**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 3**

Тема: Наследование,полиморфизм

Студент: Трофимов Максим

Группа: 80-201

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2019

1. Постановка задачи

Разработать классы выпуклых много угольников, квадрата, прямоугольника и трапеции, классы должны наследоваться от базового класса Figure. Все классы должны поддерживать набор общих методов:

1.       Вычисление геометрического центра фигуры;

2.       Вывод в стандартный поток вывода std::cout координат вершин фигуры;

3. Вычисление площади фигуры;

Создать программу, которая позволяет:

•       Вводить из стандартного ввода std::cin фигуры, согласно варианту задания.

•       Сохранять созданные фигуры в динамический массив std::vector<Figure\*>

•       Вызывать для всего массива общие функции (1-3 см. выше).Т.е. распечатывать для каждой фигуры в массиве геометрический центр, координаты вершин и площадь.

•       Необходимо уметь вычислять общую площадь фигур в массиве.

•       Удалять из массива фигуру по индексу;

1. Описание программы

Программа распечатывает меню команд, которые позволяют повторно

* + - * распечатать меню
      * ввести одну из 4х фигур( выпуклый n-угольник, n>0 и задаётся пользователем, квадрат и треугольник, которые можно построить по трём или четырём точкам, и трапецию, которая задаётся тремя точками только в том случае, когда основания параллельны оси ОХ, или по четырём точкам. )
      * вывести информацию обо всех фигурах или об одной
      * печать суммы площадей этих фигур(без учёта их пересечения)
      * удаление фигуры

ссылка на github: <https://github.com/student31415/oop_exercise_03>

1. Набор testcases

|  |  |
| --- | --- |
| Тест | Примичание |
| 2  1  5  0.0 0.0 1.0 0.0 1.0 1.0 0.5 0.5 0.0 1.0  2  2  3  0.0 0.0 1.0 0.0 1.0 2.0  2  3  3  0.0 0.0 1.0 0.0 2.0 0.0  2  4  4  0.0 0.0 1.0 0.0 2.0 0.0 3.0 0.0  3  0 | Проверка на непропуск точек не принадлежащих фигурам варианта. |
| 2  1  5  0.0 0.0 1.0 0.0 2.0 1.0 0.5 3.0 -1.0 1.0  2  2  3  0.0 0.0 -1.0 0.0 -1.0 -1.0  2  3  3  0.0 0.0 1.0 0.0 1.0 2.0  2  4  3  0.0 0.0 1.0 1.0 2.0 1.0  3  5  6  3  3  0 | Проверка работопособности программы  ввод всех 4х фигур и проверка операций над ними |

1. Результаты выполнения тестов.

**test\_01.txt:**

1. Print menu.

2. Input figure.

3. Print information of all figures.

4. Print information of figure.

5. Print sum of squres of all figures.

6. Remove figure with index.

0. Exit.

>>2

Choose figure:

1. Polygn

2. Square

3. Rectangle

4. Trapeze

>>1

input number of polygon vertex.

>>5

Input vertex x and y coordinates.

>>0.0 0.0

Input vertex x and y coordinates.

>>1.0 0.0

Input vertex x and y coordinates.

>>1.0 1.0

Input vertex x and y coordinates.

>>0.5 0.5

Input vertex x and y coordinates.

>>0.0 1.0

Inputed coordinates are not belong convex polygon.

>>2

Choose figure:

1. Polygn

2. Square

3. Rectangle

4. Trapeze

>>2

Input number of square vertex (3 or 4).

>>3

Input vertex x and y coordinates.

>>0.0 0.0

Input vertex x and y coordinates.

>>1.0 0.0

Input vertex x and y coordinates.

>>1.0 2.0

Inputed coordinates are not belong convex square.

>>2

Choose figure:

1. Polygn

2. Square

3. Rectangle

4. Trapeze

>>3

Input number of rectangle vertex (3 or 4).

>>3

Input vertex x and y coordinates.

>>0.0 0.0

Input vertex x and y coordinates.

>>1.0 0.0

Input vertex x and y coordinates.

>>2.0 0.0

Inputed coordinates are not belong convex rectangle.

>>2

Choose figure:

1. Polygn

2. Square

3. Rectangle

4. Trapeze

>>4

Input number of trapeze vertex (3, if base parallel to axis OX, or 4).

>>4

Input vertex x and y coordinates.

>>0.0 0.0

Input vertex x and y coordinates.

>>1.0 0.0

Input vertex x and y coordinates.

>>2.0 0.0

Input vertex x and y coordinates.

>>3.0 0.0

Inputed coordinates are not belong convex trapeze.

>>3

There is no figures

>>0

**test\_02.txt**

1. Print menu.

2. Input figure.

3. Print information of all figures.

4. Print information of figure.

5. Print sum of squres of all figures.

6. Remove figure with index.

0. Exit.

>>2

Choose figure:

1. Polygn

2. Square

3. Rectangle

4. Trapeze

>>1

input number of polygon vertex.

>>5

Input vertex x and y coordinates.

>>0.0 0.0

Input vertex x and y coordinates.

>>1.0 0.0

Input vertex x and y coordinates.

>>2.0 1.0

Input vertex x and y coordinates.

>>0.5 3.0

Input vertex x and y coordinates.

>>-1.0 1.0

>>2

Choose figure:

1. Polygn

2. Square

3. Rectangle

4. Trapeze

>>2

Input number of square vertex (3 or 4).

>>3

Input vertex x and y coordinates.

>>0.0 0.0

Input vertex x and y coordinates.

>>-1.0 0.0

Input vertex x and y coordinates.

>>-1.0 -1.0

>>2

Choose figure:

1. Polygn

2. Square

3. Rectangle

4. Trapeze

>>3

Input number of rectangle vertex (3 or 4).

>>3

Input vertex x and y coordinates.

>>0.0 0.0

Input vertex x and y coordinates.

>>1.0 0.0

Input vertex x and y coordinates.

>>1.0 2.0

>>2

Choose figure:

1. Polygn

2. Square

3. Rectangle

4. Trapeze

>>4

Input number of trapeze vertex (3, if base parallel to axis OX, or 4).

>>3

Input vertex x and y coordinates.

>>0.0 0.0

Input vertex x and y coordinates.

>>1.0 1.0

Input vertex x and y coordinates.

>>2.0 1.0

>>3

figures:

1.Name: 5-gon

Center: ( 0.5 , 1 )

Square: 5

Coordinates:

( 0 , 0 )

( 1 , 0 )

( 2 , 1 )

( 0.5 , 3 )

( -1 , 1 )

2.Name: Square

Center: ( -0.5 , -0.5 )

Square: 1

Coordinates:

( 0 , 0 )

( -1 , 0 )

( -1 , -1 )

( 0 , -1 )

3.Name: Rectangle

Center: ( 0.5 , 1 )

Square: 2

Coordinates:

( 0 , 0 )

( 1 , 0 )

( 1 , 2 )

( 0 , 2 )

4.Name: Trapeze

Center: ( 1.5 , 0.5 )

Square: 2

Coordinates:

( 0 , 0 )

( 1 , 1 )

( 2 , 1 )

( 3 , 0 )

>>5

10

>>6

Input index of figure

>>3

>>3

figures:

1.Name: 5-gon

Center: ( 0.5 , 1 )

Square: 5

Coordinates:

( 0 , 0 )

( 1 , 0 )

( 2 , 1 )

( 0.5 , 3 )

( -1 , 1 )

2.Name: Square

Center: ( -0.5 , -0.5 )

Square: 1

Coordinates:

( 0 , 0 )

( -1 , 0 )

( -1 , -1 )

( 0 , -1 )

3.Name: Rectangle

Center: ( 0.5 , 1 )

Square: 2

Coordinates:

( 0 , 0 )

( 1 , 0 )

( 1 , 2 )

( 0 , 2 )

>>0

1. Листинг программы

**Figures.cpp**

//#ifndef \_\_fff\_\_

//#define \_\_fff\_\_

#include<iostream>

#include<vector>

#include<cmath>

#include<string>

//#endif

struct point{

double x=0.0,y=0.0;

};

class Figure{

public:

double square(){

if(points.size()==0) return 0.0;

double res=0.0;

for(int i=2; i<points.size(); ++i){

res+=square(points[0], points[i-1], points[i]);

}

}

point center(){

point cntr;

if(points.size()==0) return cntr;

for(int i=0; i<points.size(); ++i){

cntr.x+=points[i].x;

cntr.y+=points[i].y;

}

cntr.x/=points.size();

cntr.y/=points.size();

return cntr;

}

Figure(std::vector<point> pnts){

if(pnts.size()>2){

//Алгоритм Джарвис ?

//ищем самую нижнюю и левую точку(0-вую). Она точно принадлежит выпуклой оболочке.

for(int i=1; i<pnts.size(); ++i)

if(pnts[0].y>pnts[i].y || ( pnts[0].y==pnts[i].y && pnts[0].x>pnts[i].x))

std::swap(pnts[0], pnts[i]);

//кладем её в массив, где будет лежать наш ответ.

points.push\_back(pnts[0]);

//ищем первую точку по принципу: угол между вектором через 0-вую и 1-вую точки и вектором (1, 0) минимален,

//а расстояние между точками минимально.

point X; X.x=1.0; X.y=0.0;

point O; O.x=0; O.y=0;

for(int i=2; i<pnts.size(); ++i)

if(cos\_points(pnts[0], pnts[1], O, X) < cos\_points(pnts[0], pnts[i], O, X) || ( cos\_points(pnts[0], pnts[1], O, X)==cos\_points(pnts[0], pnts[i], O, X) && size\_points(pnts[0], pnts[1]) > size\_points(pnts[0], pnts[1])))

std::swap(pnts[1], pnts[i]);

points.push\_back(pnts[1]);

int k=2;

while(1){

for(int i=k+1; i<pnts.size(); ++i)

if(cos\_points(pnts[k-2], pnts[k-1],pnts[k-1], pnts[k]) < cos\_points(pnts[k-2], pnts[k-1],pnts[k-1], pnts[i]) || ( cos\_points(pnts[k-2], pnts[k-1],pnts[k-1], pnts[k]) == cos\_points(pnts[k-2], pnts[k-1],pnts[k-1], pnts[i]) && size\_points(pnts[k-1], pnts[k]) > size\_points(pnts[k-1], pnts[i])))

std::swap(pnts[k], pnts[i]);

if(cos\_points(pnts[k-2], pnts[k-1],pnts[k-1], pnts[k]) < cos\_points(pnts[k-2], pnts[k-1],pnts[k-1], pnts[0]) || ( cos\_points(pnts[k-2], pnts[k-1],pnts[k-1], pnts[k]) == cos\_points(pnts[k-1], pnts[k],pnts[k-1], pnts[0]) && size\_points(pnts[k-1], pnts[k]) > size\_points(pnts[k-1], pnts[0]))) break;

points.push\_back(pnts[k]);

++k;

if(k==pnts.size()) break;

}

//=====================================================================================================================================================

//если не все точки принадлежат выпуклой оболочке, то полностью очищаем итоговый массив.

if(points.size()<pnts.size()) while(points.size()) points.pop\_back(); else{ name+=points.size()+'0'; name+="-gon";}

}

}

bool initialized(){

if(points.size()>0) return true;

return false;

}

virtual void reinitialize(std::vector<point> pnts){

if(points.size()){

while(points.size()) points.pop\_back();

name="";

}

if(pnts.size()>2){

//Алгоритм Джарвис ?

//ищем самую нижнюю и левую точку(0-вую). Она точно принадлежит выпуклой оболочке.

for(int i=1; i<pnts.size(); ++i)

if(pnts[0].y>pnts[i].y || ( pnts[0].y==pnts[i].y && pnts[0].x>pnts[i].x))

std::swap(pnts[0], pnts[i]);

//кладем её в массив, где будет лежать наш ответ.

points.push\_back(pnts[0]);

//ищем первую точку по принципу: угол между вектором через 0-вую и 1-вую точки и вектором (1, 0) минимален,

//а расстояние между точками минимально.

point X; X.x=1.0; X.y=0.0;

point O; O.x=0; O.y=0;

for(int i=2; i<pnts.size(); ++i)

if(cos\_points(pnts[0], pnts[1], O, X) < cos\_points(pnts[0], pnts[i], O, X) || ( cos\_points(pnts[0], pnts[1], O, X)==cos\_points(pnts[0], pnts[i], O, X) && size\_points(pnts[0], pnts[1]) > size\_points(pnts[0], pnts[1])))

std::swap(pnts[1], pnts[i]);

points.push\_back(pnts[1]);

int k=2;

while(1){

for(int i=k+1; i<pnts.size(); ++i)

if(cos\_points(pnts[k-2], pnts[k-1],pnts[k-1], pnts[k]) < cos\_points(pnts[k-1], pnts[k],pnts[k-1], pnts[i]) || ( cos\_points(pnts[k-2], pnts[k-1],pnts[k-1], pnts[k]) == cos\_points(pnts[k-1], pnts[k],pnts[k-1], pnts[i]) && size\_points(pnts[k-1], pnts[k]) > size\_points(pnts[k-1], pnts[i])))

std::swap(pnts[k], pnts[i]);

if(cos\_points(pnts[k-2], pnts[k-1],pnts[k-1], pnts[k]) < cos\_points(pnts[k-1], pnts[k],pnts[k-1], pnts[0]) || ( cos\_points(pnts[k-2], pnts[k-1],pnts[k-1], pnts[k]) == cos\_points(pnts[k-1], pnts[k],pnts[k-1], pnts[0]) && size\_points(pnts[k-1], pnts[k]) > size\_points(pnts[k-1], pnts[0]))) break;

points.push\_back(pnts[k]);

++k;

if(k==pnts.size()) break;

}

//=====================================================================================================================================================

//если не все точки принадлежат выпуклой оболочке, то полностью очищаем итоговый массив.

if(points.size()<pnts.size()) while(points.size()) points.pop\_back(); else{ name+=points.size(); name+="-угольник";}

}

}

//std::ostream operator << (std::ostream& stream){

void print\_info(){

if(this->initialized()){

std::cout<<"Name: "<<name<<std::endl;

point cntr=this->center();

std::cout<<"Center: ( "<<cntr.x<<" , "<<cntr.y<<" )"<<std::endl;

std::cout<<"Square: "<<this->square()<<std::endl;

std::cout<<"Coordinates: "<<std::endl;

for(int i=0; i<points.size(); ++i){

std::cout<<"( "<<points[i].x<<" , "<<points[i].y<<" )"<<std::endl;

}

}else{

std::cout<<"Figure was not initialized"<<std::endl;

}

}

Figure(){}

protected:

std::string name;

std::vector<point> points;

//площадь треугольника по трём точкам

double square(point a, point b, point c){

return 0.5\*sqrt(((b.x-a.x)\*(b.x-a.x)+(b.y-a.y)\*(b.y-a.y))\*((c.x-a.x)\*(c.x-a.x)+(c.y-a.y)\*(c.y-a.y))-((b.x-a.x)\*(c.x-a.x)+(b.y-a.y)\*(c.y-a.y))\*((b.x-a.x)\*(c.x-a.x)+(c.y-a.y)\*(b.y-a.y)));

}

//косинус угла между векторами ab и cd

double cos\_points(point a, point b, point c, point d){

return ((b.x-a.x)\*(d.x-c.x)+(b.y-a.y)\*(d.y-c.y))/(sqrt((b.x-a.x)\*(b.x-a.x) + (b.y-a.y)\*(b.y-a.y)) \* sqrt((d.x-c.x)\*(d.x-c.x) + (d.y-c.y)\*(d.y-c.y)) );

}

double size\_points(point a, point b){

return sqrt((b.x-a.x)\*(b.x-a.x) + (b.y-a.y)\*(b.y-a.y));

}

};

class Square : public Figure {

public:

Square(std::vector<point> pnts){

int i=0;

if(pnts.size()==3) for(i =0; i<3; ++i) if( cos\_points(pnts[i], pnts[(i+1)%3], pnts[(i+1)%3], pnts[(i+2)%3])==0.0 && size\_points(pnts[i], pnts[(i+1)%3]) == size\_points(pnts[(i+1)%3], pnts[(i+2)%3])){

point fth;

//достраивает квадрат по трём точкам, если стороны квадрата не паралельны.

if((pnts[(i+2)%3].x-pnts[(i+1)%3].x)!=0 && (pnts[i].x-pnts[(i+1)%3].x)!=0 ){

fth.y=(pnts[i].x -pnts[(i+2)%3].x + pnts[i].y\*(pnts[(i+2)%3].y-pnts[(i+1)%3].y)/(pnts[(i+2)%3].x-pnts[(i+1)%3].x) -pnts[(i+2)%3].y\*(pnts[i].y-pnts[(i+1)%3].y)/(pnts[i].x-pnts[(1+i)%3].x))/((pnts[(i+2)%3].y-pnts[(i+1)%3].y)/(pnts[(i+2)%3].x-pnts[(i+1)%3].x) - (pnts[i].y-pnts[(i+1)%3].y)/(pnts[i].x-pnts[(i+1)%3].x));

fth.x= pnts[i].x - (pnts[i].y-pnts[(i+1)%3].y)/(pnts[i].x-pnts[(i+1)%3].x)\*(fth.y-pnts[(i+2)%3].y);

}else{

if(pnts[i].x==pnts[(i+1)%3].x){

fth.x=pnts[(i+2)%3].x;

fth.y=pnts[i].y;

}else{

fth.y=pnts[(i+2)%3].y;

fth.x=pnts[i].x;

}

}

for(int j=0; j<i; ++j){

point t;

t.x=pnts[0].x;

t.y=pnts[0].y;

pnts.erase(pnts.begin());

pnts.push\_back(t);

}

pnts.push\_back(fth);

break;

}

if(pnts.size()==4 && cos\_points(pnts[0], pnts[1], pnts[1], pnts[2])==0.0 && size\_points(pnts[0], pnts[1]) == size\_points(pnts[1], pnts[2]) && size\_points(pnts[0], pnts[1]) == size\_points(pnts[0], pnts[3]) && size\_points(pnts[1], pnts[2]) == size\_points(pnts[2], pnts[3])){

for(int i=0; i<pnts.size(); ++i) points.push\_back(pnts[i]);

name="Square";

}

}

void reinitialize(std::vector<point> pnts) override {

int i=0;

if(pnts.size()==3) for(i =0; i<3; ++i) if( cos\_points(pnts[i], pnts[(i+1)%3], pnts[(i+1)%3], pnts[(i+2)%3])==0.0 && size\_points(pnts[i], pnts[(i+1)%3]) == size\_points(pnts[(i+1)%3], pnts[(i+2)%3])){

point fth;

//достраивает квадрат по трём точкам, если стороны квадрата не паралельны.

if((pnts[(i+2)%3].x-pnts[(i+1)%3].x)!=0 && (pnts[i].x-pnts[(i+1)%3].x)!=0 ){

fth.y=(pnts[i].x -pnts[(i+2)%3].x + pnts[i].y\*(pnts[(i+2)%3].y-pnts[(i+1)%3].y)/(pnts[(i+2)%3].x-pnts[(i+1)%3].x) -pnts[(i+2)%3].y\*(pnts[i].y-pnts[(i+1)%3].y)/(pnts[i].x-pnts[(1+i)%3].x))/((pnts[(i+2)%3].y-pnts[(i+1)%3].y)/(pnts[(i+2)%3].x-pnts[(i+1)%3].x) - (pnts[i].y-pnts[(i+1)%3].y)/(pnts[i].x-pnts[(i+1)%3].x));

fth.x= pnts[i].x - (pnts[i].y-pnts[(i+1)%3].y)/(pnts[i].x-pnts[(i+1)%3].x)\*(fth.y-pnts[(i+2)%3].y);

}else{

if(pnts[i].x==pnts[(i+1)%3].x){

fth.x=pnts[(i+2)%3].x;

fth.y=pnts[i].y;

}else{

fth.y=pnts[(i+2)%3].y;

fth.x=pnts[i].x;

}

}

for(int j=0; j<i; ++j){

point t;

t.x=pnts[0].x;

t.y=pnts[0].y;

pnts.erase(pnts.begin());

pnts.push\_back(t);

}

pnts.push\_back(fth);

break;

}

if(pnts.size()==4 && cos\_points(pnts[0], pnts[1], pnts[1], pnts[2])==0.0 && size\_points(pnts[0], pnts[1]) == size\_points(pnts[1], pnts[2]) && size\_points(pnts[0], pnts[1]) == size\_points(pnts[0], pnts[3]) && size\_points(pnts[1], pnts[2]) == size\_points(pnts[2], pnts[3])){

for(int i=0; i<pnts.size(); ++i) points.push\_back(pnts[i]);

name="Square";

}

}

};

//new class

class Rectangle : public Figure {

public:

Rectangle(std::vector<point> pnts){

int i=0;

if(pnts.size()==3) for(i =0; i<3; ++i) if( cos\_points(pnts[i], pnts[(i+1)%3], pnts[(i+1)%3], pnts[(i+2)%3])==0.0 ){

point fth;

//достраивает прямоугольник по трём точкам, если стороны его не паралельны.

if((pnts[(i+2)%3].x-pnts[(i+1)%3].x)!=0 && (pnts[i].x-pnts[(i+1)%3].x)!=0 ){ //проверка непаралельности

fth.y=(pnts[i].x -pnts[(i+2)%3].x + pnts[i].y\*(pnts[(i+2)%3].y-pnts[(i+1)%3].y)/(pnts[(i+2)%3].x-pnts[(i+1)%3].x) -pnts[(i+2)%3].y\*(pnts[i].y-pnts[(i+1)%3].y)/(pnts[i].x-pnts[(1+i)%3].x))/((pnts[(i+2)%3].y-pnts[(i+1)%3].y)/(pnts[(i+2)%3].x-pnts[(i+1)%3].x) - (pnts[i].y-pnts[(i+1)%3].y)/(pnts[i].x-pnts[(i+1)%3].x));

fth.x= pnts[i].x - (pnts[i].y-pnts[(i+1)%3].y)/(pnts[i].x-pnts[(i+1)%3].x)\*(fth.y-pnts[(i+2)%3].y);//страшная и противная формулы координат такого прямоугольника

}else{

if(pnts[i].x==pnts[(i+1)%3].x){

fth.x=pnts[(i+2)%3].x;

fth.y=pnts[i].y;

}else{

fth.y=pnts[(i+2)%3].y;

fth.x=pnts[i].x;

}

}

for(int j=0; j<i; ++j){

point t;

t.x=pnts[0].x;

t.y=pnts[0].y;

pnts.erase(pnts.begin());

pnts.push\_back(t);

}

pnts.push\_back(fth);

break;

}

if(pnts.size()==4 && cos\_points(pnts[0], pnts[1], pnts[1], pnts[2])==0.0 && cos\_points(pnts[1], pnts[2], pnts[2], pnts[3])==0.0 ){

for(int i=0; i<pnts.size(); ++i) points.push\_back(pnts[i]);

name="Rectangle";

}

}

void reinitialize(std::vector<point> pnts) override {

int i=0;

if(pnts.size()==3) for(i =0; i<3; ++i) if( cos\_points(pnts[i], pnts[(i+1)%3], pnts[(i+1)%3], pnts[(i+2)%3])==0.0 ){

point fth;

//достраивает прямоугольник по трём точкам, если стороны его не паралельны.

if((pnts[(i+2)%3].x-pnts[(i+1)%3].x)!=0 && (pnts[i].x-pnts[(i+1)%3].x)!=0 ){ //проверка непаралельности

fth.y=(pnts[i].x -pnts[(i+2)%3].x + pnts[i].y\*(pnts[(i+2)%3].y-pnts[(i+1)%3].y)/(pnts[(i+2)%3].x-pnts[(i+1)%3].x) -pnts[(i+2)%3].y\*(pnts[i].y-pnts[(i+1)%3].y)/(pnts[i].x-pnts[(1+i)%3].x))/((pnts[(i+2)%3].y-pnts[(i+1)%3].y)/(pnts[(i+2)%3].x-pnts[(i+1)%3].x) - (pnts[i].y-pnts[(i+1)%3].y)/(pnts[i].x-pnts[(i+1)%3].x));

fth.x= pnts[i].x - (pnts[i].y-pnts[(i+1)%3].y)/(pnts[i].x-pnts[(i+1)%3].x)\*(fth.y-pnts[(i+2)%3].y);//страшная и противная формулы координат такого прямоугольника

}else{

if(pnts[i].x==pnts[(i+1)%3].x){

fth.x=pnts[(i+2)%3].x;

fth.y=pnts[i].y;

}else{

fth.y=pnts[(i+2)%3].y;

fth.x=pnts[i].x;

}

}

for(int j=0; j<i; ++j){

point t;

t.x=pnts[0].x;

t.y=pnts[0].y;

pnts.erase(pnts.begin());

pnts.push\_back(t);

}

pnts.push\_back(fth);

break;

}

if(pnts.size()==4 && cos\_points(pnts[0], pnts[1], pnts[1], pnts[2])==0.0 && cos\_points(pnts[1], pnts[2], pnts[2], pnts[3])==0.0 ){

for(int i=0; i<pnts.size(); ++i) points.push\_back(pnts[i]);

name="Rectangle";

}

}

};

//new class

class Trapeze : public Figure {

public:

Trapeze(std::vector<point> pnts){//трапеция по трём точкам может быть только с основаниями паралельными оси ОХ.

if(pnts.size()==3){

if(pnts[0].y != pnts[1].y && pnts[1].y == pnts[2].y){

point fth;

fth.y=pnts[0].y;

fth.x=pnts[2].x + pnts[1].x - pnts[0].x;

pnts.push\_back(fth);

}else if(pnts[0].y == pnts[1].y && pnts[1].y != pnts[2].y){

point fth;

fth.y=pnts[2].y;

fth.x=pnts[0].x + pnts[1].x - pnts[2].x;

pnts.push\_back(fth);

}if(pnts[0].y == pnts[2].y && pnts[1].y != pnts[2].y){

point fth;

fth.y=pnts[1].y;

fth.x=pnts[0].x + pnts[2].x - pnts[1].x;

pnts.push\_back(fth);

}

}

if(pnts.size()==4) for(int j=0; j<pnts.size(); ++j) if((cos\_points( pnts[(j+1)%4], pnts[(j+2)%4], pnts[(j+0)%4], pnts[(j+3)%4])>0.999 ||

cos\_points( pnts[(j+1)%4], pnts[(j+2)%4], pnts[(j+0)%4], pnts[(j+3)%4])<-0.999)

&& cos\_points( pnts[(j+1)%4], pnts[(j+2)%4], pnts[(j+1)%4], pnts[(j)%4])!=-1.0

&& cos\_points( pnts[(j+1)%4], pnts[(j+2)%4], pnts[(j+1)%4], pnts[(j)%4])!=1.0

&& size\_points(pnts[(j)%4], pnts[(j+1)%4]) == size\_points(pnts[(j+3)%4], pnts[(j+2)%4]) ){

for(int i=j; i<j+pnts.size(); ++i) points.push\_back(pnts[i%4]);

name="Trapeze";break;

}

}

void reinitialize(std::vector<point> pnts) override {//трапеция по трём точкам может быть только с основаниями паралельными оси ОХ.

if(pnts.size()==3){

if(pnts[0].y != pnts[1].y && pnts[1].y == pnts[2].y){

point fth;

fth.y=pnts[0].y;

fth.x=pnts[2].x + pnts[1].x - pnts[0].x;

pnts.push\_back(fth);

}else if(pnts[0].y == pnts[1].y && pnts[1].y != pnts[2].y){

point fth;

fth.y=pnts[2].y;

fth.x=pnts[0].x + pnts[1].x - pnts[2].x;

pnts.push\_back(fth);

}if(pnts[0].y == pnts[2].y && pnts[1].y != pnts[2].y){

point fth;

fth.y=pnts[1].y;

fth.x=pnts[0].x + pnts[2].x - pnts[1].x;

pnts.push\_back(fth);

}

}

if(pnts.size()==4) for(int j=0; j<pnts.size(); ++j) if((cos\_points( pnts[(j+1)%4], pnts[(j+2)%4], pnts[(j+0)%4], pnts[(j+3)%4])>0.999 ||

cos\_points( pnts[(j+1)%4], pnts[(j+2)%4], pnts[(j+0)%4], pnts[(j+3)%4])<-0.999)

&& cos\_points( pnts[(j+1)%4], pnts[(j+2)%4], pnts[(j+1)%4], pnts[(j)%4])!=-1.0

&& cos\_points( pnts[(j+1)%4], pnts[(j+2)%4], pnts[(j+1)%4], pnts[(j)%4])!=1.0

&& size\_points(pnts[(j)%4], pnts[(j+1)%4]) == size\_points(pnts[(j+3)%4], pnts[(j+2)%4]) ){

for(int i=j; i<j+pnts.size(); ++i) points.push\_back(pnts[i%4]);

name="Trapeze";break;

}

}

};

**main.cpp**

#include<vector>

#include<iostream>

#include<string>

#include"Figures.cpp"

void print\_menu(){

std::cout<<"1. Print menu."<<std::endl

<<"2. Input figure."<<std::endl

<<"3. Print information of all figures."<<std::endl

<<"4. Print information of figure."<<std::endl

<<"5. Print sum of squres of all figures."<<std::endl

<<"6. Remove figure with index."<<std::endl

<<"0. Exit."<<std::endl;

}

bool is\_number(const std::string& a){

for(int i=0; i<a.size(); ++i) if(!(a[i]>='0' && a[i]<='9' || a[i]=='.' ||a[i]=='-')) return false;

return true;

}

int main(){

std::string buffer;

std::vector<Figure\*> figures;

int status = 1;

while(status){

switch(status){

case 1 : print\_menu(); break;

case 2 :{

std::cout<<"Choose figure: "<<std::endl

<<"1. Polygn"<<std::endl

<<"2. Square "<<std::endl

<<"3. Rectangle"<<std::endl

<<"4. Trapeze"<<std::endl;

std::cout<<">>";

int a;

std::cin>>buffer;

if(!is\_number(buffer)){std::cout<<"Wrong input"<<std::endl; break;}

a=std::stoi(buffer);

point p;

switch(a){

case 1 :{

std::cout<<"input number of polygon vertex."<<std::endl;

int n;

std::cout<<">>";

std::cin>>buffer;

if(!is\_number(buffer)){std::cout<<"Wrong input"<<std::endl; break;}

n=stoi(buffer);

if(n<=2){

std::cout<<"Too little vertex."<<std::endl; break;

}

std::vector<point> ppp;

for(int i=0; i<n; ++i){

std::cout<<"Input vertex x and y coordinates."<<std::endl;

std::cout<<">>";

std::cin>>buffer;

if(!is\_number(buffer)){std::cout<<"Wrong input"<<std::endl; break;}

p.x=std::stod(buffer);

std::cin>>buffer;

if(!is\_number(buffer)){std::cout<<"Wrong input"<<std::endl; break;}

p.y=std::stod(buffer);

ppp.push\_back(p);

}

figures.push\_back(new Figure(ppp));

if(!(figures[figures.size()-1]->initialized())){

std::cout<<"Inputed coordinates are not belong convex polygon."<<std::endl;

delete figures[figures.size()-1];

figures.pop\_back();

}

break;

}

case 2:{

std::cout<<"Input number of square vertex (3 or 4)."<<std::endl;

int n;

std::cout<<">>";

std::cin>>buffer;

if(!is\_number(buffer)){std::cout<<"Wrong input"<<std::endl; break;}

n=stoi(buffer);

if(n!=3 && n!=4){

std::cout<<"Wrong number of vertex."<<std::endl;

break;

}

std::vector<point> ppp;

point p;

for(int i=0; i<n; ++i){

std::cout<<"Input vertex x and y coordinates."<<std::endl;

std::cout<<">>";

std::cin>>buffer;

if(!is\_number(buffer)){std::cout<<"Wrong input"<<std::endl; break;}

p.x=std::stod(buffer);

std::cin>>buffer;

if(!is\_number(buffer)){std::cout<<"Wrong input"<<std::endl; break;}

p.y=std::stod(buffer);

ppp.push\_back(p);

}

figures.push\_back(new Square(ppp));

if(!figures[figures.size()-1]->initialized()){

std::cout<<"Inputed coordinates are not belong convex square."<<std::endl;

delete figures[figures.size()-1];

figures.pop\_back();

}

break;

}

case 3:{

std::cout<<"Input number of rectangle vertex (3 or 4)."<<std::endl;

int n;

std::cout<<">>";

std::cin>>buffer;

if(!is\_number(buffer)){std::cout<<"Wrong input"<<std::endl; break;}

n=stoi(buffer);

if(n!=3 && n!=4){

std::cout<<"Wrong number of vertex."<<std::endl;

break;

}

std::vector<point> ppp;

point p;

for(int i=0; i<n; ++i){

std::cout<<"Input vertex x and y coordinates."<<std::endl;

std::cout<<">>";

std::cin>>buffer;

if(!is\_number(buffer)){std::cout<<"Wrong input"<<std::endl; break;}

p.x=std::stod(buffer);

std::cin>>buffer;

if(!is\_number(buffer)){std::cout<<"Wrong input"<<std::endl; break;}

p.y=std::stod(buffer);

ppp.push\_back(p);

}

figures.push\_back(new Rectangle(ppp));

if(!figures[figures.size()-1]->initialized()){

std::cout<<"Inputed coordinates are not belong convex rectangle."<<std::endl;

delete figures[figures.size()-1];

figures.pop\_back();

}

break;

}

case 4:{

std::cout<<"Input number of trapeze vertex (3, if base parallel to axis OX, or 4)."<<std::endl;

int n;

std::cout<<">>";

std::cin>>buffer;

if(!is\_number(buffer)){std::cout<<"Wrong input"<<std::endl; break;}

n=stoi(buffer);

if(n!=3 && n!=4){

std::cout<<"Wrong number of vertex."<<std::endl;

break;

}

std::vector<point> ppp;

point p;

for(int i=0; i<n; ++i){

std::cout<<"Input vertex x and y coordinates."<<std::endl; std::cout<<">>";

std::cin>>buffer;

if(!is\_number(buffer)){std::cout<<"Wrong input"<<std::endl; break;}

p.x=std::stod(buffer);

std::cin>>buffer;

if(!is\_number(buffer)){std::cout<<"Wrong input"<<std::endl; break;}

p.y=std::stod(buffer);

ppp.push\_back(p);

}

figures.push\_back(new Trapeze(ppp));

if(!figures[figures.size()-1]->initialized()){

std::cout<<"Inputed coordinates are not belong convex trapeze."<<std::endl;

delete figures[figures.size()-1];

figures.pop\_back();

}

break;

}

}

break;

}

case 3:{

if(figures.size()==0){std::cout<<"There is no figures"<<std::endl; break;}

std::cout<<"figures: "<<std::endl;

for(int i=0; i<figures.size(); ++i){ std::cout<<i+1<<"."; figures[i]->print\_info();}// std::cout<<\*figures[i]<<std::endl;}

break;

}

case 4:{

if(figures.size()==0){std::cout<<"There is no figures"<<std::endl; break;}

std::cout<<"input index of figure."<<std::endl;

int a;

std::cout<<">>";

std::cin>>buffer;

if(!is\_number(buffer)){std::cout<<"Wrong input"<<std::endl; break;}

a=stoi(buffer);

if(!(a-1>=0 && a-1<figures.size())){ std::cout<<"Wrong index."<<std::endl; break;}

figures[a]->print\_info();

//std::cout<<\*figures[a];

break;

}

case 5 :{

if(figures.size()==0){std::cout<<"There is no figures"<<std::endl; break;}

double sum=0.0;

for(int i=0; i<figures.size(); ++i) sum +=figures[i]->square();

std::cout<<sum<<std::endl;

break;

}

case 6 :{

if(figures.size()==0){std::cout<<"There is no figures"<<std::endl; break;}

int a;

std::cout<<"Input index of figure"<<std::endl;

std::cout<<">>";

std::cin>>buffer;

if(!is\_number(buffer)){std::cout<<"Wrong input"<<std::endl; break;}

a=stoi(buffer);

if(a<0 || a>=figures.size()){ std::cout<<"Wrong index"<<std::endl; break;}

figures.erase(figures.begin() +a );

break;

}

case -1: break;

default:

std::cout<<"Wrong choose."<<std::endl;

}

std::cout<<">>";

std::cin>>buffer;

if(!is\_number(buffer)){std::cout<<"Wrong input"<<std::endl; status=-1;}else status=stoi(buffer);

}

return 0;

}

1. Вывод

Научился работать с наследованием и полиморфизмом в c++. Реализовал программу для демонстрации этого. Программа работает не слишком эффективно, но в рамках допустимого.

**Список литературы**

1. Шилдт, Герберт. С++: базовый курс, 3-е изд. : Пер. с англ. - М. : ООО “И.Д. Вильямс”, 2018. - 624 с. : ил. - Парал. тит. англ.

2. Справочник по языку С++ [Электронный ресурс]. URL: [http://www.cplusplus.com/reference/](http://www.cplusplus.com/reference/deque/) (дата обращения: 20.09.2019).