|  |  |  |
| --- | --- | --- |
| **Rubrics for Object Oriented Programming Lab** | | |
| **Lab #:** | **04** | |
| **Lab Title:** | **Operator Overloading** | |
| **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#04**

**Operator Overloading**

1. **Objectives:**

Objective of this lab is to get familiar with:

* Use of unary operators for user defined classes.
* Use of binary operators for user defined classes.
* Overall use and advantages of operator overloading in user defined classes.

1. **Introduction:**

One of the nice features of C++ is that you can give special meanings to operators, when they are used with user-defined classes. This is called operator overloading. You can implement C++ operator overloads by providing special member-functions on your classes that follow a particular naming convention. For example, to overload the + operator for your class, you would provide a member-function named operator+ on your class.

**The operator Keyword:**

The keyword operator is used to overload the ++ operator in this declarator:

**void operator ++ ()**

The return type (void in this case) comes first, followed by the keyword operator, followed by the operator itself (++), and finally the argument list enclosed in parentheses (which are empty here).

The following set of operators is commonly overloaded for user-defined classes:

1. **Examples**

|  |  |
| --- | --- |
| ++ -- (Increment and Decrement Operators) | Unary Operators |
| = (Assignment Operator)  + - \* (Binary Arithmetic Operators) |  |
| += -= \*= (Compound Assignment Operators)  == != <> (Comparison Operators) | Binary Operators |
| <<>> (Insertion and Extraction Operators) | Stream Operators |

* 1. **Example of Unary Operators Overloading**

The unary operators operate on a single operand and following are the examples of Unary operators:

* + - The increment (++) and decrement (--) operators.
    - The unary minus (-) operator.
    - The logical not (!) operator.

The unary operators operate on the object for which they were called and normally, this operator appears on the left side of the object, as in !obj, -obj, and ++obj but sometime they can be used as postfix as well like obj++ or obj--.

**4.2. Example of Binary Operators Overloading**

The binary operators take two arguments and following are the examples of Binary operators. You use binary operators very frequently like addition (+) operator, subtraction (-) operator and division (/) operator.

1. **In-Lab Tasks:**

**Task#1:** To the Distance class in the above given example in this chapter, add an overloaded - operator that subtracts two distances. It should allow statements like dist3= dist1-dist2;. Assume that the operator will never be used to subtract a larger number from a smaller one (that is, negative distances are not allowed).

**Solution:**

* **Code:**

#include<iostream>

using namespace std;

class Distance{

private:

int feet;

float inches;

public:

Distance():feet(0),inches(0.0)

{

}

Distance(int ft,float in):feet(ft),inches(in)

{

}

void getDist()

{

cout<<"\n Enter Feet : ";

cin>>feet;

cout<<"\n Enter inches : ";

cin>>inches;

}

void showDist()const

{

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

}

Distance operator + (Distance)const;

Distance operator - (Distance)const;

};

Distance Distance::operator +(Distance d2)const

{

int f=feet+d2.feet;

float i=inches+d2.inches;

if(i>=12.0)

{

i-=12.0;

f++;

}

return Distance(f,i);

}

Distance Distance::operator -(Distance d3)const

{

if(feet<d3.feet)

{

cout<<endl<<"d4 is greater than d3";

exit(0);

}

else

{

int f=feet-d3.feet;

float i=inches-d3.inches;

if(i>=12.0)

{

i-=12.0;

f++;

}

return Distance(f,i);

}

}

int main()

{

Distance dist1,dist3,dist4,dist5;

dist1.getDist();

Distance dist2(11,6.25);

dist3=dist1+dist2;

dist4=dist1+dist2+dist3;

cout<<endl<<"dist 1 = ";

dist1.showDist();

cout<<endl;

cout<<endl<<"dist 2 = ";

dist2.showDist();

cout<<endl;

cout<<endl<<"dist 3 = ";

dist3.showDist();

cout<<endl;

cout<<endl<<"dist 4 = ";

dist4.showDist();

cout<<endl;

dist5=dist3-dist4;

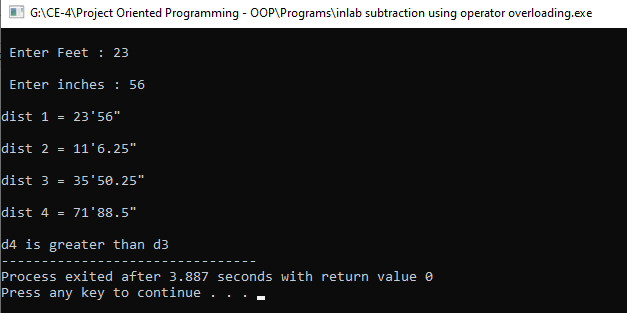
cout<<endl<<"dist 5 = ";

dist5.showDist();

return 0;

}

* **Output:**



**Task#2:** Write a program for class time which overloads the operator + to add two times instead of using function. The class time should contain hours, minutes and seconds as parameters.

**Solution:**

* **Code:**

#include<iostream>

using namespace std;

class time

{

private:

int hours,minutes,seconds;

public:

time (int h=0,int m=0,int s=0)

{

hours=h; minutes=m; seconds=s;

}

void show()const

{

cout<<endl<<"Hours : "<<hours;

cout<<endl<<"Minutes : "<<minutes;

cout<<endl<<"Seconds : "<<seconds;

}

time operator + (time T2)

{

int h,m,s;

h=hours+T2.hours;

m=minutes+T2.minutes;

s=seconds+T2.seconds;

if(m>60)

{

h++;

m-=60;

}

if(s>60)

{

s-=60;

m++;

}

return time(h,m,s);

}

};

int main()

{

time t1,t2(4,5,3),t3(5,2,6);

cout<<"\n t1 : "<<endl;

t1.show();

t1=t2+t3;

cout<<"\n t2 : "<<endl;

t2.show();

cout<<"\n t3 : "<<endl;

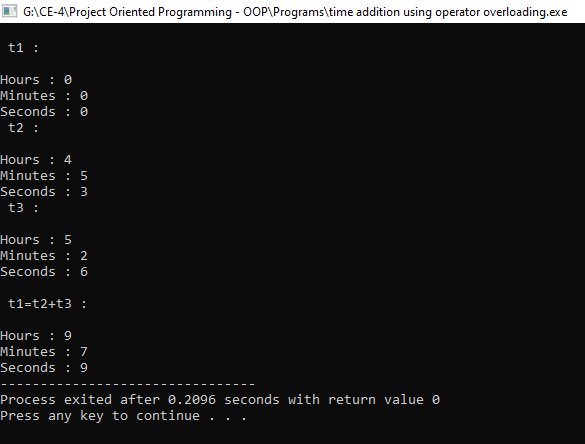
t3.show();

cout<<"\n \n t1=t2+t3 : "<<endl;

t1.show();

}

* **Output:**

****

1. **Post-Lab Tasks:**

**Task#1:** Create a class Int. Overload four integer arithmetic operators (+, -, \*, and /) so that they operate on objects of type Int. If the result of any such arithmetic operation exceeds the normal range of int (in a 32-bit environment)— from 2,147,483,648 to –2,147,483,647—have the operator print a warning and terminate the program. Such a data type might be useful where mistakes caused by arithmetic overflow are unacceptable.

Hint: To facilitate checking for overflow, perform the calculations using type long double. Write a program to test this class.

**Solution:**

* **Code:**

#include <iostream>

using namespace std;

class Int

{

private:

long i;

public:

Int( ) :i(0)

{

}

Int(int n) :i(n)

{

}

void display( Int obj2, Int obj3, char ch ) const

{

cout<< obj2.i <<ch<< obj3.i << " = " << i <<endl;

}

Int operator + (Int) const;

Int operator - (Int) const;

Int operator \* (Int) const;

Int operator / (Int) const;

};

Int Int::operator + (Int temp) const

{

long r;

r = i+temp.i;

return Int(r);

}

Int Int::operator - (Int temp) const

{

long r;

r = i-temp.i;

return Int(r);

}

Int Int::operator \* (Int temp) const

{

long r;

r = i\*temp.i;

return Int(r);

}

Int Int::operator / (Int temp) const

{

long r;

r = i/temp.i;

return Int(r);

}

int main()

{

Int i1,i2(250),i3(122);

i1=i2+i3;

i1.display( i2,i3,'+' );

i1 = i2-i3;

i1.display( i2,i3,'-' );

i1 = i2\*i3;

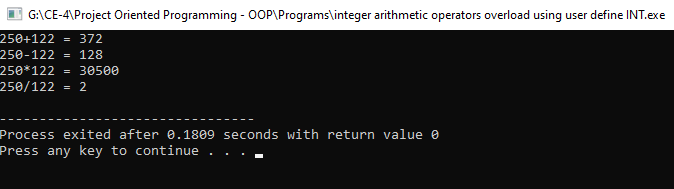
i1.display( i2,i3,'\*' );

i1 = i2/i3;

i1.display( i2,i3,'/' );

}

* **Output:**



1. **Conclusion:**

After completing this lab, we are able to know:

* the purpose of Operator Overloading.
* The use unary and binary operators for user defined classes.