# Лабораторная работа 3. Виртуальные функции в С++

### Задание

Для каждого типа фигуры(Треугольник, Квадрат) вычисляются площадь, центр тяжести, радиусы вписанных или описанных окружностей.

Операции над объектами этих классов могут быть такими (функции с 2-мя аргументами Shape\* obj1, Shape\* obj2):

* сравнить два объекта по площади - Compare;
* определить факт пересечения объектов – IsIntersect;
* определить факт включения одного объекта в другой – IsInclude.

### Диаграмма классов

Operation

Shape

Point

Line

Triangle

Square

Hexagon

Rectangle

Octagon

Parallelogram

Trapeze

### Перечень сигнатур методов с комментариями согласно заданию

Point.h

#pragma once

class Point

{

public:

double x;

double y;

Point(void);

Point(double x, double y);

~Point(void);

};

Line.h

#include "Point.h"

#include <algorithm>

#pragma once

class Line

{

private:

Point A;

Point B;

double Get\_k();//получение соответственного коэффициэнта уравнения прямой y=kx+b

double Get\_b();//получение соответственного коэффициэнта уравнения прямой y=kx+b

public:

double GetLength();//получение длины отрезка

Point GetCentralPoint();//получение точки центра отрезка

Point GetCrossoverPoint(Line line);//получение точки пересечения для 2х отрезков

Line(Point A\_, Point B\_);

~Line(void);

};

Shape.h

#include "Point.h"

#pragma once

class Shape

{

protected:

Point\* arc;

public:

char id;

virtual double GetArea() = 0;//вычисление площади

virtual Point GetCenterOfGravity() = 0;//вычисление центра тяжести

virtual double GetInscribedRadius() = 0;//вычисление вписанной окружности

virtual double GetCircumscribedRadius() = 0;//вычисление описанной окружности

Shape(void);

virtual ~Shape(void);

};

Triangle.h

#pragma once

#include "shape.h"

#include "Line.h"

class Triangle :

public Shape

{

private:

void Init();

double GetPerimeter() const;//вычисление периметра

public:

double GetArea();

Point GetCenterOfGravity();

double GetInscribedRadius();

double GetCircumscribedRadius();

Triangle(void);

Triangle(Point p1, Point p2, Point p3);

~Triangle(void);

};

#pragma once

#include "shape.h"

#include "Line.h"

#include <algorithm>

class Square :

public Shape

{

private:

void Init();

double GetA() const;//получить длину стороны квадрата

public:

double GetArea();

Point GetCenterOfGravity();

double GetInscribedRadius();

double GetCircumscribedRadius();

operator bool() const;

Square(Point A, Point B, Point C, Point D);

Square(void);

~Square(void);

};

Operation.h

#pragma once

#include "Shape.h"

#include "Line.h"

#include "Point.h"

#include "Triangle.h"

#include <iostream>

using namespace std;

class Operation

{

public:

void Compare(Shape\* s1, Shape\* s2) const;//сравнение по площади

void IsInclude(Shape\* s1, Shape\* s2) const;//включение одной из фигур другую

void IsIntersect(Shape\* s1, Shape\* s2) const;//пересечение двух фигур

Operation(void);

~Operation(void);

};

### Листинг

Point.cpp

#include "Point.h"

Point::Point(void) { }

Point::Point(double x\_, double y\_)

{

x = x\_;

y = y\_;

}

Point::~Point(void) { }

Line.cpp

#include "Line.h"

Line::Line(Point A\_, Point B\_)

{

A = A\_;

B = B\_;

}

double Line::GetLength()

{

return sqrt((A.x - B.x)\*(A.x - B.x) + (A.y - B.y)\*(A.y - B.y));

}

Point Line::GetCentralPoint()

{

return Point((A.x + B.x) / 2, (A.y + B.y) / 2);

}

Point Line::GetCrossoverPoint(Line line)

{

double a = -line.Get\_k(), b = line.A.x == line.B.x ? 0 : 1, c = -line.Get\_b();

double a1 = -this->Get\_k(), b1 = A.x == B.x ? 0 : 1, c1 = -this->Get\_b();

if ((a \* a1 + b \* b1) == 0)

{

return Point(-c1, -c);

}

else

{

return Point((c1 \* b - c \* b1) / (a \* b1 - a1 \* b), (c1 \* a - c \* a1) / (b \* a1 - b1 \* a));

}

}

double Line::Get\_b()

{

return (A.y - Get\_k() \* A.x);

}

double Line::Get\_k()

{

if (A.x - B.x != 0)

{

if (A.y != B.y)

{

return (A.y - B.y) / (A.x - B.x);

}

else

{

return 0;

}

}

else

{

return 0;

}

}

Line::~Line(void) { }

Shape.cpp

#include "Shape.h"

Shape::Shape(void) { }

Shape::~Shape(void) { }

Triangle.cpp

#include "Triangle.h"

#include "Line.h"

#include <algorithm>

using namespace std;

void Triangle::Init()

{

arc = new Point[3];

id = 'T';

}

Triangle::Triangle(void)

{

Init();

}

Triangle::Triangle(Point p1, Point p2, Point p3)

{

Init();

arc[0] = p1;

arc[1] = p2;

arc[2] = p3;

}

Triangle::~Triangle(void)

{

delete []arc;

}

double Triangle::GetArea()

{

Line a (arc[0], arc[1]), b (arc[0], arc[2]), c (arc[1], arc[2]);

double p = (a.GetLength() + b.GetLength() + c.GetLength()) / 2;

double S = sqrt(p \* (p - a.GetLength()) \* (p - b.GetLength()) \* (p -c.GetLength()));

return S;

}

Point Triangle::GetCenterOfGravity()

{

Line l1(arc[0], arc[1]);

l1 = Line(l1.GetCentralPoint(), arc[2]);

Line l2(arc[0], arc[2]);

l2 = Line(l2.GetCentralPoint(), arc[1]);

return l1.GetCrossoverPoint(l2);

}

double Triangle::GetInscribedRadius()

{

return GetArea() / GetPerimeter();

}

double Triangle::GetCircumscribedRadius()

{

Line a (arc[0], arc[1]), b (arc[0], arc[2]), c (arc[1], arc[2]);

return (a.GetLength() \* b.GetLength() \* c.GetLength()) / (4 \* GetArea());

}

double Triangle::GetPerimeter() const

{

Line a (arc[0], arc[1]), b (arc[0], arc[2]), c (arc[1], arc[2]);

return (a.GetLength() + b.GetLength() + c.GetLength()) / 2;

}

Square.h

#include "Square.h"

void Square::Init()

{

arc = new Point[4];

id = 'S';

}

Square::Square(Point A, Point B, Point C, Point D)

{

Init();

arc[0] = A;

arc[1] = B;

arc[2] = C;

arc[3] = D;

}

Square::operator bool() const

{

Line d1(arc[0], arc[2]), d2(arc[1], arc[3]);

if (d1.GetLength() == d2.GetLength())

{

return true;

}

else

{

return false;

}

}

Square::Square(void)

{

Init();

}

double Square::GetArea()

{

double a = this->GetA();

return a \* a;

}

Point Square::GetCenterOfGravity()

{

Line line(arc[0], arc[2]);

return line.GetCentralPoint();

}

double Square::GetInscribedRadius()

{

return this->GetA() / 2;

}

double Square::GetCircumscribedRadius()

{

Line line(arc[0], arc[1]);

return line.GetLength() / 2;

}

double Square::GetA() const

{

Line line(arc[0], arc[1]);

return line.GetLength() / sqrt(2);

}

Square::~Square(void)

{

delete []arc;

}

Operation.cpp

#include "Operation.h"

Operation::Operation(void) { }

Operation::~Operation(void) { }

void Operation::Compare(Shape\* s1, Shape\* s2) const//сравнение фигур следующим условием по их площадям s1 < s2

{

//Проверка правильности подбора типов фигур

if((s1->id == 'T' && s2->id == 'S') || (s2->id == 'T' && s1->id == 'S'))

{

char sign;

if(s1->GetArea() < s2->GetArea() && abs(s1->GetArea() - s2->GetArea()) > 0.0001)

{

sign = '<';

}

else if(s1->GetArea() > s2->GetArea() && abs(s1->GetArea() - s2->GetArea()) > 0.0001)

{

sign = '>';

}

else

{

sign = '=';

}

std::cout << "Area " << s1->id << " " << sign << " " << s2->id << std::endl;

}

else

{

std::cout << "Not such operation" << std::endl;

}

}

void Operation::IsInclude(Shape\* s1, Shape\* s2) const

{

//Проверка правильности подбора типов фигур

if((s1->id == 'T' && s2->id == 'S') || (s2->id == 'T' && s1->id == 'S'))

{

Shape \*triagle = s1->id == 'T' ? s1 : s2, \*square = s1->id == 'S' ? s1 : s2;

Line distanceLine(triagle->GetCenterOfGravity(), square->GetCenterOfGravity());

double distance = distanceLine.GetLength();

double rT = triagle->GetInscribedRadius(), rS = square->GetInscribedRadius();

if (distance < abs(rS - rT))

{

cout << "Shape " << (s1->GetArea() > s2->GetArea() ? s1->id : s2->id) << " is include " << (s1->GetArea() > s2->GetArea() ? s2->id : s1->id) << endl;

}

else

{

cout << "Shape " << s1->id << " and " << s2->id << " is not include each other!" << endl;

}

}

else

{

std::cout << "Not such operation" << std::endl;

}

}

void Operation::IsIntersect(Shape\* s1, Shape\* s2) const

{

//Проверка правильности подбора типов фигур

if((s1->id == 'T' && s2->id == 'S') || (s2->id == 'T' && s1->id == 'S'))

{

Shape \*triagle = s1->id == 'T' ? s1 : s2, \*square = s1->id == 'S' ? s1 : s2;

Line distanceLine(triagle->GetCenterOfGravity(), square->GetCenterOfGravity());

double RT = triagle->GetCircumscribedRadius(), RS = square->GetCircumscribedRadius(), distance = distanceLine.GetLength();

double rT = triagle->GetInscribedRadius(), rS = square->GetInscribedRadius();

cout << "Shape " << s1->id << " and " << s2->id << " is " << (distance <= RT + RS && distance >= abs(rS - rT) ? ("") : ("not ")) << "intersect!" << endl;

}

else

{

std::cout << "Not such operation" << std::endl;

}

}

#pragma once

#include "Hexagon.h"

#include "Octagon.h"

#include "Parallelogram.h"

#include "Rectangle.h"

#include "Square.h"

#include "Triangle.h"

#include "Trapeze.h"

#include "Shape.h"

#include <iostream>

class FactoryShape

{

public:

Shape\* Generator();

Shape\* Choice();

};

#include "FactoryShape.h"

Shape\* FactoryShape::Generator()

{

switch (rand() % 7)

{

case 0:

return new Hexagon();

case 1:

return new Octagon();

case 2:

return new Parallelogram();

case 3:

return new Rectangle();

case 4:

return new Square(Point(0, 0), Point(0, 4), Point(4, 4), Point(4, 0));

case 5:

return new Trapeze();

case 6:

return new Triangle(Point(1, 1), Point(1, 3), Point(2, 3));

default:

break;

}

}

Shape\* FactoryShape::Choice()

{

char \_continue = 'y';

Shape\* tmp = 0;

while (\_continue == 'y')

{

tmp = Generator();

std::cout << "This shape is " << tmp->id << std::endl;

std::cout << "Continue choice? (y/...)" << std::endl;

std::cin >> \_continue;

}

return tmp;

}

**Все остальные фигуры:**

#pragma once

#include "shape.h"

class Hexagon :

public Shape

{

public:

virtual double GetArea() { return 0; };

virtual Point GetCenterOfGravity() { return Point(0, 0); };

virtual double GetInscribedRadius() { return 0; };

virtual double GetCircumscribedRadius() { return 0; };

Hexagon(void){ id = 'H'; }

~Hexagon(void);

};

class Octagon :

public Shape

{

public:

virtual double GetArea() { return 0; };

virtual Point GetCenterOfGravity() { return Point(0, 0); };

virtual double GetInscribedRadius() { return 0; };

virtual double GetCircumscribedRadius() { return 0; };

Octagon(void){ id = 'O'; }

~Octagon(void);

};

class Parallelogram :

public Shape

{

public:

virtual double GetArea() { return 0; };

virtual Point GetCenterOfGravity() { return Point(0, 0); };

virtual double GetInscribedRadius() { return 0; };

virtual double GetCircumscribedRadius() { return 0; };

Parallelogram(void){ id = 'P'; }

~Parallelogram(void);

};

class Rectangle :

public Shape

{

public:

virtual double GetArea() { return 0; };

virtual Point GetCenterOfGravity() { return Point(0, 0); };

virtual double GetInscribedRadius() { return 0; };

virtual double GetCircumscribedRadius() { return 0; };

Rectangle(void){ id = 'R'; }

~Rectangle(void);

};

class Trapeze :

public Shape

{

public:

virtual double GetArea() { return 0; };

virtual Point GetCenterOfGravity() { return Point(0, 0); };

virtual double GetInscribedRadius() { return 0; };

virtual double GetCircumscribedRadius() { return 0; };

Trapeze(void) { id = 'Z'; }

~Trapeze(void);

};

#include "Operation.h"

#include "FactoryShape.h"

int main()

{

Shape \*shape1, \*shape2;//создание 2х указателей на базовый класс

FactoryShape f;//создание фабрики фигур

Operation oper;//создание экземпляра класса для операций над фигурами

shape1 = f.Choice();//выбор 1й фигуры

shape2 = f.Choice();//выбор 2й фигуры

oper.Compare(shape1, shape2);//выполнение операции сравнения

oper.IsIntersect(shape2, shape1);//проверка пересечения фигур

oper.IsInclude(shape2, shape1);//проверка включения фигур друг в друга

return 0;

}

Знание кода Жиденко Н.Н. Отметки

Контрольные вопросы Жиденко Н.Н. Отметки

Автор Преподаватель

Жиденко Н.Н. \_\_\_\_\_\_\_\_\_\_\_\_\_\_ (подпись) Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Подпись \_\_\_\_\_\_\_\_ Дата