```
Script started on Sat 08 Dec 2012 09:56:19 PM CST
\033]0;georgia@georgia-MT6017: ~/cplusplus\007georgia@georgia-MT6017:~/cplusplus$ p
wd
/home/georgia/cplusplus
\033]0;georgia@georgia-MT6017: ~/cplusplus\007georgia@georgia-MT6017:~/cplusplus$ C
PP --version
This is CPP version 1.219 executing under perl v5.12.4 and compiling with:

g++ (Ubuntu/Linaro 4.6.1-9ubuntu3) 4.6.1
Copyright (C) 2011 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.


\033]0;georgia@georgia-MT6017: ~/cplusplus\007georgia@georgia-MT6017:~/cplus[illegible]
at shapes.info
Name: Jakob Hansen
Class: CSC122 - Evening Section
Lab: Shapes

Levels attempted:

4 - For just getting the basic instructions down.

+ 3 for 3rd order bezier curves.

+ 2 for length of a curve.

+ 4 for drawing the shapes


Program Description:

A class hierachy to describe a series of different shapes in 1, 2, and 3
dimensions. The program prints information about the shapes, as well as drawing
(a few) of them...
\033]0;georgia@georgia-MT6017: ~/cplusplus\007georgia@georgia-MT6017:~/cplusplus$ c
at shapes.tpq
1. How many libraries did you create for your hierarchy? Do all of them have
both interface and implementation files?

Two - one for 3D points and another for the shapes themselves.

2. How can you store information about so many different classes in a single
container?

By using virtual functions and derived classes, since they're different shapes
but in a way they're all the "same" thing and they all do the "same" things...

3. What does that new keyword virtual have to do with any of this?

It tells the compiler which methods are the part of the "common language" of
the base class.

4. Will you ever need/want to create an object of type Shape, OneD, TwoD, or
ThreeD? How can you assure that this won't happen?

Nope...and you can make sure by making their methods "pure" virtual functions,
or rather tell the compiler that those classes don't "do" anything.

5. What other methods/operators might prove useful in an application for drawing
shapes? What if the application were more of a computer-aided instruction in
geometry? Is there a need to limit your classes? (Note: You don't have to
implement these, I'm just looking for descriptive responses.)

I found + and - for matrix addition and subraction and * for scalar
multiplication quite useful...and there are a whole host of tantalizing
prospects for function objects, like passing arguments that would transform
or translate shapes maybe?

6 .What kind of container should you use to store the Shapes: dynamic array,
static array, templated dynamic Array class, vector, ...? Since this lab has
nothing to do with array management, what would be the most appropriate/easiest
choice? (After all, you have nothing more to learn about container management...
for now.)

I used a simple static array, because I didn't really need to do any management
of the list besides loop through it... I used vectors inside the class though,
and I can proudly say I did so because it is much, much, easier!


\033]0;georgia@georgia-MT6017: ~/cplusplus\007georgia@georgia-MT6017:~/cplusplus$ c
[illegible]
[illegible]phasetest\033[C\033[C\033[C[illegible]
[illegible]out\033[K033[K033[K033[K033[K033[K033[K033[K033
[K033[K033[K033[K033[K033[K033[K033[K\007\007CPP shapes\033[K shapelib033[K033
[K033[K033[K033[K033[K033[K033[K033[K033[K033[K033[K033[K033[K033[K033[K
\033[K\007\007cat point3D.h
#ifndef POINT3D_H_INC
#define POINT3D_H_INC

#include <iostream>
#include <string>
#include <cmath>

using namespace std;

class point3D
{
    double x, y, z;
public:
    point3D(void) : x(0.0), y(0.0), z(0.0) { }
    point3D(double new_x, double new_y, double new_z) : x(0.0), y(0.0), z(0.0)
    {
        set_x(new_x);
        set_y(new_y);
        set_z(new_z);
    }

    void print(void);
    void read(void);

    double distance(point3D other);

    point3D operator+(point3D other)
    {
        return point3D(x+other.x, y+other.y, z+other.z);
    }
    point3D operator-(point3D other)
    {
        return point3D(x-other.x, y-other.y, z-other.z);
    }
    point3D operator*(double mult)
    {
        return point3D(x*mult, y*mult, z*mult);
    }

    double get_x(void) const { return x; }
    double get_y(void) const { return y; }
    double get_z(void) const { return z; }

    void set_x(double new_x)
    {
        x = new_x;
        return;
    }
    void set_y(double new_y)
    {
        y = new_y;
        return;
    }
    void set_z(double new_z)
    {
        z = new_z;
        return;
    }


};

inline point3D operator*(const double & m, const point3D & p)
{
    return point3D(p.get_x()*m, p.get_y()*m, p.get_z()*m);
}

#endif
\033]0;georgia@georgia-MT6017: ˜/cplusplus\007georgia@georgia-MT6017:˜/cplusplus$ c
ay\033[K\033[Kt point3D.cpp
#include "point3D.h"
#include <iostream>
#include <cmath>

using namespace std;

void point3D::print(void)
{
    cout << '(' << x << ", " << y << ", " << z << ')';
    return;
}

void point3D::read(void)
{
    char trash;
    cin >> trash >> x >> trash >> y >> trash >> z;
    return;
}

double point3D::distance(point3D other)
{
    return sqrt(pow(other.x-x, 2.0) +
                pow(other.y-y, 2.0) +
                pow(other.z-z, 2.0));
}
\033]0;georgia@georgia-MT6017: ˜/cplusplus\007georgia@georgia-MT6017:˜/cplusplus$ c
at shapelib.h
#ifndef SHAPELIB_H_INC
#define SHAPELIB_H_INC

#include <iostream>
#include <sstream>
#include <string>
#include <cmath>
#include "point3D.h"
#include "ccc_win.h"
#include <vector>


using namespace std;

const double PI = 3.141592653589793238462;


/* Rotates a point3D around the origin of the coordinate system. Takes arguments
point3D and x, y, z rotation angles. Returns point3D */

point3D rotate(point3D apoint, double x_rot, double y_rot, double z_rot)
{
    double x = apoint.get_x(), y = apoint.get_y(), z = apoint.get_z();
    double xradns = PI/180*x_rot;
    double yradns = PI/180*y_rot;
    double zradns = PI/180*z_rot;

    double rot_mat[3][3];
    rot_mat[0][0] = cos(yradns)*cos(zradns);
    rot_mat[0][1] = -(cos(xradns)*sin(zradns)) +
                     (sin(xradns)*sin(yradns)*cos(zradns));
    rot_mat[0][2] =  (sin(xradns)*sin(zradns)) +
                     (cos(xradns)*sin(yradns)*cos(zradns));
    rot_mat[1][0] =  cos(yradns)*sin(zradns);
    rot_mat[1][1] =  (cos(xradns)*cos(zradns)) +
                     (sin(xradns)*sin(yradns)*sin(zradns));
    rot_mat[1][2] = -(sin(xradns)*cos(zradns)) +
                     (cos(xradns)*sin(yradns)*sin(zradns));
    rot_mat[2][0] = -sin(yradns);
    rot_mat[2][1] =  sin(xradns)*cos(yradns);
    rot_mat[2][2] =  cos(xradns)*cos(yradns);


    return point3D((rot_mat[0][0]*x + rot_mat[0][1]*y + rot_mat[0][2]*z),
                   (rot_mat[1][0]*x + rot_mat[1][1]*y + rot_mat[1][2]*z),
                   (rot_mat[2][0]*x + rot_mat[2][1]*y + rot_mat[2][2]*z));
}



/* Projects a 3D point onto a 2D plane (ie screen) as if the z-axis is
prodtruding/receding at a given angle, defaults to 45 degrees. Takes arguments
point3D and optional z-axis projection angle. Returns 2D Point from the book's
graphics library. */

Point project(point3D apoint, double projAngle=45)
{
    double x = apoint.get_x(), y = apoint.get_y(), z = apoint.get_z();
    double rot_mat[3][3];
    rot_mat[0][0] = 1;
    rot_mat[0][1] = 0;
    rot_mat[0][2] = 0.5 * cos(PI/180 * projAngle);
    rot_mat[1][0] = 0;
    rot_mat[1][1] = 1;
    rot_mat[1][2] = 0.5 * sin(PI/180 * projAngle);
    rot_mat[2][0] = 0;
    rot_mat[2][1] = 0;
    rot_mat[2][2] = 0;


    return Point((rot_mat[0][0]*x + rot_mat[0][1]*y + rot_mat[0][2]*z),
                 (rot_mat[1][0]*x + rot_mat[1][1]*y + rot_mat[1][2]*z));
}


/* draws a handy little coordinate plane for reference */
void drawAxes(void)
{
    Point x1 = project(point3D(-10, 0, 0));
    Point x2 = project(point3D(10, 0, 0));
    Point y1 = project(point3D(0, -10, 0));
    Point y2 = project(point3D(0, 10, 0));
    Point z1 = project(point3D(0, 0, -10));
    Point z2 = project(point3D(0, 0, 10));
    cwin << Line(x1, x2);
    cwin << Message(x1, "(-10, 0, 0)");
    cwin << Message(x2, "(10, 0, 0)");
    cwin << Line(y1, y2);
    cwin << Message(y1, "(0, -10, 0)");
    cwin << Message(y2, "(0, 10, 0)");
    cwin << Line(z1, z2);
    cwin << Message(z1, "(0, 0, -10)");
    cwin << Message(z2, "(0, 0, 10)");
    return;
}

/* Base class for all 1, 2, and 3 dimensional shapes. Contains a single point3D
and a name. */

class shape
{
    point3D myPoint;
    string myName;

public:
    virtual ˜shape(void) { }

    shape(point3D p, const string & n = "") :
        myPoint(p), myName(n) { }

    virtual string tellName(void) const {  return myName; }

    point3D get_point(void) { return myPoint; }

    virtual void printInfo(void) = 0;

    virtual void draw(void) = 0;
};

/* One dimensional derived abstract class. */

class OneD : public shape
{
    string shapetype;

public:

    OneD(point3D one, const string & n = "",
                      const string & t = "One Dimensional")
    : shape(one, n), shapetype(t) { }

    virtual string tellType(void) const { return shapetype; }

};

/* Two dimensional derived abstract class */

class TwoD : public shape
{
    string shapetype;

public:

    TwoD(point3D one, const string & n = "",
                      const string & t = "Two Dimensional")
    : shape(one, n), shapetype(t) { }

    virtual string tellType(void) const { return shapetype; }


};

/* Three dimensional derived abstract class */

class ThreeD : public shape
{
    string shapetype;
public:
    ThreeD(point3D one, const string & n = "",
                        const string & t = "Three Dimensional")
    : shape(one, n), shapetype(t) { }

    virtual string tellType(void) const { return shapetype; }
};



class line : public OneD
{
    point3D endpoint;
public:
     line(point3D start, point3D end, const string & n ="Line")
     : OneD(start, n), endpoint(end) { }

     point3D get_start(void) { return get_point(); }

     point3D get_end(void) { return endpoint; }

     double get_length(void) { return get_point().distance(endpoint); }

     virtual void printInfo(void)
     {
         cout << "\nI’m a " << tellName() << ’!’;
         cout << "\nMy dimensionality is: " << tellType();
         cout << "\nFirst point is: ";
         get_start().print();
         cout << "\nSecond point is: ";
         get_end().print();
         cout << "\nMy length is: " << get_length() << "\n";

     }
};

// takes in 4 points that define a cubic bezier curve
class curve : public OneD
{
    point3D control1;
    point3D control2;
    point3D end;
    vector<point3D> pointvec;

    // populates the point vector with 100 points
    void fill_points(void)
    {
        double inc=0.01;
        point3D point1, point2, point3, point4, point5, point6;

        for (double i=inc; i<=1; i+=inc)
        {
            point1 = get_point() + (control1 - get_point()) * i;
            point2 = control1 + (control2 - control1) * i;
            point3 = control2 + (end - control2) * i;
            point4 = point1 + (point2 - point1) * i;
            point5 = point2 + (point3 - point2) * i;
            point6 = point4 + (point5 - point4) * i;
            pointvec.push_back(point6);
        }
        return;
    }

public:
    curve(point3D s, point3D c1, point3D c2,
          point3D e, const string & n="Curve")
    : OneD(s, n), control1(c1), control2(c2),
      end(e), pointvec(vector<point3D>())
    {
        fill_points();
    }
    void draw(void)
    {
        cwin << project(get_point())
             << project(control1)
             << project(control2)
             << project(end);
        for(short i=0; i<pointvec.size()-2; i++)
        {
            cwin << Line(project(pointvec[i]), project(pointvec[i+1]));
        }
        return;
    }
    double get_length(void)
    {
        double accumulator;
        for(short i; i<pointvec.size()-2; i++)
        {
            accumulator += pointvec[i].distance(pointvec[i+1]);
        }
        return accumulator;
    }
    void printInfo(void)
    {
        cout << "\nI’m a " << tellName() << ’!’;
        cout << "\nMy dimensionality is: " << tellType();
        get_point().print();
        control1.print();
        control2.print();
        end.print();
        cout << "\nMy length is: " << get_length();
        draw();
    }
};

// inherits point and adds radius and 3 rotation angles
class circle : public TwoD
{
    double radius, xrot, yrot, zrot;
    vector<point3D> pointvec;
public:
    circle(point3D one, double r, double x=0,
                        double y=0, double z=0, const string & n = "Circle")
    : TwoD(one, n), radius(r), xrot(x), yrot(y), zrot(z),
      pointvec(vector<point3D>())
    {
        createPoints();
    }

    double get_area(void) { return PI * pow(radius, 2);  }

    double get_circumference(void)  { return 2 * PI * radius; }

    void createPoints(void)
    {
        point3D radpoint(radius, 0, 0);

        for (double i=0; i<360; i+=1)
        {
            pointvec.push_back(rotate(radpoint, 0, 0, i));
        }
        for (short v=0; v<pointvec.size()-1; v++)
        {
            pointvec[v] = get_point()+rotate(pointvec[v], xrot, yrot, zrot);
        }
        return;
    }
    void draw(void)
    {
        cwin << Line(project(pointvec[0]), project(pointvec[180]));
        cwin << Line(project(pointvec[90]), project(pointvec[270]));

        for (short i=0; i<pointvec.size()-2; i++)
        {
            cwin << Line(project(pointvec[i]), project(pointvec[i+1]));
        }
        return;
    }

    virtual void printInfo(void)
    {

        cout << "\nI’m a " << tellName() << ’!’;
        cout << "\nMy dimensionality is: " << tellType();
        cout << "\nMy center is: ";
        get_point().print();
        cout << "\nMy radius is: " << radius;
        cout << "\nMy area is: " << get_area();
        cout << "\nMy circumference is: " << get_circumference() << ’\n’;
        draw();
        return;
    }
};

class square : public TwoD
{
    double sideLength;
public:
    square(point3D upperleft, double l, const string & n = "Square")
    : TwoD(upperleft, n), sideLength(l) { }

    double get_perimeter(void) { return sideLength * 4; }
    double get_area(void) { return pow(sideLength, 2); }

    virtual void printInfo(void)
    {
        cout << "\nI’m a " << tellName() << ’!’;
        cout << "\nMy Dimensionality is: " << tellType();
        cout << "\nMy upper left corner is: ";
        get_point().print();
        cout << "\nMy side length is: " << sideLength;
        cout << "\nMy perimeter is: " << get_perimeter();
        cout << "\nMy area is: " << get_area() << ’\n’;
    }
};



// inherits point as its center, and adds sidelengths and 3 rotation angles
class cube : public ThreeD
{
    double sidelength;
    double x_rot;
    double y_rot;
    double z_rot;
    point3D points[8];
public:
     cube(point3D center, double l, double xr = 0.0, double yr = 0.0,
                          double zr = 0.0, const string & n = "Cube")
    : ThreeD(center, n), sidelength(l), x_rot(xr), y_rot(yr), z_rot(zr)
    {
        fillPointArray();
    }

    void fillPointArray(void)
    {
        points[0] = get_point()+
                    rotate(point3D(-sidelength/2, sidelength/2, -sidelength/2),
                                                          x_rot, y_rot, z_rot);
        points[1] = get_point()+
                    rotate(point3D(-sidelength/2, sidelength/2, sidelength/2),
                                                         x_rot, y_rot, z_rot);

        points[2] = get_point()+
                    rotate(point3D(sidelength/2, sidelength/2, sidelength/2),
                                                        x_rot, y_rot, z_rot);

        points[3] = get_point()+
                    rotate(point3D(sidelength/2, sidelength/2, -sidelength/2),
                                                         x_rot, y_rot, z_rot);

        points[4] = get_point()+
                    rotate(point3D(-sidelength/2, -sidelength/2, -sidelength/2),
                                                           x_rot, y_rot, z_rot);

        points[5] = get_point()+
                    rotate(point3D(-sidelength/2, -sidelength/2, sidelength/2),
                                                          x_rot, y_rot, z_rot);

        points[6] = get_point()+
                    rotate(point3D(sidelength/2, -sidelength/2, sidelength/2),
                                                         x_rot, y_rot, z_rot);

        points[7] = get_point()+
                    rotate(point3D(sidelength/2, -sidelength/2, -sidelength/2),
                                                          x_rot, y_rot, z_rot);
        return;
    }

    point3D get_A(void) { return points[0]; }
    point3D get_B(void) { return points[1]; }
    point3D get_C(void) { return points[2]; }
    point3D get_D(void) { return points[3]; }
    point3D get_E(void) { return points[4]; }
    point3D get_F(void) { return points[5]; }
    point3D get_G(void) { return points[6]; }
    point3D get_H(void) { return points[7]; }

    virtual void draw(void)
    {
        cwin << project(get_point());
        cwin << Line(project(points[0]), project(points[7]));
        cwin << Line(project(points[3]), project(points[4]));

        cwin << Line(project(points[0]), project(points[1]));
        cwin << Line(project(points[1]), project(points[2]));
        cwin << Line(project(points[2]), project(points[3]));
        cwin << Line(project(points[3]), project(points[0]));
        cwin << Line(project(points[0]), project(points[4]));
        cwin << Line(project(points[1]), project(points[5]));
        cwin << Line(project(points[2]), project(points[6]));
        cwin << Line(project(points[3]), project(points[7]));
        cwin << Line(project(points[4]), project(points[5]));
        cwin << Line(project(points[5]), project(points[6]));
        cwin << Line(project(points[6]), project(points[7]));
        cwin << Line(project(points[4]), project(points[7]));
        return;
    }


    virtual void printInfo(void)
    {
        cout << "\nI'm a " << tellName() << '!';
        cout << "\nMy Dimensionality is: " << tellType();
        cout << "\nMy center is: ";
        get_point().print();
        cout << "\nMy side length is: " << sidelength;
        cout << "\nMy A point is: ";
        get_A().print();
        cout << "\nMy B point is: ";
        get_B().print();
        cout << "\nMy C point is: ";
        get_C().print();
        cout << "\nMy D point is: ";
        get_D().print();
        cout << "\nMy E point is: ";
        get_E().print();
        cout << "\nMy F point is: ";
        get_F().print();
        cout << "\nMy G point is: ";
        get_G().print();
        cout << "\nMy H point is: ";
        get_H().print();
        cout << '\n';
        draw();
        return;
    }

};



#endif
\033]0;georgia@georgia-MT6017: ~/cplusplus\007georgia@georgia-MT6017:~/cplusplus$ c
at shapes.cpp
#include <iostream>
#include "point3D.h"
#include "shapelib.h"
#include "ccc_win.h"

using namespace std;



int ccc_win_main(void)
{
    vector<point3D> pointvec;


    point3D point1(-15, 10, 0), point2(-11, 10, 0),
            point3(-7, 10, 0), point4(-3, 10, 0),
            point5(-15, 5, 0), point6(-11, 5, 0),
            point7(-7, 5, 0), point8(-3, 5, 0), point9(-3, 3, 0);

    shape * shptr[9];

    circle circlederived(point5, 1.5);
    circle circle2(point6, 1.5, 45, 0, 0);
    circle circle3(point7, 1.5, 0, 45, 0);
    circle circle4(point8, 1.5, 0, 0, 45);
    cube cubederived(point1, 2.0);
    cube cube2(point2, 2.0, 45, 0, 0);
    cube cube3(point3, 2.0, 0, 45, 0);
    cube cube4(point4, 2.0, 0, 0, 45);

    curve curvetest(point3D(4, 6, 0), point3D(3, 10, 0),
                    point3D(7, 6, 0), point3D(6, 10, 0));

    drawAxes();



    shptr[0] = &cubederived;
    shptr[1] = &cube2;
    shptr[2] = &cube3;
    shptr[3] = &cube4;
    shptr[4] = &circlederived;
    shptr[5] = &circle2;
    shptr[6] = &circle3;
    shptr[7] = &circle4;
    shptr[8] = &curvetest;


    for (short i = 0; i<9; i++)
    {
        shptr[i]->printInfo();
    }


    return 0;
}



\033]0;georgia@georgia-MT6017: ˜/cplusplus\007georgia@georgia-MT6017:˜/cplusplus$ C
RO33hA[e\033[K033[K033[Kpes shapelib point3D ccc_x11 ccc_shap
ccc_shap.cpp...
ccc_x11.cpp...
point3D.cpp...
shapes.cpp..*
In file included from ccc_x11.cpp:31:0:
In file included from ccc_win.h:36:0,
                 from shapelib.h:9,
                 from shapes.cpp:3:
In file included from shapes.cpp:3:0:
shapelib.h: In member function â\200\230virtual void curve::draw()â\200\231:
shapelib.h:236:42: warning: comparison between signed and unsigned
integer expressions [-Wsign-compare]
shapelib.h: In member function â\200\230double curve::get_length()â\200\231:
shapelib.h:245:40: warning: comparison between signed and unsigned
integer expressions [-Wsign-compare]
shapelib.h: In member function â\200\230void circle::createPoints()â\200\231:
shapelib.h:290:43: warning: comparison between signed and unsigned
integer expressions [-Wsign-compare]
shapelib.h: In member function â\200\230virtual void circle::draw()â\200\231:
shapelib.h:301:43: warning: comparison between signed and unsigned
integer expressions [-Wsign-compare]


\033]0;georgia@georgia-MT6017: ˜/cplusplus\007georgia@georgia-MT6017:˜/cplusplus$ .
/shapes.out

I’m a Cube!
My Dimensionality is: Three Dimensional
My center is: (-15, 10, 0)
My side length is: 2
My A point is: (-16, 11, -1)
My B point is: (-16, 11, 1)
My C point is: (-14, 11, 1)
My D point is: (-14, 11, -1)
My E point is: (-16, 9, -1)
My F point is: (-16, 9, 1)
My G point is: (-14, 9, 1)
My H point is: (-14, 9, -1)

I’m a Cube!
My Dimensionality is: Three Dimensional
My center is: (-11, 10, 0)
My side length is: 2
My A point is: (-12, 11.4142, -1.11022e-16)
My B point is: (-12, 10, 1.41421)
My C point is: (-10, 10, 1.41421)
My D point is: (-10, 11.4142, -1.11022e-16)
My E point is: (-12, 10, -1.41421)
My F point is: (-12, 8.58579, 1.11022e-16)
My G point is: (-10, 8.58579, 1.11022e-16)
My H point is: (-10, 10, -1.41421)

I’m a Cube!
My Dimensionality is: Three Dimensional
My center is: (-7, 10, 0)
My side length is: 2
My A point is: (-8.41421, 11, -1.11022e-16)
My B point is: (-7, 11, 1.41421)
My C point is: (-5.58579, 11, 1.11022e-16)
My D point is: (-7, 11, -1.41421)
My E point is: (-8.41421, 9, -1.11022e-16)
My F point is: (-7, 9, 1.41421)
My G point is: (-5.58579, 9, 1.11022e-16)
My H point is: (-7, 9, -1.41421)

I’m a Cube!
My Dimensionality is: Three Dimensional
My center is: (-3, 10, 0)
My side length is: 2
My A point is: (-4.41421, 10, -1)
My B point is: (-4.41421, 10, 1)
My C point is: (-3, 11.4142, 1)
My D point is: (-3, 11.4142, -1)
My E point is: (-3, 8.58579, -1)
My F point is: (-3, 8.58579, 1)
My G point is: (-1.58579, 10, 1)
My H point is: (-1.58579, 10, -1)

I’m a Circle!
My dimensionality is: Two Dimensional
My center is: (-15, 5, 0)
My radius is: 1.5
My area is: 7.06858
My circumference is: 9.42478

I’m a Circle!
My dimensionality is: Two Dimensional
My center is: (-11, 5, 0)
My radius is: 1.5
My area is: 7.06858
My circumference is: 9.42478

I’m a Circle!
My dimensionality is: Two Dimensional
My center is: (-7, 5, 0)
My radius is: 1.5
My area is: 7.06858
My circumference is: 9.42478

I’m a Circle!
My dimensionality is: Two Dimensional
My center is: (-3, 5, 0)
My radius is: 1.5
My area is: 7.06858
My circumference is: 9.42478

I’m a Curve!
My dimensionality is: One Dimensional(4, 6, 0)(3, 10, 0)(7, 6, 0)(6, 10, 0)
My length is: 15.3212\033]0;georgia@georgia-MT6017: ˜/cplusplus\007georgia@georgia-
MT6017:˜/cplusplus$ exit
exit

Script done on Sat 08 Dec 2012 10:01:38 PM CST
```