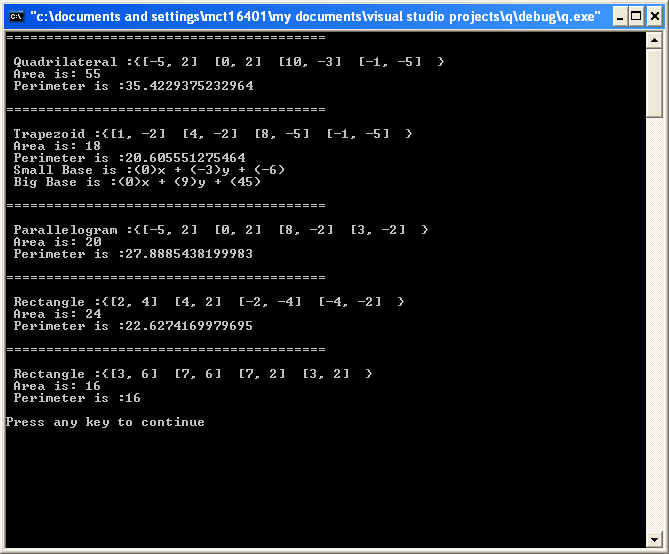
Object Oriented Programming

Homework 8

Xiufeng Cheng



**// Point.h**

// Point class represents an x-y coordinate pair.

#pragma once

#using <mscorlib.dll>

using namespace System;

// Point class definition implicitly inherits from Object

public \_\_gc class Point {

public:

Point(); // default constructor

Point( int, int ); // constructor

Point::~Point()

{

Console::WriteLine( S"Point4 destructor: {0}", this );

}

\_\_property int get\_X()

{

return x;

}

\_\_property void set\_X( int value )

{

x = value;

}

\_\_property int get\_Y()

{

return y;

}

\_\_property void set\_Y( int value )

{

y = value;

}

String \*ToString();

private:

int x, y; // point coordinates

}; // end class Point

**// Point.cpp**

// Member function definitions for class Point.

#include "Point.h"

// default (no-argument) constructor

Point::Point()

{

// implicit call to Object constructor occurs here

}

// constructor

Point::Point( int xValue, int yValue )

{

// implicit call to Object constructor occurs here

x = xValue;

y = yValue;

}

// return string representation of Point

String \*Point::ToString()

{

return String::Concat( S"[", x.ToString(),S", ", y.ToString(), S"] " );

}

**//line.h**

//Line class represents an line consist of two points.

#pragma once

#include "Point.h"

public \_\_gc class Line

{

public:

Line();// default constructor

Line(Point \* , Point \*);// constructor

\_\_property Point\* get\_P1(){

return p1;

}

\_\_property Point\* get\_P2(){

return p2;

}

\_\_property void set\_P1(Point \* value){

p1 = value;

}

\_\_property void set\_P2(Point \* value){

p2 = value;

}

double length();

double slope();

double ACoefficient();

double BCoefficient();

double CCoefficient();

String \*Line::ToString();

private:

Point \* p1;

Point \* p2;

};// end class Line

**//Line.cpp**

#include "Line.h"

#define MAX (999999);

Line::Line()

{

}

Line::Line(Point\* pointValue1, Point \*pointValue2)

{

P1 = pointValue1;

P2 = pointValue2;

}

double Line::slope()

{

if(P2->X - P1->X)

return (P1->Y - P1->Y)/(P2->X - P1->X);

else

return MAX;

}

double Line::length()

{

return Math::Sqrt((P1->X - P2->X) \* (P1->X - P2->X) + (P1->Y - P2->Y) \* (P1->Y - P2->Y));

}

double Line::ACoefficient()

{

return P2->Y - P1->Y;

}

double Line::BCoefficient()

{

return P1->X - P2->X;

}

double Line::CCoefficient()

{

return (P2->X)\*(P1->Y) - (P1->X)\*(P2->Y);

}

String \*Line::ToString()

{

return String::Concat(S"(",this->ACoefficient().ToString(),S")x + (",

this->BCoefficient().ToString(),S")y + (",

this->CCoefficient().ToString(),S")");

}

**// Quadrilateral.h**

#pragma once

#include "Line.h"

//Trapezoid,Parallelogram,Rectangle,Square will inheret to this class

public \_\_gc class Quadrilateral

{

public:

Quadrilateral();

Quadrilateral(Line \*, Line \*, Line \*, Line \*);

~Quadrilateral();

String \* ToString();

//Line properties

\_\_property Line \* get\_L1()

{

return l1;

}

\_\_property void set\_L1( Line \*value )

{

l1 = value;

}

\_\_property Line \* get\_L2()

{

return l2;

}

\_\_property void set\_L2( Line \*value )

{

l2 = value;

}

\_\_property Line \* get\_L3()

{

return l3;

}

\_\_property void set\_L3( Line \*value )

{

l3 = value;

}

\_\_property Line \* get\_L4()

{

return l4;

}

\_\_property void set\_L4( Line \*value )

{

l4 = value;

}

double distanceFromPointToPoint(Point \*p1,Point \*p2);

double calculatePerimeter();

double area();

protected:

Line \*l1;

Line \*l2;

Line \*l3;

Line \*l4;

}; // end class Quadrilateral

**//Quadrilateral.cpp**

#include "Quadrilateral.h"

Quadrilateral::Quadrilateral()

{

}

Quadrilateral::Quadrilateral(Line \*line1, Line \*line2, Line \*line3, Line \*line4)

{

L1 = line1;

L2 = line2;

L3 = line3;

L4 = line4;

}

Quadrilateral::~Quadrilateral()

{

}

/\*

Below is very important function, it calculate the area of a quadrilateral no matter what shape it is.

\*/

double Quadrilateral::area()

{

double area = 0;

double diagonal\_1 = this->distanceFromPointToPoint(L1->P1,L3->P1);

double diagonal\_2 = this->distanceFromPointToPoint(L2->P1,L4->P1);

double abcd = L1->length()\*L1->length()+L3->length()\*L3->length() - L2->length()\*L2->length() - L4->length()\*L4->length();

return 0.25 \* Math::Sqrt(4\*diagonal\_1 \* diagonal\_1 \* diagonal\_2 \* diagonal\_2 -abcd\*abcd);

}

double Quadrilateral::distanceFromPointToPoint(Point \*p1,Point \*p2)

{ double length = 0;

length = Math::Sqrt((p1->X - p2->X) \* (p1->X - p2->X) + (p1->Y - p2->Y) \* (p1->Y - p2->Y));

return length;

}

double Quadrilateral::calculatePerimeter()

{

return L1->length()+L2->length()+L3->length()+L4->length();

}

String \* Quadrilateral::ToString()

{

return String::Concat(S"{",L1->P1->ToString(),L2->P1->ToString(),L3->P1->ToString(),L4->P1->ToString(),

S"}",S"\n Area is: ",this->area().ToString(),S"\n Perimeter is :", this->calculatePerimeter().ToString(),S"\n");

}

**// Trapezoid.h**

#pragma once

#include "Quadrilateral.h"

// Inherits from Quadrilateral

public \_\_gc class Trapezoid : public Quadrilateral

{

public:

Trapezoid();

Trapezoid(Line \*, Line \*, Line \*, Line \*);

~Trapezoid();

String \* ToString();

private:

bool isParallel(Line \*, Line \*);

Line \* findSmallBase();

Line \* findOtherBase();

}; // end class Trapezoid

**//Trapezoid.cpp**

#include "Trapezoid.h"

Trapezoid::Trapezoid()

{

}

Trapezoid::Trapezoid(Line \*line1, Line \*line2, Line \*line3, Line \*line4)

{

L1 = line1;

L2 = line2;

L3 = line3;

L4 = line4;

}

Trapezoid::~Trapezoid()

{

}

bool Trapezoid::isParallel(Line \* line1, Line \* line2)

{

if (line1->slope() == line2->slope())

return true;

else

return false;

}

Line \* Trapezoid::findSmallBase()

{

if ( isParallel(L1, L3) ) // function isParallel is used here

return (L1->length() <= L3->length()) ? L1 : L3;

else

return (L2->length() <= L4->length()) ? L2 : L4;

}

Line \* Trapezoid::findOtherBase()

{

if ( isParallel(L1, L3) ) // function isParallel is used here

return (L1->length() >= L3->length()) ? L1 : L3;

else

return (L2->length() >= L4->length()) ? L2 : L4;

}

String \* Trapezoid::ToString()

{

return String::Concat(S" Trapezoid :" , \_\_super::ToString(),

S" Small Base is :",this->findSmallBase()->ToString(),

S"\n Big Base is :",this->findOtherBase()->ToString());

}

**// Parallelogram.h**

#pragma once

#include "Trapezoid.h"

// Inherits from Trapezoid

public \_\_gc class Parallelogram : public Trapezoid

{

public:

Parallelogram();

Parallelogram(Line \*, Line \*, Line \*, Line \*);

~Parallelogram();

String \* ToString();

}; // end class Praallelogram

**//Parallelogram.cpp**

#include "Parallelogram.h"

Parallelogram::Parallelogram()

{

}

Parallelogram::Parallelogram(Line \*line1, Line \*line2, Line \*line3, Line \*line4)

{

L1 = line1;

L2 = line2;

L3 = line3;

L4 = line4;

}

Parallelogram::~Parallelogram()

{

}

String \* Parallelogram::ToString()

{

return String::Concat(S"\n Parallelogram :" , Quadrilateral::ToString());

}

**// Rectangle.h**

#pragma once

#include "Parallelogram.h"

public \_\_gc class Rectangle : public Parallelogram

{

public:

Rectangle();

~Rectangle();

Rectangle(Line \*, Line \*, Line \*, Line \*);

String \* ToString();

}; // end class Rectangle

**//Rectangle.cpp**

#include "Rectangle.h"

Rectangle::Rectangle()

{

}

Rectangle::Rectangle(Line \*line1, Line \*line2, Line \*line3, Line \*line4)

{

L1 = line1;

L2 = line2;

L3 = line3;

L4 = line4;

}

Rectangle::~Rectangle()

{

}

String \* Rectangle::ToString()

{

return String::Concat(S"\n Rectangle :" , Quadrilateral::ToString());

}

**// Square.h**

#pragma once

#include "Rectangle.h"

public \_\_gc class Square : public Rectangle

{

public:

Square();

~Square();

Square(Line \*, Line \*, Line \*, Line \*);

String \* ToString();

}; // end class Square

**//Square.cpp**

#include "Square.h"

Square::Square()

{

}

Square::Square(Line \*line1, Line \*line2, Line \*line3, Line \*line4)

{

L1 = line1;

L2 = line2;

L3 = line3;

L4 = line4;

}

Square::~Square()

{

}

String \* Square::ToString()

{

return String::Concat(S"\n Rectangle :" ,Quadrilateral::ToString());

}