Oct 08, 2015

COMPUTER PROGRAMMING

LAB 7

ABDUL AZIZ

FAST NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES

Lab Instructor	**Mr. Abdul Aziz**
Course	**Computer Programming Lab**
Duration	**2hrs**

Objectives:

In this lab, following topics will be covered

- ❖ Header Files
- ❖ Operator Overloading
- ❖ Friend Class
- ❖ Friend function

1. Creating Header Files

A header file is a file with extension **.h** which contains C function declarations and macro definitions and to be shared between several source files. There are two types of header files: the files that the programmer writes and the files that come with your compiler.

A simple practice in C or C++ programs is that we keep all the constants, macros, system wide global variables, and function prototypes in header files and include that header file wherever it is required.

Both user and system header files are included using the preprocessing directive **#include**. It has following two forms:

```
#include <file>
```

This form is used for system header files. It searches for a file named file in a standard list of system directories. You can prepend directories to this list while compiling your source code.

```
#include “file”
```

This form is used for header files of your own program. It searches for a file named file in the directory containing the current file. You can prepend directories to this list while compiling your source code.

Note: using namespace in a header file

Namespace is just a way to mangle function signature so that it won't conflict

Example: “MyClass.h”

```
namespace MyNamespace
{
  class MyClass {
  public:
    int foo();
  };
}
```

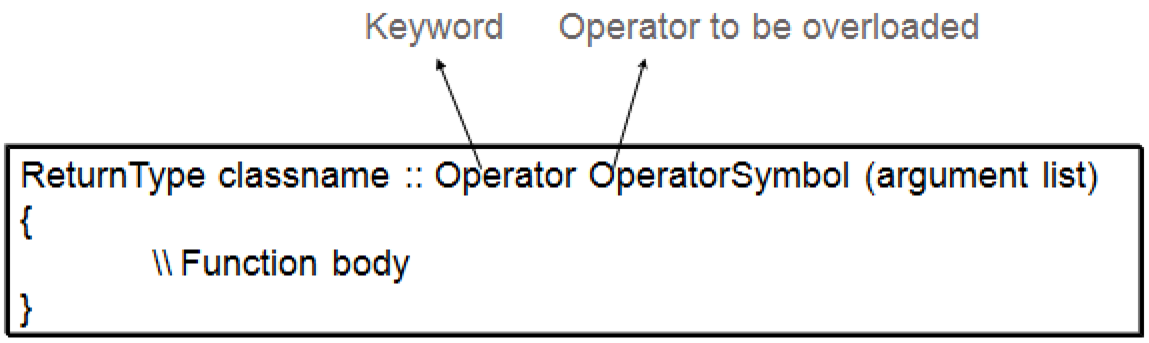
```
#include "MyClass.h"
using namespace MyNamespace;

int MyClass::foo() { ... }
```

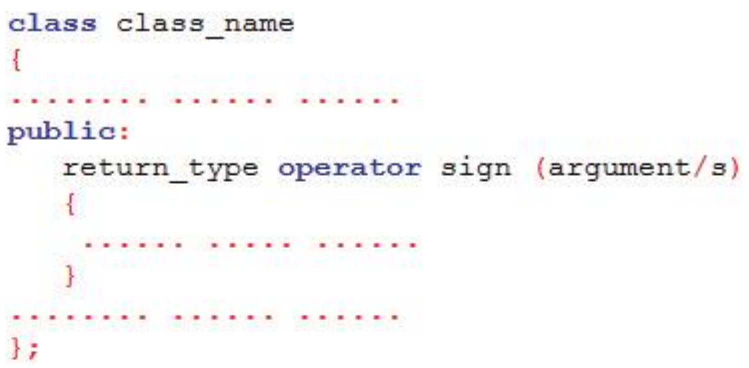
2. Operator Overloading

If you want to add two integers then, + operator is used. But, for user-defined types(like: objects), you can define the meaning of operator, i.e, you can redefine the way that operator works. For example: If there are two objects of a class that contain string as its data member, you can use + operator to concatenate two strings. Suppose, instead of strings if that class contains integer data member, then you can use + operator to add integers. This feature in C++ programming that allows programmer to redefine the meaning of operator when they operate on class objects is known as operator overloading.

To overload an operator, an operator function is defined inside a class as:



```
ReturnType classname :: Operator OperatorSymbol (argument list)
{
        \\ Function body
}
```



```
class class_name
{
........ ...... ......
public:
    return_type operator sign (argument/s)
    {
      ...... ..... ......
    }
........ ...... ......
};
```

The return type comes first which is followed by keyword **operator**, followed by operator sign, i.e., the operator you want to overload like: +, <, ++ etc. and finally the arguments are passed. Then, inside the body of you wants perform the task you want when this operator function is called.

Example:

```
/* Simple example to demonstrate the working of operator overloading*/
#include <iostream>
using namespace std;
class temp
{
   private:
      int count;
   public:
       temp():count(5){  }
       void operator ++() {
        count=count+1;
       }
       void Display() { cout<<"Count: "<<count; }
};
int main()
{
    temp t;
    ++t;        /* operator function void operator ++() is called */
    t.Display();
    return 0;
}
```

Explanation

In this program, a operator function **void operator ++ ()** is defined(inside class *temp*), which is invoked when ++ operator operates on the object of type *temp*. This function will increase the value of count by 1.

Things to remember while using Operator overloading in C++ language

1. Operator overloading cannot be used to change the way operator works on built-in types. Operator overloading only allows to redefine the meaning of operator for user-defined types.
2. There are two operators assignment operator(=) and address operator(&) which does not need to be overloaded. Because these two operators are already overloaded in C++ library. For example: If*obj1* and *obj2* are two objects of same class then, you can use code **obj1=obj2;** without overloading = operator. This code will copy the contents object of *obj2* to *obj1*. Similarly, you can use address operator directly without overloading which will return the address of object in memory.
3. Operator overloading cannot change the precedence of operators and associativity of operators. But, if you want to change the order of evaluation, parenthesis should be used.
4. Not all operators in C++ language can be overloaded. The operators that cannot be overloaded in C++ are ::(scope resolution), .(member selection), .*(member selection through pointer to function) and ?:(ternary operator).

3. Friend Function

If a function is defined as a friend function then, the private and protected data of class can be accessed from that function. The complier knows a given function is a friend function by its keyword **friend**. The declaration of friend function should be made inside the body of class (can be anywhere inside class either in private or public section) starting with keyword friend.

```
#include <iostream>
using namespace std;
class Distance
{
  private:
    int meter;
  public:
    Distance(): meter(0){ }
    friend int func(Distance);            //friend function
};
int func(Distance d)                        //function definition
{
  d.meter=5;            //accessing private data from non-member function
  return d.meter;
}
int main()
{
  Distance D;
  cout<<"Distace: "<<func(D);
  return 0;
}
```

Here, friend function func() is declared inside Distance class. So, the private data can be accessed from this function.

4. Friend Class

Just like functions are made friends of classes, we can also make one class to be a friend of another class. Then, the friend class will have access to all the private members of the other class.

```
#include<iostream>
using namespace std;
class Storage
{
private:
  int m_nValue;
  double m_dValue;
public:
  Storage(int nValue, double dValue)
```

```
    {
        m_nValue = nValue;
        m_dValue = dValue;
    }

    // Make the Display class a friend of Storage
    friend class Display;
};

class Display
{
private:
    bool m_bDisplayIntFirst;

public:
    Display(bool bDisplayIntFirst) { m_bDisplayIntFirst = bDisplayIntFirst; }

    void DisplayItem(Storage &cStorage)
    {
        if (m_bDisplayIntFirst)
            std::cout << cStorage.m_nValue << " " << cStorage.m_dValue << std::endl;
        else // display double first
            std::cout << cStorage.m_dValue << " " << cStorage.m_nValue << std::endl;
    }
};
int main()
{
    Storage cStorage(5, 6.7);
    Display cDisplay(false);

    cDisplay.DisplayItem(cStorage);

    return 0;
}
```

Because the Display class is a friend of Storage, any of Display’s members that use a Storage class object ca n access the private members of Storage directly.

Just because Display is a friend of Storage that does not mean Storage is also a friend of Display. If you want two classes to be friends of each other, both must declare the other as a friend. Finally, if c lass A is a friend of B, and B is a friend of C that does not mean A is a friend of C.

EXERCISE

1. Write a program having following conditions.

- ➢ Declare the class.
- ➢ Declare the variables and its member function.
- ➢ Using the function getvalue() to get the two numbers.
- ➢ Define the function operator +() to add two complex numbers.
- ➢ Define the function operator –()to subtract two complex numbers.
- ➢ Define the display function.
- ➢ Declare the class objects obj1,obj2 and result.
- ➢ Call the function getvalue using obj1 and obj2
- ➢ Calculate the value for the object result by calling the function operator + and operator.
- ➢ Call the display function using obj1 and obj2 and result.
- ➢ Return the values.

2. Write a class Current with a data member I of type integer and a member function say, voltCalculator(). Define another class Resistor with a private data member R of type integer. Initialize R in an argumented constructor. Define Current class as a friend in class Resistor. In the function voltCalculator(), print the voltage according to the formula: V=IR.

3. Create a class RationalNumber (fractions) with the following capabilities

a. Create a constructor that prevents a 0 denominator in a fraction, reduces or simplifies fractions that are not in reduced form and avoids negative denominators.

b. Overload the addition, subtraction, multiplication and division operators for this class

c. Overload the relational and equality operators for this class.