOP) offers several advantages in specific scenarios:

- Enhanced Encapsulation: Friendly functions provide controlled access to private and protected members of a class, allowing external functions or other classes to work with these members without violating encapsulation principles. This access is restricted only to specific functions or classes declared as friends, maintaining data integrity.
- Flexibility and Utility: They enable certain operations that are closely related to a class but are not part of it to directly access private data. This can be beneficial for streamlining specific algorithms.

- Improved Efficiency: In certain cases, accessing private data through friend functions can be more efficient than using public member functions or getter/setter methods. This efficiency gain comes from avoiding the overhead associated with function calls.
- Facilitates Inter-Class Communication: Friendly functions can promote interaction between different classes, especially in cases where one class needs to access private members of another class without exposing them to the public interface.

```
#include<iostream>
using namespace std;
class sample
{
	int a;
	int b;
	public:
	void setvalue( )
	{
		a=25;
		b=40;
	}
	friend float mean(sample s);
};
float mean (sample s)
{
	return (float(s.a+s.b)/2.0);
}

main()
{
	sample x;
	x.setvalue();
	cout<<"mean value="<<mean(x)<<endl;
}
```

A function friendly to two classes

```
#include<iostream>
using namespace std;
class abc;
class xyz
{
       int x;
       public:
       void setvalue(int I) { x= I; }
       friend void max (xyz,abc);
};
class abc
{
       int a;
       public:
       void setvalue( int i) {a=i; }
       friend void max(xyz,abc);
};
void max(xyz m, abc n)
{
       if(m.x >= n.a)
       cout<<m.x;
       else
       cout<< n.a;
}
main( )
{
       abc j;
       j.setvalue(100);
       xyz s;
       s.setvalue(201);
       max(s, j );
}
```

H/W: Swapping private data of classes.

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

Write C++ Program to demonstrate the functioning of a friend class.

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

class GFG {
private:
	int private_variable;

protected:
	int protected_variable;

public:
	GFG()
	{
		private_variable = 10;
		protected_variable = 99;
	}

	// friend class declaration
	friend class F;
};
class F {
public:
	void display(GFG& t)
	{
		cout << "The value of Private Variable = "
			<< t.private_variable << endl;
		cout << "The value of Protected Variable = "
			<< t.protected_variable;
	}
};
main()
{
	GFG g;
	F fri;
	fri.display(g);
}
```

```
#include <iostream>
using namespace std;
class Box {
   double width;
   public:
      friend void printWidth( Box box );
      void setWidth( double wid );
};
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() {
   Box box;
   box.setWidth(10.0);
   printWidth( box );
}
```

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)
    std::cout << a.x << std::endl;
    else
    std::cout << b.y << std::endl;
}
main()
{
   A a;
   B b;
   a.setdata(10);
   b.setdata(20);
   min(a,b);
 }
```

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;
}
```