**Lab-3 Operator Overloading & Type Casting**

1. Write a program to overload unary minus operator.

2. a ) Write a program to add two complex numbers by overloading binary operator ( + ).

b). Write a program to overload ( = ) operator.

#include <iostream>

using namespace std;

class Sample

{

int num;

public:

Sample(){}

Sample(int n)

{ num = n; }

void operator = (Sample &y )

{

cout<<"Value = "<<y.num;

}

};

int main() {

Sample val;

Sample f(2);

val = f;

return 0;

}

c) Write a program to find largest of two numbers by overloading ( > ) operator.

d) Write a program to overload “ ++ “ (both prefix and postfix ) operator.

3. Write a program showing basic to class type conversion

4. Write a program to convert hours into minutes. Use the concept of user-defined to basic type conversion.

#include<iostream>

#include<math.h>

using namespace std;

class Hour

{

int hr;

public:

Hour() { }

operator int()

{

int minute;

minute= hr \* 60;

return (minute);

}

void getdata()

{

cout<<"Enter Hours";

cin>>hr;

}

};

int main()

{

Hour h1;

float min;

h1.getdata();

min = h1; //basic to user defined type

cout<<"Minutes = "<<min;

}

5. Convert **rectangle to polar** by Class type to Class type conversion using constructor in the destination class.

6. Convert **polar to rectangle** type conversion using casting operator in the destination class.

#include<iostream>

#include<math.h>

#define PI 3.141592654

using namespace std;

class rectangle //*destination class*

{

float x;

float y;

public:

rectangle(){ }

rectangle(float a, float b)

{ x=a;

y=b;

}

void show()

{

cout<<"x="<<x<<" "<<"y="<<y;

}

};

class polar //*source class*

{ float radius;

float thita;

public:

polar(){ radius =0.0,thita=0.0;}

polar(float r,float t)

{ radius= r;

thita= t;

}

operator rectangle() {

double a= radius \* cos(thita);

double b= radius \* sin(thita);

return(rectangle(a,b));

}

void show()

{

cout<<"radius is="<<radius<<" and "<<"thita="<<thita;

}

};

int main()

{

rectangle r1;

polar p1(10.8167,56.338\*PI/180);

r1=p1;

cout<<"\npolar coordinate"<<endl;

p1.show();

cout<<"\n\nRectangle coordiante "<<endl;

r1.show();

return 0;

}

7. Convert  **polar to rectangle** by Class type to Class type conversion using constructor in the destination class. (reverse of Q No. 5)

8. Convert **rectangle to polar** type conversion using casting operator in the destination class. (reverse of Q No. 6)