8)**Write a program to implement conversion routines**

//to convert polar to rectangular and vice versa using operator overloading concept

#include<iostream>

#include<math.h>

using namespace std;

class Rectangle

{

public:

double x,y;

Rectangle()

{

x=y=0;

}

Rectangle(double a,double b)

{

x=a;

y=b;

}

void setX(double x1)

{

x=x1;

}

void setY(double y1)

{

y=y1;

}

double getX(){return x;}

double getY(){return y;}

void dispRect()

{

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

}

};

class Polar

{

public:

double r,theta;

Polar()

{

r=0;

theta=0;

}

Polar(double r1,double t)

{

r=r1;

theta=t;

}

void setR(double r1)

{

r=r1;

}

8)

void setT(double t)

{

theta=t;

}

double getR()

{

return r;

}

double getT()

{

return theta;

}

Polar(Rectangle &y)

{

r=sqrt((y.x\*y.x)+(y.y\*y.y));

theta=atan(y.y/y.x);

}

operator Rectangle()

{

Rectangle r1;

r1.x=r\*cos(theta);

r1.y=r\*sin(theta);

return r1;

}

void dispPo()

{

cout<<"Radius ="<<r<<" Theta ="<<theta<<endl;

}

};

int main()

{

Polar p1(1,0);

Rectangle r2;

r2=p1; //call the operator Rectangle() function

Rectangle r3(1,1);

Polar p2;

p2=r3; //call the constructor Polar(Rectangle)

r2.dispRect();

r3.dispRect();

p1.dispPo();

p2.dispPo();

return 0;

}

OUTPUT:

