Lab 6: Understanding and Implementation of Friend Function and Friend Class in C++

Friend Function

Data hiding is a fundamental concept of object-oriented programming. It restricts the access of private members from outside of the class.

Similarly, protected members can only be accessed by derived classes and are inaccessible from outside. For example,

```
class MyClass {
   private:
      int member1;
}

int main() {
   MyClass obj;

   // Error! Cannot access private members from here.
   obj.member1 = 5;
}
```

However, there is a feature in C++ called friend functions that break this rule and allow us to access member functions from outside the class.

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 className {
```

```
    ... .. ...
    friend returnType functionName(arguments);
    ... .. ...
}
```

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.**

C++ friend function Example

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

```
class Box {
private:
   int length;
public:
   Box(int l) {
      length = l;}
   // Declaration only (not definition)
   friend void showLength(Box b);
};
// Definition outside the class
void showLength(Box b) {
   cout << "Length of box: " << b.length << endl;}

int main() {
   Box box1(15);
   showLength(box1); // friend function accessing private data
   return 0;
}
```

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

```
#include <iostream>
```

```
using namespace std;
class B;   // Forward declaration of class B (so class A can refer to it)
    // Definition of class A
    class A
    {
       int x;   // private data member
    public:
       void setdata(int i)  // member function to set value of x
       {
          x = i;  }
       friend void min(A, B);   // Declaration of friend function
    };

    // Definition of class B
    class B
    {
       int y;   // private data member
    public:
       void setdata(int i)  // member function to set value of y
       {
          y = i;
       }
       friend void min(A, B);   // Same friend function declared here too
    };
    // Definition of friend function (can access private data of both A and B)
    void min(A a, B b)
    {
       if(a.x <= b.y)       // Accessing private members directly
          cout << a.x << endl;
       else
          cout << b.y << endl;
    }

    int main()
    {
       A a;
       B b;

       a.setdata(10);   // Set x = 10
       b.setdata(20);   // Set y = 20

       min(a, b);       // Call friend function to print smaller value

       return 0;
    }
```

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.

In C++, a friend class allows one Class to access another class's private and protected members. This means that the friend class can access the private and protected members of the Class, and it is declared as a friend, just like a member function of that Class.

Syntax

```
class ClassName1 {
    // Class definition

    friend class ClassName2; // Declaration of friend class
};

class ClassName2 {
    // Class definition
};
```

In this syntax:

- friend class **ClassName2**; declares **ClassName2** as a friend class of **ClassName1**.
- This declaration allows **ClassName2** to access the private and protected members of **ClassName1**.
- After the friend keyword, **ClassName2** is specified, indicating that **ClassName2** has access to **ClassName1**'s private and protected members.

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;
}
```

Tasks:

Task 1

Define two classes, Rectangle and Circle, each having a private data member representing an area. Create a friend function that compares the area of a Rectangle object with that of a Circle object and outputs which area is larger.

Task2

Write a C++ program with two classes, Engine and Car. Make Engine a friend class of Car so that it can access and modify the private data members of Car. Implement a function in Engine that increases the horsepower of Car and demonstrates the use of a friend class to access and modify Car's private data.

Task 3

Friend Function for Increment

Task 4

Create a class Matrix and declare and initialize two 3by3 2D arrays .both arrays must be private data members of classs Matrix. Create a firend class of Matrix named modify and add the both arrays of Matrixs class inside the modify class and then transpose the resultant array.

Lab #06 Marks distribution

	ER1	**ER6**	**ER8**
Task	3 points	3 points	4 points

Lab #6 Rubric Evaluation Guideline:

#	Qualities & Criteria	0 < Poor <=0.5	0.5 < Satisfactory <= 1.5	1.5 < Excellent <=2
ER1	**Task Completion**	Minimal or no program functionality was achieved.	Some tasks were completed, but the program has errors or incomplete functionalities.	All tasks were completed, and the program runs without errors.
#	**Qualities & Criteria**	**0 < Poor <=0.5**	**0.5 < Satisfactory <= 1.5**	**1.5 < Excellent <=2**
ER6	Program Output	Output is inaccurate or poorly presented.	Output is mostly accurate but may lack labels, captions, or formatting.	Output is clear, accurate, and well presented with labels, captions, and proper formatting.
#	**Qualities & Criteria**	**0 < Poor <= 1.5**	**1.5< Satisfactory <= 3**	**3< Excellent <= 4**
ER8	Question & Answer	Answers some questions but not confidently or based on lab task knowledge.	Answers most questions confidently and based on lab task knowledge.	Answers all questions confidently and demonstrates a deep understanding of the given lab task.