

Object Oriented Programming

Week 9

Friend Functions

- A **friend function** of a class is defined outside that class's scope, yet has the right to access the non-public (and public) members of the class. **friend function can be**
- Standalone functions,
- entire classes or
- member functions of other classes may be declared to be friends of another class.

Friend Functions

- The functions which are not member functions of the class yet they can access all private members of the class are called friend functions.
- 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 compiler knows a given function is a friend function by the use of the keyword **friend**.

Example

Create a function AssignCourse that takes all courses and

assigns one course per call to the current faculty object based on the following criteria:

- a. If the calling object is from “Computer Science” department, then assign the available course with course code starting with “C”.
- b. If the calling object is from “Management Science” department, then assign the available course with course code starting with “M”.
- c. If the calling object is from “Electrical Engineering” department, then assign the available course with course code starting with “E”.
- d. While assigning courses to the faculty, do invoke a warning message if the total assigned credit hours exceed maximum 12 credit hours.

Also create the Salary function calculates and prints the salary on the following criteria: If the faculty’s working hours are equal to 36 display the current salary. If the faculty’s working hours are more than 36 then add 1000 Rs for each extra hour and display the updated salary.

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;  
}
```

Friend Function

- There is a feature in C++ called **friend functions** that break this rule and allow us to access member functions from outside the class.
- A friend function can access the private and protected data of a class. We declare a friend function using the friend keyword inside the body of the class.

```
class className {  
    ... ..  
    friend returnType functionName(arguments);  
    ... ..  
}
```

Working of friend Function-example

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

class Distance {
    private:
        int meter;

        // friend function
        friend int addFive(Distance);

    public:
        Distance() : meter(0) {}

};
```

```
// friend function definition
int addFive(Distance d) {

    //accessing private members from
    the friend function
    d.meter += 5;
    return d.meter;
}
```

```
int main() {
    Distance D;
    cout << "Distance: " <<
    addFive(D);
    return 0;
}
```

Friend Function cont'd

- Friend function must be declared with **friend** keyword.
- Friend function must be declare in all the classes from which we need to access private or protected members.
- Friend function will be defined outside the class without specifying the class name.
- Friend function will be invoked like normal function, without any object.

Function Class

- A **friend class** is a class that can access the private and protected members of a class in which it is declared as **friend**. This is needed when we want to allow a particular class to access the private and protected members of a class.
- For example: we have two classes XYZ and ABC. The XYZ class has two private data members ch and num, this class declares ABC as friend class. This means that ABC can access the private members of XYZ,

```
friend class ABC;
```

Working of friend Class-example

```
#include <iostream>
using namespace std;
class XYZ {
private:
    char ch='A';
    int num = 11;
public:
    friend class ABC;
};
```

```
class ABC {
public:
    void disp(XYZ obj){
        cout<<obj.ch<<endl;
        cout<<obj.num<<endl;
    }
};
```

```
int main() {
    ABC obj;
    XYZ obj2;
    obj.disp(obj2);
    return 0;
}
```

Are friend functions against the concept of Object Oriented Programming?

- One of the important concepts of OOP is data hiding, i.e.,
- a nonmember function cannot access an object's private or protected data.
- But, sometimes this restriction may force programmer to write long and complex codes. 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.
- It can be said that friend functions are against the principle of object oriented programming because they violate the principle of encapsulation which clearly says that each object methods and functions should be encapsulated in it. But there we are making our private member accessible to other outside functions.

Consider the following class

```
class X {  
private:  
int a, b;  
public:  
void MemberFunction (); }
```

Suppose we have a global function DoSomething that need to access the private members of class X, when we will try to access them compiler will generate error as outside world cannot access private members of a class except its member functions.

Friend Functions

```
void DoSomething(X obj) {  
obj.a = 3; //Error  
obj.b = 4; //Error }
```

- In order to access the member variables of the class, we must make function friend of that class

Friend Functions

```
class X {  
private:  
int a, b;  
public:  
friend void DoSomething(X obj); }
```

Now the function DoSomething can access data members of

```
class X  
void DoSomething(X obj) {  
obj.a = 3;  
obj.b = 4; }
```

Friend Functions

- Prototypes of friend functions appear in the class definition. But friend functions are NOT member functions.
- Friend functions can be placed anywhere in the class without any effect Access specifiers don't affect friend functions or Classes.

Friend Functions

```
class X{  
...  
private:  
friend void DoSomething(X); public: friend void DoAnything(X);  
...  
};
```

While the definition of the friend function is:

```
void DoSomething(X obj)  
{ obj.a = 3; // No Error obj.b = 4; // No Error  
...  
}
```

friend keyword is not given in definition.

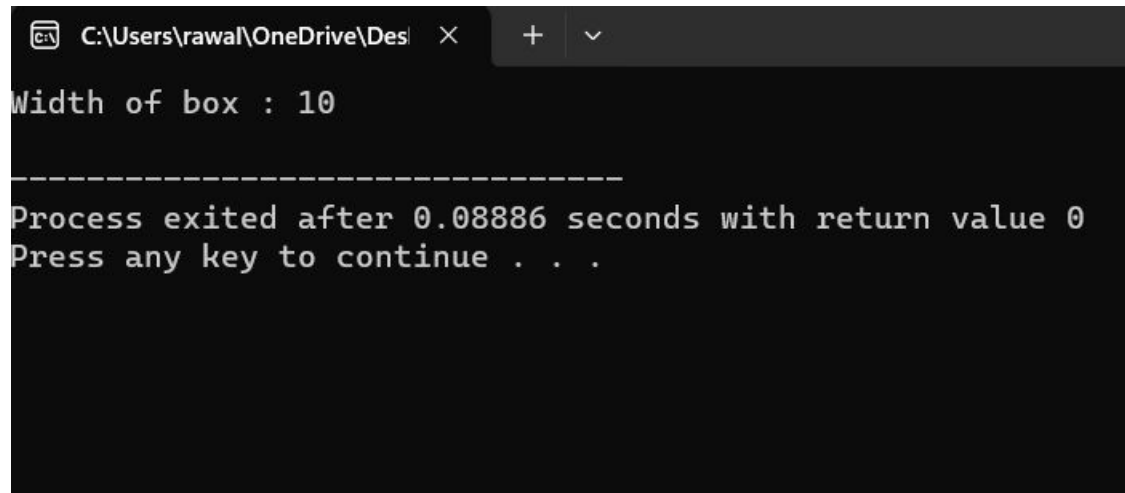
global friend function

```
#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;}
void printWidth( Box box ) {
    cout << "Width of box : " << box.width << endl;}
```

global friend function

```
int main() {  
    Box box;  
    box.setWidth(10.0);  
    printWidth( box );  
    return 0;  
}
```

global friend function



```
C:\Users\rawal\OneDrive\Desktop > .\program.exe  
Width of box : 10  
-----  
Process exited after 0.08886 seconds with return value 0  
Press any key to continue . . .
```

Addition of members of two different classes using friend Function (Multiple friends)

```
#include <iostream>
using namespace std;
class B;
class A {
    private:
        int numA;
    public:
        A(): numA(12) { }
        friend int add(A, B);};
```

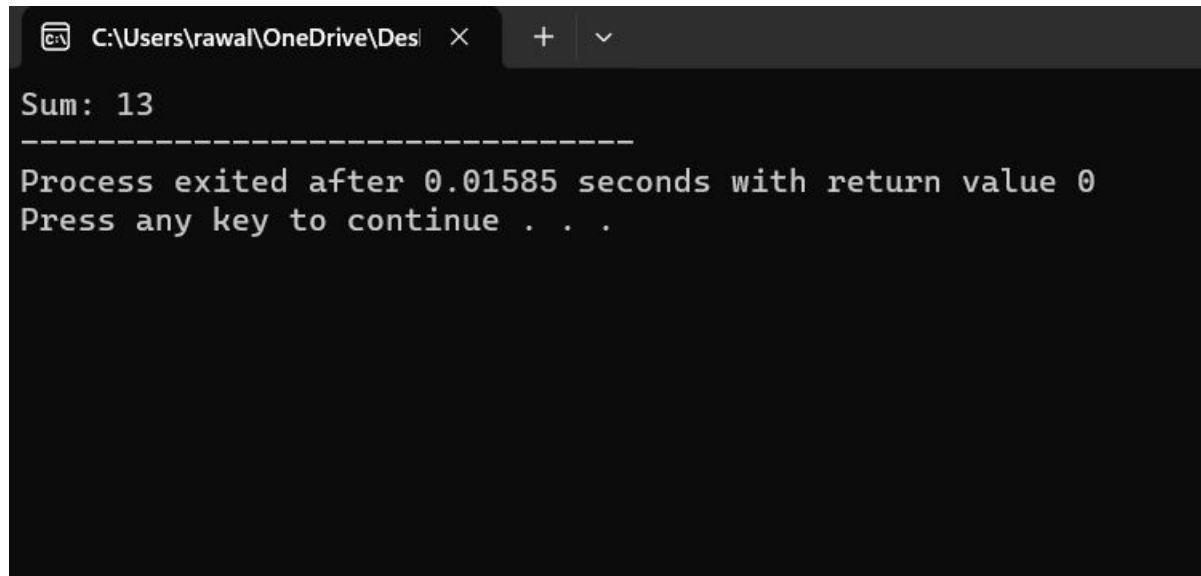
Multiple friends

```
class B {  
    private:  
        int numB;  
    public:  
        B(): numB(1) { }  
        friend int add(A , B);};  
int add(A objectA, B objectB){  
    return (objectA.numA + objectB.numB);}
```

Multiple friends

```
int main()
{
    A objectA;
    B objectB;
    cout<<"Sum: "<< add(objectA, objectB);
    return 0;
}
```

Multiple friends



```
C:\Users\rawal\OneDrive\Desktop > .\Program.exe  
Sum: 13  
-----  
Process exited after 0.01585 seconds with return value 0  
Press any key to continue . . .
```

Friend Classes:

- A friend class can access private and protected members of other class in which it is declared as friend. It is sometimes useful to allow a particular class to access private members of other class.

To declare all member functions of class ClassTwo as friends of class ClassOne, place a declaration of the form

```
friend class ClassTwo;
```

in the definition of class ClassOne.

Syntax

-

```
class B;
```

```
class A
```

```
{
```

```
    // class B is a friend class of class A
```

```
    friend class B;
```

```
    ... ..
```

```
}
```

```
class B
```

```
{
```

```
    ... ..
```

```
}
```

Friend Classes:

When a class is made a friend class, all the member functions of that class becomes friend functions.

In this program, all member functions of class B will be friend functions of class A.

Thus, any member function of class B can access the private and protected data of class A.

But, member functions of class A cannot access the data of class B.

Friend Classes:

```
1  #include <iostream>
2  class A {
3  private:
4      int a;
5  public:
6      A() { a = 0; }
7      friend class B; // Friend Class };
8  class B {
9  private:
10     int b;
11  public:
12     void showA(A x) {
13         // Since B is friend of A, it can access private members of A
14         std::cout << "A::a=" << x.a; } };
15  int main(){
16     A a;
17     B b;
18     b.showA(a);
19     return 0; }
```

friend function of another class

Member functions of other classes may be declared to be friends of another class

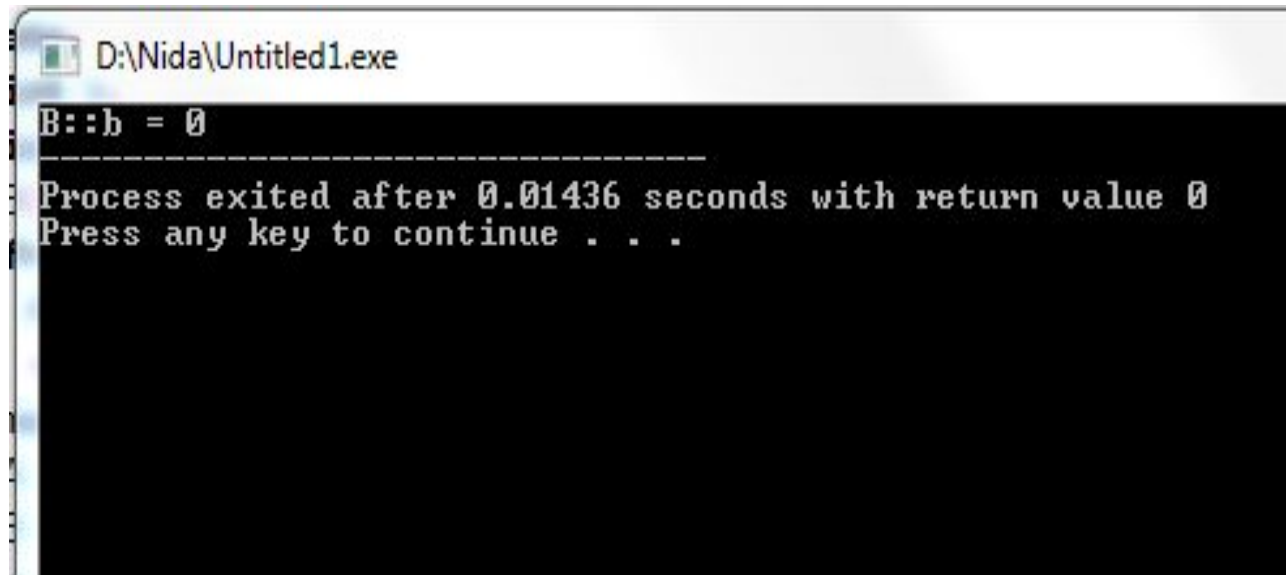
friend function of another class

```
class A {  
public:  
void showB(B);};  
class B {  
int b;  
public:  
B() { b = 0; }  
friend void A::showB(B x);}; // Friend function  
void A::showB(B x){  
cout << "B::b = " << x.b;}
```

friend function of another class

- `int main(){`
- `A a;`
- `B x;`
- `a.showB(x);`
- `return 0;}`

friend function of another class



```
D:\Nida\Untitled1.exe
B::b = 0
-----
Process exited after 0.01436 seconds with return value 0
Press any key to continue . . .
```

Why they are needed?

- They are needed in situations where we have written code for some function in one class and it need to be used by other classes as well for example, suppose we wrote the code to compute a complex mathematical formula in one class but later it was required by other classes as well, in that case we will make that function friend of all other classes.

Friendship is not **mutual/symmetric**

- Friendship is granted, not taken—i.e.,
- for class B to be a friend of class A, class A must explicitly declare that class B is its friend.

Friendship is not mutual unless explicitly specified

In the above example, member functions of YourClass cannot access the private members of YourOtherClass.

Friendship is not transitive

- if class A is a friend of class B, and class B is a friend of class C, you cannot infer that class A is a friend of class C

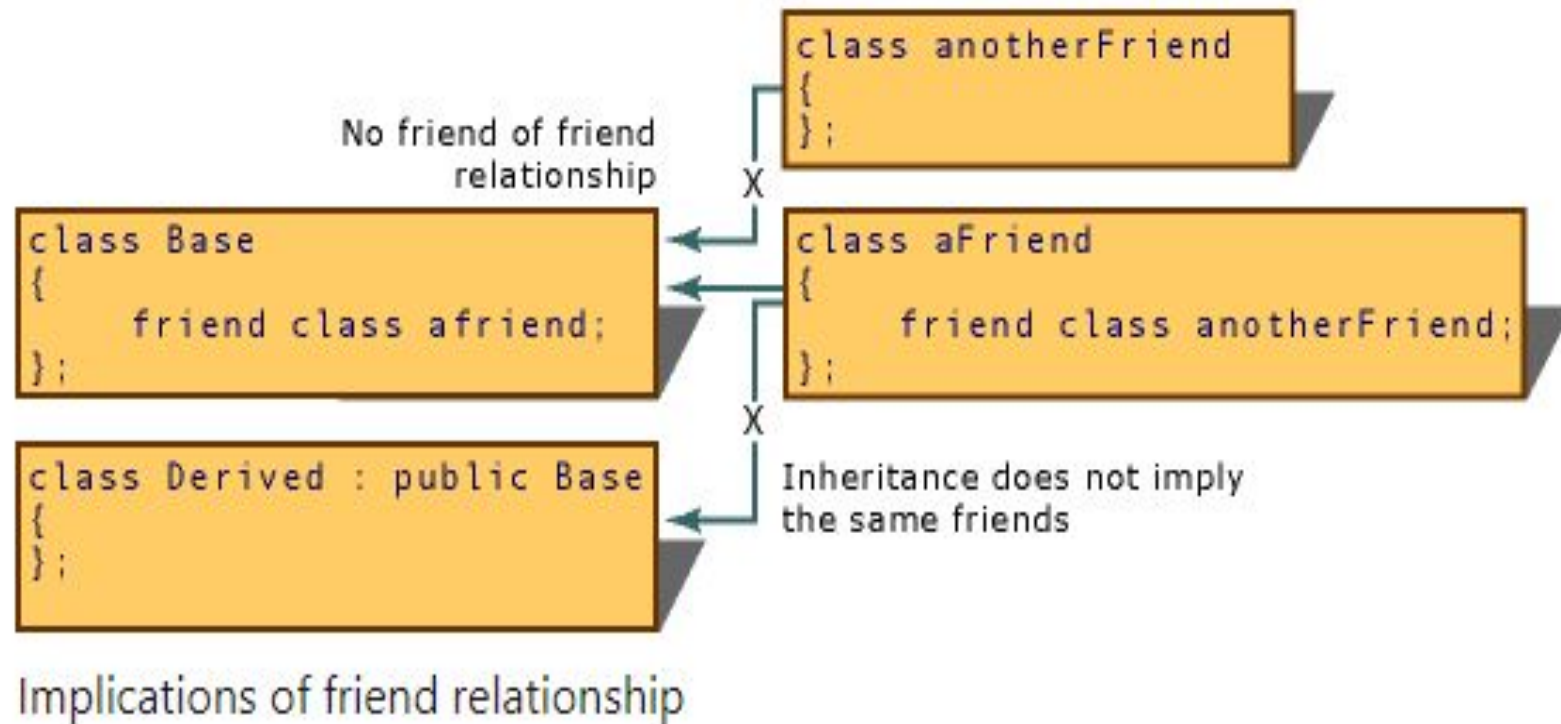
Friendship is not transitive, so classes that are friends of **YourOtherClass** cannot access **YourClass's** private members.

Friendship is not inherited

- In C++, friendship is not inherited. If a base class has a friend function, then the function doesn't become a friend of the derived class(es).

Friendship is not inherited, meaning that classes derived from **YourOtherClass** cannot access **YourClass**'s private members.

Implications of friendship



Overloaded friend Functions

- It's possible to specify overloaded functions as friends of a class.
- Each function intended to be a friend must be explicitly declared in the class definition as a friend of the class.

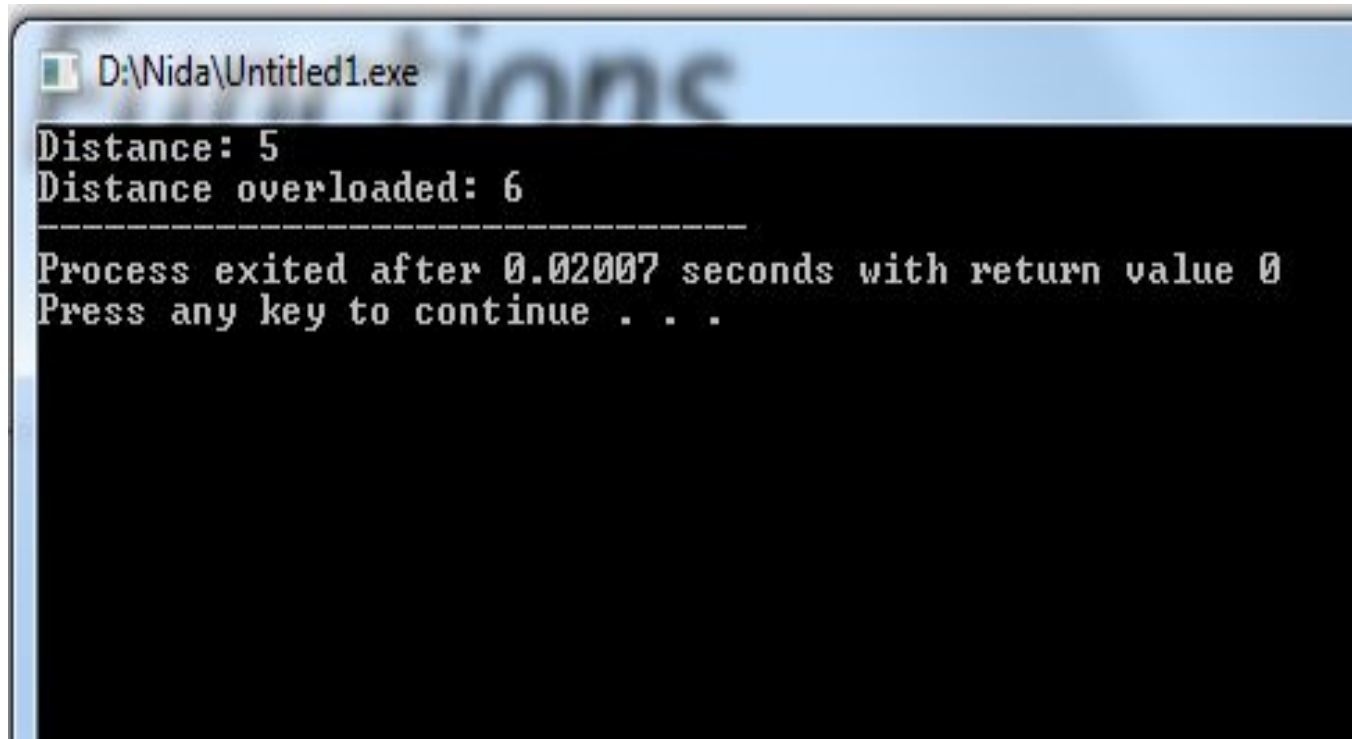
Overloaded friend Functions

```
class Distance {  
    int meter;  
    friend int addFive(Distance);  
    friend int addFive(Distance , int);  
public:  
    Distance() : meter(0) {};  
int addFive(Distance d) {  
    d.meter += 5;  
    return d.meter;}  
int addFive(Distance d , int i) {  
    d.meter += i;  
    return d.meter;}  
}
```

Overloaded friend Functions

- `int main() {`
- `Distance D;`
- `cout << "Distance: " << addFive(D) << endl;;`
- `cout << "Distance overloaded: " << addFive(D,6);`
- `return 0;}`

Overloaded friend Functions

A screenshot of a Windows command prompt window. The title bar at the top is light blue and contains the text 'D:\Nida\Untitled1.exe'. The main area of the window is black with white text. The text displayed is: 'Distance: 5', 'Distance overloaded: 6', a horizontal line of dashes, 'Process exited after 0.02007 seconds with return value 0', and 'Press any key to continue . . .'.

```
D:\Nida\Untitled1.exe
Distance: 5
Distance overloaded: 6
-----
Process exited after 0.02007 seconds with return value 0
Press any key to continue . . .
```


Operator Functions as Class Members vs. Global Functions

- Operator functions can be member functions(already discussed)
- or global functions (non-member function)
- global functions are often made friends for performance reasons.
 - performance reasons?

performance reasons

- *It's possible to overload an operator as a global, non-friend function, but such a function requiring access to a class's private or protected data would need to use set or get functions provided in that class's public interface. The overhead of calling these functions could cause poor performance, so these functions can be inlined to improve performance.*

some criteria/rules to define the operator function:

- Operator overloading function can be applied on a member function if the left operand is an object of that class, but if the Left operand is different, then the Operator overloading function must be defined as a non-member function.

some criteria/rules to define the operator function:

- Member functions use this pointer implicitly to obtain one of their class object arguments (the left operand for binary operators).
- Arguments for both operands of a binary operator must be explicitly listed in a global function call

some criteria/rules to define the operator function:

- In case of a non-static function, the binary operator should have only one argument and unary should not have an argument.
- In the case of a friend function, the binary operator should have only two argument and unary should have only one argument.

Operator Overloading using a Friend function

- In this approach, the operator overloading function must precede with friend keyword, and declare a function class scope. Keeping in mind, friend operator function takes two parameters in a binary operator, varies one parameter in a unary operator. All the working and implementation would same as binary operator function except this function will be implemented outside of the class scope.

Operator Overloading using a Friend function

- . In case of a non-static function, the binary operator should have only one argument and unary should not have an argument.
- . In the case of a friend function, the binary operator should have only two argument and unary should have only one argument.

Syntax for binary operator overloading using friend function

- friend return-type operator operator-symbol (Variable 1, Variable2)
 {
 //Statements;
 }

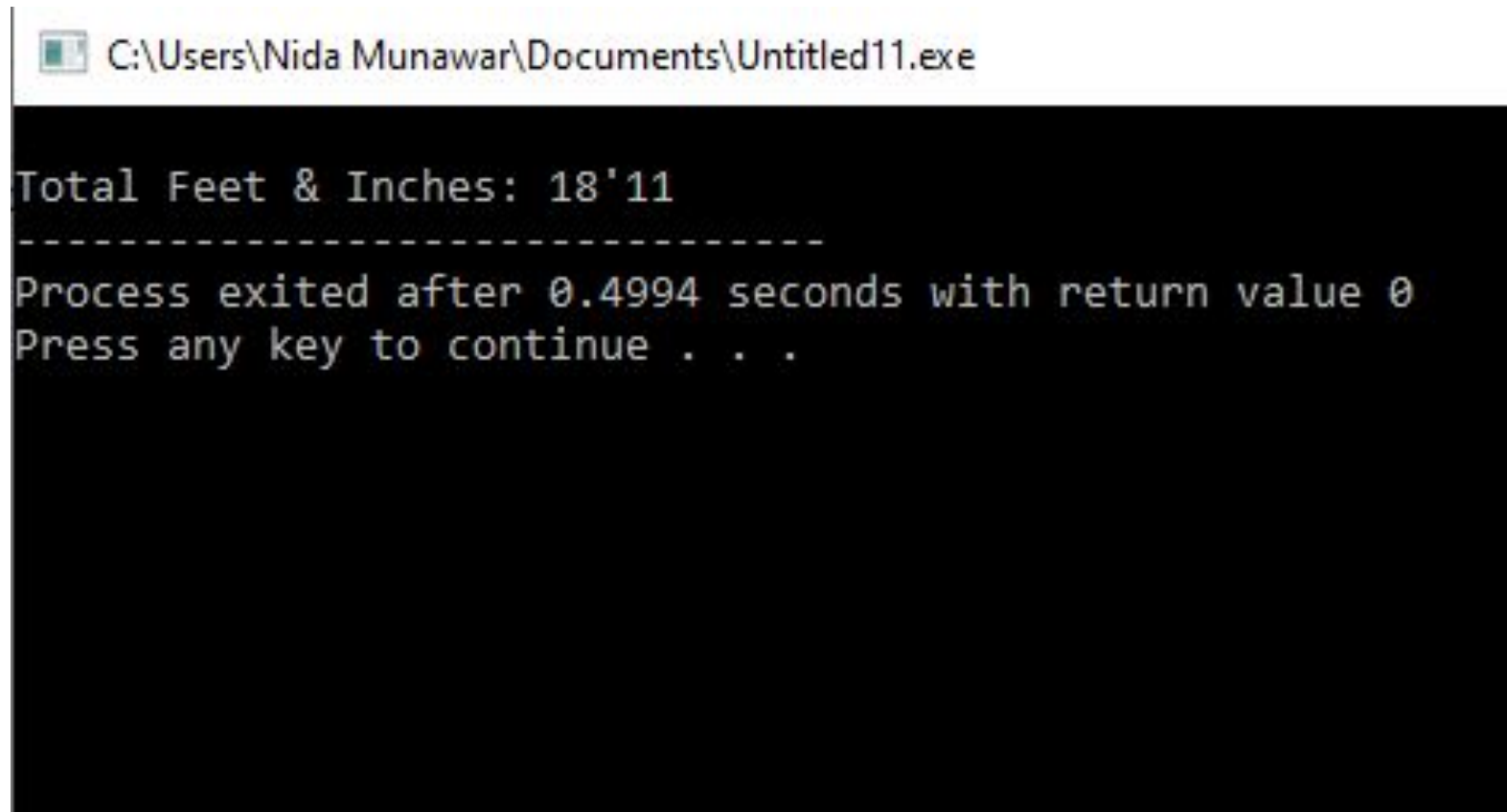
Overloading Binary Operator using a Friend function

```
class Distance {  
public:  
    int feet, inch;  
    Distance() {  
        this->feet = 0;  
        this->inch = 0; }  
    Distance(int f, int i) {  
        this->feet = f;  
        this->inch = i; }  
    friend Distance operator+(Distance, Distance); };
```

Overloading Binary Operator using a Friend function

```
Distance operator+(Distance d1, Distance d2) {  
    Distance d3;  
    d3.feet = d1.feet + d2.feet;  
    d3.inch = d1.inch + d2.inch;  
    return d3; }  
  
int main() {  
    Distance d1(8, 9);  
    Distance d2(10, 2);  
    Distance d3;  
    d3 = d1 + d2;  
    cout << "\nTotal Feet & Inches: " << d3.feet << " " << d3.inch;  
    return 0; }
```

Overloading Binary Operator using a Friend function



```
C:\Users\Nida Munawar\Documents\Untitled11.exe  
Total Feet & Inches: 18'11  
-----  
Process exited after 0.4994 seconds with return value 0  
Press any key to continue . . .
```

Overloading Binary Operator using a Friend function

- Another way of calling binary operator overloading with friend function is to call like a non member function as follows,

```
d3 =operator+ ( d1,d2 );
```

Unary operator overloading using Friend function

```
class UnaryFriend{
    int a=10;
    int b=20;
    public:
        void getvalues(){
            cout<<"Values of A and B\n";
            cout<<a<<"\n"<<b<<"\n"<<endl;}
        void friend operator-(UnaryFriend &x); };
void operator-(UnaryFriend &x){
    x.a = -x.a;    //Object name must be used as it is a friend function
    x.b = -x.b;}
```

Unary operator overloading using Friend function

```
int main(){  
    UnaryFriend x1;  
    cout<<"Before Overloading\n";  
    x1.getvalues();  
    cout<<"After Overloading \n";  
    -x1;// operator-(x1);  
    x1.getvalues();  
    return 0;}
```

Overloading Binary Operator using a Friend function

```
#include<iostream>
using namespace std;
class Distance {
public:
    int feet, inch;
    Distance() {
        this->feet = 0;
        this->inch = 0; }
    Distance(int f, int i) {
        this->feet = f;
        this->inch = i; }
    Distance operator+(Distance d1) {
        Distance d3;
        d3.feet = feet + d1.feet;
        d3.inch = inch + d1.inch;
        return d3; }
```

Overloading Binary Operator using a Friend function

```
friend Distance operator+(Distance, Distance); };  
Distance operator+(Distance d1, Distance d2) {  
    Distance d3;  
    d3.feet = d1.feet + d2.feet;  
    d3.inch = d1.inch + d2.inch;  
    return d3; }  
  
int main() {  
    Distance d1(8, 9);  
    Distance d2(10, 2);  
    Distance d3;  
    d3=d1+d2;  
    //d3=operator+(d1,d2);  
    //d3=d1.operator+(d2);  
    cout << "\nTotal Feet & Inches: " << d3.feet << " " << d3.inch;  
    return 0; }
```


Task

- Calculate Area of Triangle the one class is acute angle (less than 90 degree) and another one is obtuse angle (greater than 90 degree) both class wants to access only one function which is “Area of Triangle” which is independent of all classes.

Task 1

Create a function AssignCourse that takes all courses and

assigns one course per call to the current faculty object based on the following criteria:

- a. If the calling object is from “Computer Science” department, then assign the available course with course code starting with “C”.
- b. If the calling object is from “Management Science” department, then assign the available course with course code starting with “M”.
- c. If the calling object is from “Electrical Engineering” department, then assign the available course with course code starting with “E”.
- d. While assigning courses to the faculty, do invoke a warning message if the total assigned credit hours exceed maximum 12 credit hours.

Also create the Salary function calculates and prints the salary on the following criteria: If the faculty’s working hours are equal to 36 display the current salary. If the faculty’s working hours are more than 36 then add 1000 Rs for each extra hour and display the updated salary.