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

1. Write a program to overload unary minus operator.

#include<iostream>

using namespace std;

class space

{

int x;

int y;

int z;

public:

void getdata (int a, int b, int c);

void display (void);

void operator - (); // overloaded unary minus

};

void space:: getdata (int a, int b, int c)

{

x = a;

y = b;

z = c;

}

void space :: display (void)

{

cout <<x<< " " ;

cout << y << " " ;

cout<< z << "\n" ;

}

void space :: operator - () // here operator is a keyword

{

x = -x;

y = -y;

z = -z;

}

int main ()

{

space s;

s.getdata (10, -20, 30);

cout << "S:";

s.display();

-s; // activates operator-() function

cout << "S:";

s.display ();

}

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

# include<iostream>

using namespace std;

class complex

{

float x;

float y;

public:

complex () {}

complex (float real, float imag)

{

x = real; y = imag;

}

complex operator + (complex);

void display();

};

complex complex :: operator + (complex c)

{

complex temp;

temp.x = x + c.x;

temp.y = y + c.y;

return (temp);

}

void complex :: display (void)

{

cout<< x << "+j" <<y<<"\n";

}

int main ()

{

complex C1, C2, C3; // invokes constructor1

C1 = complex (2.5,3.5); // invokes constructor 2

C2 = complex (1.6, 2.7);

C3 = C1 + C2; // activates operator + () function

cout<<"C1= " ; C1.display();

cout<<"C2= " ; C2.display();

cout<<"C3= "; C3.display ();

}

2 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;

}

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

#include<iostream>

using namespace std;

class Sample {

private:

int greater;

public:

Sample(int r)

{greater = r;}

Sample operator > (Sample &obj)

{

if(greater > obj.greater)

cout<<"\nGreater number is: " <<greater;

else

cout<<"\nGreater number is: " <<obj.greater;

}

};

int main()

{

Sample c1(10);

Sample c2(2);

c1 > c2;

return 0;

}

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

**/\* Overloading prefix ++ operator \*/**

#include<iostream>

using namespace std;

class Time

{

private:

int hours;

int minutes;

public:

Time()

{ hours = 0;

minutes = 0;

}

Time(int h, int m)

{ hours = h; minutes = m;

}

void displayTime()

{

cout << "H: " << hours << " M:" << minutes <<endl;

}

Time operator++ (int) ***// overloaded postfix ++ operator***

{

++minutes;

if(minutes >= 60)

{ ++hours;

minutes -= 60;

}

return Time(hours, minutes);

}

};

int main()

{

Time T1(11, 59);

T1++; // increment T1

T1.displayTime(); // display T1

T1++; // increment T1 again

T1.displayTime(); // display T1

return 0;

}

**/\* overloading prefix ++ operator \*/**

/\* Overloading prefix ++ operator \*/

#include<iostream>

using namespace std;

class Time

{

private:

int hours;

int minutes;

public:

Time()

{ hours = 0;

minutes = 0;

}

Time(int h, int m)

{ hours = h; minutes = m;

}

void displayTime()

{

cout << "H: " << hours << " M:" << minutes <<endl;

}

**Time operator++ () *// overloaded prefix ++ operator***

{

++minutes;

if(minutes >= 60)

{ ++hours;

minutes -= 60;

}

return Time(hours, minutes);

}

};

int main()

{

Time T1(11, 59);

++T1; // increment T1

T1.displayTime(); // display T1

++T1; // increment T1 again

T1.displayTime(); // display T1

return 0;

}

3. Write a program showing basic to class type conversion

#include<iostream>

using namespace std;

class X

{

int z;

char y;

public:

X() { }

X (char p)

{

z = (int)p;

y = p;

}

void show()

{

cout<<z<<y;

}

};

int main ()

{

char s = 'a';

X x1;

x1 = s; // *calls parameterized constructor. 's' is basic type and x1 is class type.*

x1.show();

return 0;

}

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.

#include<iostream>

#include<math.h>

using namespace std;

class rectangle

{ float x,y;

public:

rectangle(float a, float b)

{ x=a;

y=b;

}

float get\_x()

{ return(x);

}

float get\_y()

{ return(y);

}

};

class polar

{ float radius,thita;

public:

void show();

polar(){ }

polar(rectangle r)

{ float tempx=r.get\_x();

float tempy=r.get\_y();

radius = sqrt(tempx\*tempx + tempy\*tempy);

thita = atan(tempy/tempx);

}

};

void polar :: show()

{ cout<<"radius is:"<<radius<<endl;

cout<<"thita is:"<<thita\*(180/3.14);

}

int main()

{

rectangle r(6,9);

polar p(r);

p.show();

return 0;

}

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.