Assignment

Topic: Friend functions

Muhammad Ehsanul Kader | S201805067 | L-1/T-2 | CSE (B)

Date of Submission: 09/09/2020

Friend Functions

What are friend functions?

In C++ private members of a class are not accessible outside the class. But there are cases where we might want a function to have access to the private members of a class without the function being a member of that class. To solve this , C++ introduced friend functions. Friend functions have the right to access all private and protected members of the classes for which they are friend. A friend function can be a global function that is not part of any particular class or a member function of another class.

Syntax of friend functions

Friend functions are defined in the same way as regular global functions or member functions. However, inside the class declaration for which it will be a friend , its prototype is also included, prefaced by the keyword **friend.** Here are two snippets of code that illustrate how to declare a friend function inside a class named myClass:

Snippet 1:

```
//declaring a global function as a friend function of myClass
friend int isEven(myClass obj);
```

Snippet 2:

```
//declaring a member function of yourClass as a friend function of myClass
friend int yourClass::isEven(myClass obj);
```

In the snippet 2, scope resolution operator(::) is used to point that friend function isEven is a member function of class yourClass.

In which cases should we use friend functions?

1. Permitting one function to have access to the private member of other classes it is not member of.
2. Friend functions are used in operator overloading. In some cases regarding binary operators and built-in types, using friend functions is the only way to overload operators.
3. Friend functions are also useful in the creation of certain types of I/O functions.

Examples of friend functions

In the previous section we have seen the cases where we should use friend functions. Now, in this section we will see some code snippets to better understand how friend functions help us in the first two cases.

Example 1:

```
class Student{
	int secret_code;
	int encoded_id;
public:
	Student(int code, int id){
		secret_code=code;
		encoded_id=secret_code*id;
	}
	friend int Id_decoder(Student student);
};

int Id_decoder(Student student){
	return student.encoded_id/student.secret_code;
}

int main()
{
	Student ehsan(132,1805067);
	cout<<"Student's id is "<<Id_decoder(ehsan);
	return 0;
}
```

Output:

1805067

Exaplanation: Friend function *Id_decoder* has permission to access the private members of student class and it returns the original id without breaching security of the student.

Example 2:

```
class Coord{

	int x,y;
public:
	Coord(){
		x=0; y=0;
	}
	Coord(int i, int j){
		x=i; y=j;
	}
	void get_xy(int &i, int &j){
		i=x; j=y;
	}
	friend Coord operator+(int n,Coord obj);
};

Coord operator+(int n,Coord obj){
	Coord temp;
	temp.x=obj.x+n;
	temp.y=obj.y+n;
	return temp;
}

int main()
{
	Coord ob1(12,20);
	int x,y;
	ob1=ob1+4;
	ob1.get_xy(x,y);
	cout<<"x: "<<x<<" y: "<<y<<"\n";
	return 0;
}
```

Output:

x: 16 y: 24

Explanation: Using member function we can only overload binary operators involving built-in types only if left operand is an object and right operand is a built-in-type.Here, friend operator function allowed built-in type to be on the left side and object to be on the right side of the operator.