|  |  |  |
| --- | --- | --- |
| **Rubrics for Object Oriented Programming Lab** | | |
| **Lab #:** | **09** | |
| **Lab Title:** | **Friend Function and Classes** | |
| **Submitted by:** | | |
| **Name** | | **Registration #** |
| **AMMAR**  **MUHAMMAD KALEEM ULLAH** | | **FA19-BCE-001**  **FA19-BCE-007** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Rubrics name & number** | | | **Marks** | | |
| **In-Lab** | | **Post-Lab** |
| **Engineering Knowledge** | ***R2: Use of Engineering Knowledge and follow Experiment Procedures:***  *Ability to follow experimental procedures, control variables, and record procedural steps on lab report.* | |  | | |
| **Problem Analysis** | | ***R5: Data/Evidence Measurements:***  *Ability to record raw data / evidence.* | |  | |
| **Design** | | ***R8: Best Coding Standards:***  *Ability to follow the coding standards and programming practices.* | |  | |
| **Modern Tools Usage** | | ***R9: Understand Tools:*** *Ability to describe and explain the principles behind and applicability of engineering tools.* | |  | |
| **Individual and Teamwork** | | ***R12: Individual Work Contributions:*** *Ability to carry out individual responsibilities.* | |  | |
| ***R13: Management of Team Work:***  *Ability to appreciate, understand and work with multidisciplinary team members.* | |  | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Rubrics #** | R2 | R5 | R8 | R9 | R12 | R13 |
| **In –Lab** |  |  |  |  |  |  |
| **Post- Lab** |  |  |  |  |  |  |

**Lab#09**

**Friend Function and Classes**

1. **Objectives:**

Objective of this lab is to get familiar with:

* 1. The concept of friend functions.
  2. The concept of friend classes.
  3. How to implement the both concepts in object-oriented programming.

1. **Introduction:**
   1. **Friend Function:**

The concepts of encapsulation and data hiding dictate that nonmember functions should not be able to access an object’s private or protected data. The policy is, if you’re not a member, you can’t get in. However, there are situations where such rigid discrimination leads to considerable inconvenience. So, there is mechanism built in C++ programming to access private or protected data from non-member functions.

This is done using a friend function or/and a friend class. The member functions of a class can all be made friends at the same time when you make the

entire class a friend.

* 1. **Access specification and syntax:**

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

The complier knows a given function is a friend function by the use of the **keyword friend.**

For accessing the data, the **declaration** of a friend function should be made inside the body of the class (can be anywhere inside class either in private or public section).

A friend function of a class is **defined** outside that class' scope but it has the right to access all private and protected members of the class. Even though the prototypes for friend functions appear in the class definition, friends are not member functions.

1. **In-Lab Tasks:**
   1. **Task#01:**

Write a program to convert a decimal number into its binary equivalent with a friend function.

**Solution:**

* **Code:**

#include<iostream>

using namespace std;

class decToBin

{

private:

int n;

public:

decToBin()

{

cout<<"\nEnter Decimal Number :";

cin>>n;

}

friend void decimalToBinary(decToBin dtb);

};

void decimalToBinary(decToBin dtb)

{

int i=1,j=dtb.n,b=0;

for(j=dtb.n;j>0;j=j/2)

{

b=b+(dtb.n%2)\*i;

i=i\*10;

dtb.n=dtb.n/2;

}

cout<<"binary number = "<<b<<endl;

}

int main()

{

cout<<"Hello Decimal To Binary ! ";

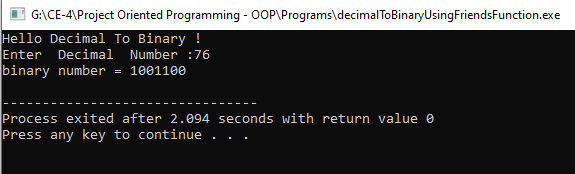
decToBin A;

decimalToBinary(A);

return 0;

}

* **Output:**

****

* 1. **Task#02:**

Make a class distance with data members feet and inches. Now, suppose we want a function that will square (multiply by itself) an object of the Distance class and return the result in square feet, as a type float. Using the concept of friend function implement this Distance class.

**Solution:**

* **Code:**

#include<iostream>

using namespace std;

class Distance

{

private:

float feet,inches;

public:

Distance(float f=0,float i=0)

{

feet=f;

inches=i;

}

void getData()

{

cout<<"\nEnter the feet : ";cin>>feet;

cout<<"\nEnter the Inches : ";cin>>inches;

}

friend float square(Distance d);

void show()const

{

cout<<"\nDistance : "<<feet<<"\'"<<inches<<"\"";

}

};

float square(Distance d)

{

float fsquare;

d.inches=d.inches/12;

fsquare=(d.feet+d.inches)\*(d.feet+d.inches);

return fsquare;

}

int main()

{

Distance d1(5,10);

Distance d2;

d1.show();

cout<<"\nSquare Feets : "<<square(d1);

d2.getData();

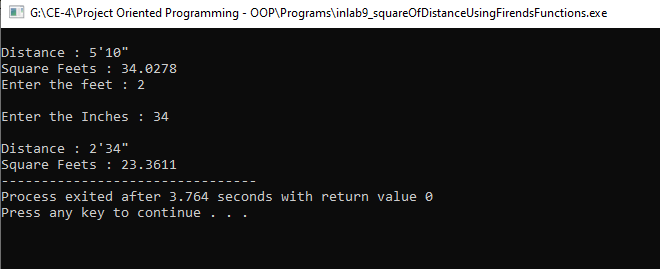
d2.show();

cout<<"\nSquare Feets : "<<square(d2);

return 0;

}

* **Output:**

****

* 1. **Task#03:**

Write a program to convert temperature from Celsius to Fahrenheit scale by creating two different classes for each temperature scale and making both classes friend of each other.

**Solution:**

* **Code:**

#include<iostream>

using namespace std;

class Fahrenheit;

class Celsius

{

private:

float cel,fahr;

public:

Celsius(int c=0,int f=0)

{

cel=c;

fahr=f;

}

friend class Fahrenheit;

};

class Fahrenheit

{

public:

void converter(Celsius c1)

{

c1.fahr = (1.8 \* c1.cel) + 32.0;

cout << "\nTemperature in degree Fahrenheit: " << c1.fahr << " F" << endl;

}

};

int main()

{

Fahrenheit f;

Fahrenheit f1;

Celsius c1;

Celsius c2(34,23);

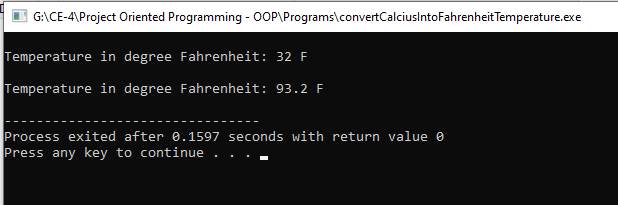
f.converter(c1);

f1.converter(c2);

return 0;

}

* **Output:**

****

1. **Post-Lab Tasks:**
   1. **Task#01:**

In the Distance class, implemented in the in lab task, create an overloaded \* operator so that two distances can be multiplied together. Make it a friend function so that you can use such expressions as Wdist1 = 7.5 \* dist2;

You’ll need a one-argument constructor to convert floating-point values into Distance values. Write a main() program to test this operator in several ways.

**Solution:**

* **Code:**

#include <iostream>

#include <conio.h>

#include<cmath>

using namespace std;

class Distance

{

int feet;

int inches;

public:

Distance() //constructor1

{

feet=0;

inches=0;

}

Distance(float f) //constructor1

{

feet = floor(f);

inches = (f-(float)feet)\*10;

}

Distance(int ft,int inch) //constructor2

{

feet=ft;

inches=inch;

}

void getdata()

{

cout<<"Enter Feet and inches respectively: ";

cin>>feet>>inches;

}

void display()

{

cout<<"Feet : "<<feet<<endl<<"Inches :"<<inches;

}

//Operator declaration in Class

//friend function

friend Distance operator \*(Distance &d, Distance &ob); //ob3= ob1 \* ob2

friend Distance operator \*(float d, Distance &ob); //ob3= floating value \* ob1

};

//operator overloading

Distance operator \*(Distance &d, Distance &ob) //ob1= 2 \* ob2

{

Distance temp;

int i;

temp.feet = d.feet \* ob.feet;

temp.inches = d.inches \* ob.inches;

i = temp.inches/12;

temp.feet = temp.feet + i;

temp.inches = temp.inches-(i\*12);

return(temp);

}

//operator overloading

Distance operator \*(float d, Distance &ob) //ob3= floating value \* ob1

{

Distance temp;

int i;

temp.feet = floor(d) \* ob.feet;

temp.inches = (d-(float)floor(d))\*10 \* ob.inches;

i = temp.inches/12;

temp.feet = temp.feet + i;

temp.inches = temp.inches-(i\*12);

return(temp);

}

//Driver code

int main()

{

Distance ob1, ob2, ob3, ob4;

//float f= 5.8;

ob1 = Distance(5.8);

ob2 = Distance(5,2);

ob3 = ob1\*ob2;

//multiplying Distance with floating value

ob4 = (5.8)\*ob2;

ob3.display();

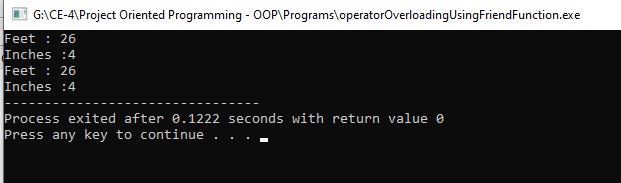
cout<<endl;

ob4.display();

return 0;

}

* **Output:**

****

1. **Conclusion:**

After Completing this lab, we are able to:

* Access private or protected data from non-member functions i.e., friend functions.
* Make a class to access private and protected members of another class using the concept of friend classes.
* Additionally, I have learned about floor() function which returns the largest possible integer value which is less than or equal to the given argument.