|  |  |  |
| --- | --- | --- |
| **Rubrics for Object Oriented Programming Lab** | | |
| **Lab #:** | **05** | |
| **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 05 – Operator Overloading (Contd.)**

1. **Objectives**

The objective of this lab is to teach the students, the use of special binary operators i.e. comparison and compound assignment operators for user defined classes i.e. operator overloading.

1. **Outcome** 
   1. At the end of this lab student will know the purpose of Operator Overloading.
   2. Student will be able to use of special binary operators for user defined classes.
2. **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 following set of operators is commonly overloaded for user-defined classes:

**4.In-Lab Tasks**

**4.1** Use the Arithmetic Assignment operator (+=) for a Distance class to add one distance to a second, leaving the result in the first. This is similar to example shown earlier, but there is a subtle difference.

**CODE:**

#include<iostream>

using namespace std;

class Distance

{

private:

int Feet;

float Inches;

public:

Distance():Feet(0),Inches(0)

{ }

Distance(int ft,float in)

{

Feet=ft;

Inches=in;

}

void Get\_Dist()

{

cout<<"\n\t Enter Feets : ";cin>>Feet;

cout<<"\n\t Enter Inches : ";cin>>Inches;

}

void Disply() const

{

cout<<"\n\t\t Feets = "<<Feet<<" Inches = "<<Inches;

}

Distance operator +=(Distance B);

};

Distance Distance::operator +=(Distance B)

{

this->Feet+=B.Feet;

this->Inches+=B.Inches;

while(Inches>12)

{

if(Inches >= 12)

{

Inches=Inches-12;

Feet++;

}

}

}

int main()

{

Distance D1,D2;

D1.Get\_Dist();

D2.Get\_Dist();

cout<<"\n\t Addition : ";

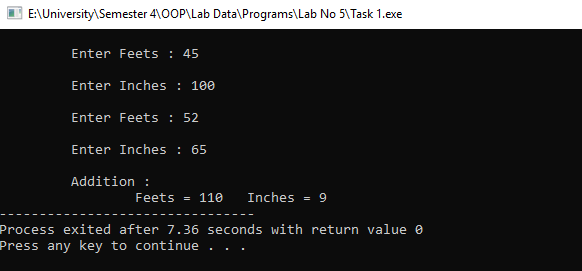
D1+=D2;

D1.Disply();

return 0;

}

**OUTPUT:**



**4.2** Write a program that substitutes an overloaded += operator for the overloaded + operator in the example given above. This operator should allow statements like s1 += s2; where s2 is added (concatenated) to s1 and the result is left in s1. The operator should also permit the results of the operation to be used in other calculations, as in s3 = s1 += s2;

**CODE:**

#include<iostream>

using namespace std;

#include<string.h>

#include<stdlib.h>

class String

{

private:

enum

{

SZ=80

};

char str[SZ];

public:

String()

{

strcpy(str,"");

}

String(char s[])

{

strcpy(str,s);

}

void display()const

{

cout<<str;

}

String operator += (String ss)

{

if(strlen(str)+strlen(ss.str)<SZ)

{

strcat(str,ss.str);

}

else

{

cout<<"\n String overflow "; exit(1);

}

}

};

int main()

{

String s1=" Merry Christmas ! ";

String s2="Happy New Year ! ";

cout<<"\n String Number #1 \n\t\t";

s1.display();

cout<<"\n String Number #2 \n\t\t";

s2.display();

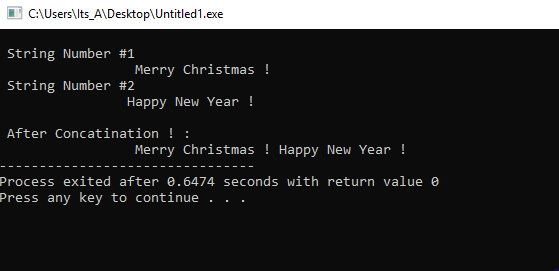
s1+=s2;

cout<<"\n\n After Concatination ! : \n\t\t";

s1.display();

}

**OUTPUT:**



**4.3** Write a program for class String that uses an overloaded == operator for comparing two strings together. This operator should allow statements like s1 == s2; and display the results showing whether the strings entered by user are same or not.

**CODE:**

#include<iostream>

using namespace std;

#include<string.h>

#include<stdlib.h>

class String

{

private:

enum

{

SZ=80

};

char str[SZ];

public:

String()

{

strcpy(str,"");

}

String(char s[])

{

strcpy(str,s);

}

void display()const

{

cout<<str;

}

bool operator == (String ss)

{

if(strcmp(str,ss.str))

{

return 0;

}

else

{

return 1;

}

}

};

int main()

{

String s1="\n\t Merry Christmas ! ";

String s2="\n\t Happy New Year ! ";

cout<<"\n String No # 1 ";

s1.display();

cout<<"\n String No # 2 ";

s2.display();

cout<<"\n\n Comparison Results ";

if(s1==s2)

{

cout<<"\n\t\t Both Are Equal ! ";

}

else

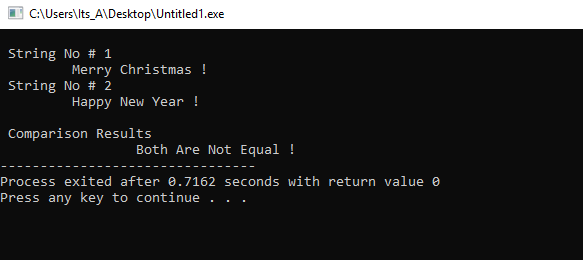
{

cout<<"\n\t\t Both Are Not Equal ! ";

}

}

**OUTPUT:**



1. **Post Lab Tasks:**
   1. Write a program for class time which has the ability to subtract two time values using the overloaded (- ) operator, and to multiply a time value by a number of type float, using the overloaded (\*) operator.

**CODE:**

#include<iostream>

using namespace std;

class Time

{

private:

int Hour;

int Mint;

int Sec;

public:

Time(); // Constructor

void GetData();//Input Function

Time operator -(Time B); // Operator 1

Time operator \*(float N); // Operator 2

void GetDisplay(); // Output

};

Time::Time()

{

Hour=0;Mint=0;Sec=0;

}

void Time::GetData()

{

cout<<"\n\t Please Enter The Hours : ";cin>>Hour;

cout<<"\n\t Please Enter The Minutes : ";cin>>Mint;

cout<<"\n\t Please Enter The Seconds : ";cin>>Sec;

}

void Time::GetDisplay()

{

cout<<"\n\t The Subtraction : "<<Hour<<"/"<<Mint<<"/"<<Sec<<endl<<endl;

}

Time Time::operator -(Time B)

{

Time Temp;

Temp.Hour=this->Hour-B.Hour;

Temp.Mint=this->Mint-B.Mint;

Temp.Sec=this->Sec-B.Sec;

if(this->Hour<B.Hour)

{

Temp.Hour=-1\*Temp.Hour;

}

if(this->Mint<B.Mint)

{

Temp.Mint=-1\*Temp.Mint;

}

if(this->Sec<B.Sec)

{

Temp.Sec=-1\*Temp.Sec;

}

while(Temp.Sec>=60)

{

Temp.Sec=Temp.Sec-60;

Temp.Mint++;

}

while(Temp.Mint>=60)

{

Temp.Mint=Temp.Mint-60;

Temp.Hour++;

}

return Temp;

}

Time Time::operator \*(float N)

{

this->Hour=this->Hour\*N;

this->Mint=this->Mint\*N;

this->Sec=this->Sec\*N;

while(this->Sec>=60)

{

this->Sec=this->Sec-60;

this->Mint++;

}

while(this->Mint>=60)

{

this->Mint=this->Mint-60;

this->Hour++;

}

}

int main()

{

float Num;

Time Alpha[3];

cout<<"\n Please Enter The Time No 1 : \n";

Alpha[1].GetData();

cout<<"\n Please Enter The Time No 2 : \n";

Alpha[2].GetData();

Alpha[0]=Alpha[2]-Alpha[1];

Alpha[0].GetDisplay();

cout<<"\n\t Please Enter The Number You Want To Multiply With Time : ";cin>>Num;

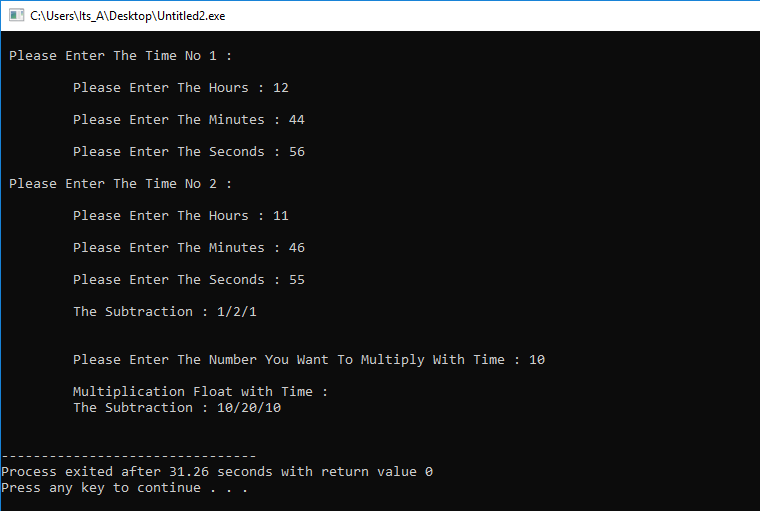
Alpha[0]\*Num;

cout<<"\n\t Multiplication Float with Time : ";

Alpha[0].GetDisplay();

}

**OUTPUT:**



* 1. Create a class Cartesian such that it compares the x and y co-ordinates of two points in a plain using an overloaded operator and after comparison print out which one of the point will be the head of the vector and which one will be tail if a vector would be formed between these two points keeping that a vector will go from smallest to largest point. Also check if both points are at the same place using overloaded operator then print out an error message “Both points are same…Resultant vector can’t be created”.

**CODE:**

#include<iostream>

#include<math.h>

using namespace std;

class Carte

{

private:

int X;int Y;int R;

public:

Carte(); //Constructor

void GetData(); // Input

void Display(); //Output

int Resultant();

bool operator ==(Carte B);

};

int Carte::Resultant()

{

R=sqrt((X\*X)+(Y\*Y));

return(R);

}

Carte::Carte()

{

X=0;

Y=0;

}

void Carte::GetData()

{

cout<<"\n\t Please Enter The X-Cordinate : ";cin>>X;

cout<<"\n\t Please Enter The Y-Cordibate : ";cin>>Y;

}

void Carte::Display()

{

cout<<"\n\t("<<X<<","<<Y<<")";

}

bool Carte::operator == (Carte B)

{

if(this->Resultant()==B.Resultant())

{

return 1;

}

else

return 0;

}

int main()

{

Carte Alpha[2];

cout<<"\n Please Enter The Coordinate No 1 : \n";

Alpha[0].GetData();

cout<<"\n Please Enter The Coordinate No 2 : \n";

Alpha[1].GetData();

if(Alpha[0]==Alpha[1])

{

cout<<"\n\t Sorry You Have Entered The Wrong Cordinates";

}

else if(Alpha[0].Resultant()>Alpha[1].Resultant())

{

cout<<"Head point : "<<endl;

Alpha[0].Display();

cout<<endl<<"Tail point : "<<endl;

Alpha[1].Display();

}else

{

cout<<"Head point : "<<endl;

Alpha[1].Display();

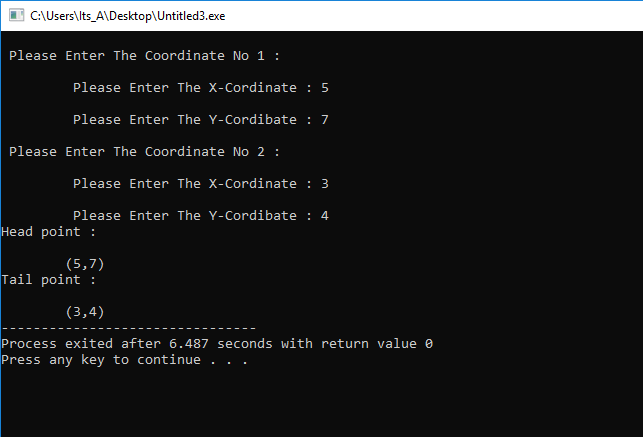
cout<<endl<<"Tail point : "<<endl;

Alpha[2].Display();

}

}

**OUTPUT:**



**Conclusion:**

* Operator overloading enables programmers to use notation closer to the target domain.
* Operator overloading provides similar syntactic support of built-in types to user-defined types.
* Operator overloading makes the program easier to understand.