МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ»

КАФЕДРА ТЕОРЕТИЧЕСКОЙ И ПРИКЛАДНОЙ ИНФОРМАТИКИ

Лабораторная работа №3

по дисциплине «Объектно-ориентированное программирование»

Факультет: ПМИ

Группа: ПМИ-03

Студенты: Малыгин С. А, Сидоров Д. И.

Преподаватель: Лисицын Д. В., Неделько В. М.

НОВОСИБИРСК

2021

1. Условие задачи:

По предложенному преподавателем варианту разработать программу на языке С++, в которой был бы определен класс-контейнер, т. е. класс объектов, служащих для хранения объектов класса, разработанного в предыдущей лабораторной работе. Контейнер должен быть реализован как динамическая структура данных. Разработать следующие функции-члены класса: конструктор, деструктор, функции для помещения объектов-фигур в контейнер, их возвращения (удаления), поиска в контейнере, «распечатки содержимого» контейнера – вывода информации о содержащихся в объекте-контейнере объектах и/или их графических образов, а также функции, обеспечивающие сохраняемость контейнера с использованием файла. Реализацию класса-контейнера поместить в отдельный файл. Разработать функцию, демонстрирующую поведение объекта-контейнера с несколькими объектами-фигурами.

Вариант 6а: неупорядоченная таблица

Описание функций:

Класс Quadrilateral:

void set\_data(RECT rt, POINT \*figure1, POINT \*figure2);

Загрузка точек фигуры в класс

void set\_RGB\_contour(int \*RGB);

Загрузка цвета контура в класс

void set\_RGB\_fill(int \*RGB);

Загрузка цвета заливки в класс

POINT get\_data(int fig, int numberOfDots);

Получение точек фигуры

void get\_RGB(int Whichcolor,int \*rgb);

Получение цвета контура или заливки

void save(std::ofstream &fout);

Сохранение в файл

void position(const int x, const int y);

Изменить позицию фигуры

void draw(HDC hdc, int type);

Нарисовать фигуры

bool Test\_Convex(const POINT \*figure);

Проверка на выпуклость

bool Test\_In\_Figure(const POINT \*outside, const POINT \*inside);

Проверка на невыход фигуры за фигуру

void reader(std::ifstream &fin, RECT rt);

Загрузка точек из файла

Класс Table\_for\_Quadrilateral:

Table\_for\_Quadrilateral(int a);

конструктор

~Table\_for\_Quadrilateral();

деструктор

void add\_elem(int fkey, Quadrilateral\* rec);

Добавить элемент в таблицу

void delete\_elem(int key);

Удалить элемент из таблицы

POINT get\_elem(int key, int numderFigure, int numberDots);

Получить точки фигуры из таблицы

void Paint(int key, int type, HDC hdc);

Нарисовать фигуру из таблицы

bool find(int key);

Найти фигуру в таблице

void saving(std::ofstream &fout);

Сохранить в файл фигуры в таблице

1. Алгоритм:

Table\_for\_Quadrilateral.сpp:

Table\_for\_Quadrilateral::Table\_for\_Quadrilateral(целая a)

{

maxsize = a;

data = new Tab[maxsize];

size = 0;

}

Table\_for\_Quadrilateral::~Table\_for\_Quadrilateral()

{

Удалить data;

}

Процедура Table\_for\_Quadrilateral::add\_elem(целая fkey, Quadrilateral\* rec)

{

Если (size == maxsize) Выдать исключение 4;

Quadrilateral \*x = rec;

data[size].key = fkey;

data[size].elem = x;

size++;

}

Table\_for\_Quadrilateral::delete\_elem(Целая key)

{

for (Целая i = 0; i < size; i++)

{

Если (data[i].key == key)

for (Целая j(i); j + 1 <= size; j++)

data[j] = data[j + 1];

}

size--;

data[size].key = NULL;

data[size].elem = NULL;

}

POINT Table\_for\_Quadrilateral::get\_elem(Целая key,Целая numderFigure,Целая numberDots)

{

POЦЕЛАЯ a = { 0,0 };

for (Целая i = 0; i < size; i++)

Если (data[i].key == key) a=data[i].elem->get\_data(numderFigure, numberDots);

вернуть a;

}

Процедура Table\_for\_Quadrilateral::PaЦелая(Целая key, Целая type, HDC hdc)

{

for(Целая i= 0;i<size;i++)

Если (data[i].key==key) data[i].elem->draw(hdc,type);

}

логическая Table\_for\_Quadrilateral::вводd(Целая key)

{

for (Целая i = 0; i < size; i++)

Если (data[i].key == key) вернуть истина;

вернуть ложь;

}

Процедура Table\_for\_Quadrilateral::saving(std::ofstream &вывод)

{

for (Целая i = 0; i < size; i++)

{

data[i].elem->save(вывод);

}

}

Quadrilateral.сpp:

Целая Quadrilateral::AB\_D(Целая x, Целая y, Целая xA, Целая yA, Целая xB, Целая yB)

{

вернуть (x - xA) \* (yB - yA) - (y - yA) \* (xB - xA);

}

Логическая Quadrilateral::Test\_Convex(константа POЦЕЛАЯ \*figure)

{

POINT ab =

{

figure[1].x - figure[0].x,

figure[1].y - figure[0].y

};

POINT bc =

{

figure[2].x - figure[1].x,

figure[2].y - figure[1].y

};

POINT cd =

{

figure[3].x - figure[2].x,

figure[3].y - figure[2].y

};

POINT da =

{

figure[0].x - figure[3].x,

figure[0].y - figure[3].y

};

Целая product1 = ab.x \* bc.y - ab.y \* bc.x;

Целая product2 = bc.x \* cd.y - bc.y \* cd.x;

Целая product3 = cd.x \* da.y - cd.y \* da.x;

Целая product4 = da.x \* ab.y - da.y \* ab.x;

вернуть ((product1 >= 0) && (product2 >= 0) && (product3 >= 0) && (product4 >= 0));

}

Логическая Quadrilateral::Test\_In\_Figure(константа POINT \*outside, константа POINT \*inside)

{

for (Целая i(0); i < 4; i++)

{

for (Целая j(0); j < 3; j++)

{

Если (AB\_D(inside[i].x, inside[i].y, outside[j].x, outside[j].y, outside[j + 1].x, outside[j + 1].y) > 0)

вернуть ложь;

}

}

вернуть истина;

}

Процедура Quadrilateral::reader(std::ifstream &ввод, RECT rt)

{

POINT check[4];

POINT check2[4];

Целая count(0);

Если (!ввод.is\_open()) Выбросить исключение 0;

иначе

{

ввод >> count;

Если (count == 2)

{

for (Целая i(0); i < 4; i++)

{

ввод >> check[i].x;

ввод >> check[i].y;

}

for (Целая i(0); i < 4; i++)

{

ввод >> check2[i].x;

ввод >> check2[i].y;

}

Закрыть файл;

Если (!Test\_Convex(check)) Выбросить исключение 1;

Если (!Test\_Convex(check2)) Выбросить исключение 1;

Если (!Test\_In\_Figure(check,check2)) Выбросить исключение 2;

for (Целая i(0); i < 4; i++)

{

XY[i].x = check[i].x;

XY[i].y = check[i].y;

}

for (Целая i(0); i < 4; i++)

{

XY2[i].x = check2[i].x;

XY2[i].y = check2[i].y;

}

}

иначе

{

for (Целая i(0); i < 4; i++)

{

ввод >> check[i].x;

ввод >> check[i].y;

}

Закрыть файл;

Если (!Test\_Convex(check)) Выбросить исключение 1;

for (Целая i(0); i < 4; i++)

{

XY[i].x = check[i].x;

XY[i].y = check[i].y;

}

}

}

}

Процедура Quadrilateral::set\_data(RECT rt, POINT \*figure1, POINT \*figure2)

{

Если (!Test\_Convex(figure1)) Выбросить исключение 1;

for (Целая i(0); i < 4; i++)

{

XY[i].x = figure1[i].x;

XY[i].y = figure1[i].y;

}

Целая sum(0);

for (Целая i = 0; i < 4; i++)

{

sum += figure2[i].x;

sum += figure2[i].y;

}

Если (sum != 0)

{

Если (!Test\_Convex(figure2)) Выбросить исключение 1;

Если (!Test\_In\_Figure(figure1, figure2)) Выбросить исключение 2;

for (Целая i(0); i < 4; i++)

{

XY2[i].x = figure2[i].x;

XY2[i].y = figure2[i].y;

}

}

}

Процедура Quadrilateral::set\_RGB\_contour(Целая \*rgb)

{

RGB\_contour = RGB(rgb[0], rgb[1], rgb[2]);

}

Процедура Quadrilateral::set\_RGB\_fill(Целая \*rgb)

{

RGB\_fill = RGB(rgb[0], rgb[1], rgb[2]);

}

POINT Quadrilateral::get\_data(Целая fig, Целая numberOfDots)

{

POЦЕЛАЯ a;

Если (fig == 1)

{

a.x = XY[numberOfDots].x;

a.y = XY[numberOfDots].y;

вернуть a;

}

иначе Если (fig == 2)

{

a.x = XY2[numberOfDots].x;

a.y = XY2[numberOfDots].y;

вернуть a;

}

}

Процедура Quadrilateral::get\_RGB(Целая Whichcolor,Целая \*rgb)

{

Если (Whichcolor == 1)

{

rgb[0] = GetRValue(RGB\_contour);

rgb[1] = GetGValue(RGB\_contour);

rgb[2] = GetBValue(RGB\_contour);

}

иначе

{

rgb[0] = GetRValue(RGB\_fill);

rgb[1] = GetGValue(RGB\_fill);

rgb[2] = GetBValue(RGB\_fill);

}

}

Процедура Quadrilateral::save(std::ofstream &вывод)

{

Если (вывод.is\_open())

{

for (Целая i(0); i < 4; i++)

вывод << XY[i].x << " " << XY[i].y << std::endl;

for (Целая i(0); i < 4; i++)

вывод << XY2[i].x << " " << XY2[i].y << std::endl;

вывод << std::endl;

}

иначе Выбросить исключение 0;

}

Процедура Quadrilateral::position(константа Целая x, константа Целая y)

{

for (Целая i(0); i < 4; i++)

{

XY[i].x += x;

XY[i].y += y;

}

}

Процедура Quadrilateral::draw(HDC hdc,Целая type)

{

Если (type == 1)

{

HPEN hPen1 = Создать перо(PS\_SOLID, 3, RGB\_contour);

XY[4] = { XY[0].x, XY[0].y };

Выбрать перо(hdc, hPen1);

do

{

Polyline(hdc, XY, 5);

} Пока (\_getch() != 27);

Удалить перо(hPen1);

}

Если (type == 2)

{

HPEN hPen1 = Создать перо(PS\_SOLID, 3, RGB\_contour);

HBRUSH hBrush1 = Создать кисть(RGB\_fill);

Выбрать кисть(hdc, hBrush1);

Выбрать перо(hdc, hPen1);

do

{

Polygon(hdc, XY, 4);

} Пока (\_getch() != 27);

Удалить кисть(hBrush1);

Удалить перо(hPen1);

}

Если (type == 3)

{

HPEN hPen1 = Создать перо(PS\_SOLID, 3, RGB\_contour);

HBRUSH hBrush1 = Создать кисть(RGB\_fill);

HBRUSH hBrush2 = GetStockBrush(BLACK\_BRUSH);

Выбрать перо(hdc, hPen1);

do

{

Выбрать кисть(hdc, hBrush1);

Polygon(hdc, XY, 4);

Выбрать кисть(hdc, hBrush2);

Polygon(hdc, XY2, 4);

} Пока (\_getch() != 27);

Удалить кисть(hBrush1);

Удалить кисть(hBrush2);

Удалить перо(hPen1);

}

}

1. Программа:

Table\_for\_Quadrilateral.h:

#pragma once

#include"Quadrilateral.h"

struct Tab

{

int key;

Quadrilateral \*elem;

};

class Table\_for\_Quadrilateral

{

private:

Tab \*data;

int size;

int maxsize;

public:

Table\_for\_Quadrilateral(int a);

~Table\_for\_Quadrilateral(); //деструктор

//int find\_place(int fkey);

void add\_elem(int fkey, Quadrilateral\* rec);

void delete\_elem(int key);

POINT get\_elem(int key, int numderFigure, int numberDots);

void Paint(int key, int type, HDC hdc);

bool find(int key);

void saving(std::ofstream &fout);

};

Table\_for\_Quadrilateral.cpp:

#include "Table\_for\_Quadrilateral.h"

Table\_for\_Quadrilateral::Table\_for\_Quadrilateral(int a)

{

maxsize = a;

data = new Tab[maxsize];

size = 0;

}

Table\_for\_Quadrilateral::~Table\_for\_Quadrilateral()

{

delete data;

}

//int Table\_for\_Quadrilateral::find\_place(int fkey)

//{

// for (int i = 0; i < size; i++)

// {

// if (data[i].key > fkey)

// return i;

// }

// return size;

//}

void Table\_for\_Quadrilateral::add\_elem(int fkey, Quadrilateral\* rec)

{

if (size == maxsize) throw 4;

/\*int place = find\_place(fkey);

for (int i = size; i > place; i--)

{

data[i] = data[i - 1];

}\*/

Quadrilateral \*x = rec;

data[size].key = fkey;

data[size].elem = x;

size++;

}

void Table\_for\_Quadrilateral::delete\_elem(int key)

{

for (int i = 0; i < size; i++)

{

if (data[i].key == key)

for (int j(i); j + 1 <= size; j++)

data[j] = data[j + 1];

}

size--;

data[size].key = NULL;

data[size].elem = NULL;

}

POINT Table\_for\_Quadrilateral::get\_elem(int key,int numderFigure,int numberDots)

{

POINT a = { 0,0 };

for (int i = 0; i < size; i++)

if (data[i].key == key) a=data[i].elem->get\_data(numderFigure, numberDots);

return a;

}

void Table\_for\_Quadrilateral::Paint(int key, int type, HDC hdc)

{

for(int i= 0;i<size;i++)

if (data[i].key==key) data[i].elem->draw(hdc,type);

}

bool Table\_for\_Quadrilateral::find(int key)

{

for (int i = 0; i < size; i++)

if (data[i].key == key) return true;

return false;

}

void Table\_for\_Quadrilateral::saving(std::ofstream &fout)

{

for (int i = 0; i < size; i++)

{

data[i].elem->save(fout);

}

}

Quadrilateral.h:

#pragma once

#include <fstream>

#include <conio.h>

#include <vector>

#include <windows.h>

#include <windowsx.h>

class Quadrilateral

{

private:

POINT XY[4] = { 0, 0, 0, 0, 0, 0, 0, 0 };

POINT XY2[4] = { 0, 0, 0, 0, 0, 0, 0, 0 };

COLORREF RGB\_contour;

COLORREF RGB\_fill;

int AB\_D(int x, int y, int xA, int yA, int xB, int yB);

public:

void set\_data(RECT rt, POINT \*figure1, POINT \*figure2);

void set\_RGB\_contour(int \*RGB);

void set\_RGB\_fill(int \*RGB);

POINT get\_data(int fig, int numberOfDots);

void get\_RGB(int Whichcolor,int \*rgb);

void save(std::ofstream &fout);

void position(const int x, const int y);

void draw(HDC hdc, int type);

bool Test\_Convex(const POINT \*figure);

bool Test\_In\_Figure(const POINT \*outside, const POINT \*inside);

void reader(std::ifstream &fin, RECT rt);

};

Quadrilateral.cpp:

#include "Quadrilateral.h"

int Quadrilateral::AB\_D(int x, int y, int xA, int yA, int xB, int yB)

{

return (x - xA) \* (yB - yA) - (y - yA) \* (xB - xA);

}

bool Quadrilateral::Test\_Convex(const POINT \*figure)

{

POINT ab =

{

figure[1].x - figure[0].x,

figure[1].y - figure[0].y

};

POINT bc =

{

figure[2].x - figure[1].x,

figure[2].y - figure[1].y

};

POINT cd =

{

figure[3].x - figure[2].x,

figure[3].y - figure[2].y

};

POINT da =

{

figure[0].x - figure[3].x,

figure[0].y - figure[3].y

};

int product1 = ab.x \* bc.y - ab.y \* bc.x;

int product2 = bc.x \* cd.y - bc.y \* cd.x;

int product3 = cd.x \* da.y - cd.y \* da.x;

int product4 = da.x \* ab.y - da.y \* ab.x;

return ((product1 >= 0) && (product2 >= 0) && (product3 >= 0) && (product4 >= 0));

}

bool Quadrilateral::Test\_In\_Figure(const POINT \*outside, const POINT \*inside)

{

for (int i(0); i < 4; i++)

{

for (int j(0); j < 3; j++)

{

if (AB\_D(inside[i].x, inside[i].y, outside[j].x, outside[j].y, outside[j + 1].x, outside[j + 1].y) > 0)

return false;

}

}

return true;

}

void Quadrilateral::reader(std::ifstream &fin, RECT rt)

{

POINT check[4];

POINT check2[4];

int count(0);

if (!fin.is\_open()) throw 0;

else

{

fin >> count;

if (count == 2)

{

for (int i(0); i < 4; i++)

{

fin >> check[i].x;

fin >> check[i].y;

}

for (int i(0); i < 4; i++)

{

fin >> check2[i].x;

fin >> check2[i].y;

}

fin.close();

if (!Test\_Convex(check)) throw 1;

if (!Test\_Convex(check2)) throw 1;

if (!Test\_In\_Figure(check,check2)) throw 2;

for (int i(0); i < 4; i++)

{

XY[i].x = check[i].x;

XY[i].y = check[i].y;

}

for (int i(0); i < 4; i++)

{

XY2[i].x = check2[i].x;

XY2[i].y = check2[i].y;

}

}

else

{

for (int i(0); i < 4; i++)

{

fin >> check[i].x;

fin >> check[i].y;

}

fin.close();

if (!Test\_Convex(check)) throw 1;

for (int i(0); i < 4; i++)

{

XY[i].x = check[i].x;

XY[i].y = check[i].y;

}

}

}

}

void Quadrilateral::set\_data(RECT rt, POINT \*figure1, POINT \*figure2)

{

if (!Test\_Convex(figure1)) throw 1;

for (int i(0); i < 4; i++)

{

XY[i].x = figure1[i].x;

XY[i].y = figure1[i].y;

}

int sum(0);

for (int i = 0; i < 4; i++)

{

sum += figure2[i].x;

sum += figure2[i].y;

}

if (sum != 0)

{

if (!Test\_Convex(figure2)) throw 1;

if (!Test\_In\_Figure(figure1, figure2)) throw 2;

for (int i(0); i < 4; i++)

{

XY2[i].x = figure2[i].x;

XY2[i].y = figure2[i].y;

}

}

}

void Quadrilateral::set\_RGB\_contour(int \*rgb)

{

RGB\_contour = RGB(rgb[0], rgb[1], rgb[2]);

}

void Quadrilateral::set\_RGB\_fill(int \*rgb)

{

RGB\_fill = RGB(rgb[0], rgb[1], rgb[2]);

}

POINT Quadrilateral::get\_data(int fig, int numberOfDots)

{

POINT a;

if (fig == 1)

{

a.x = XY[numberOfDots].x;

a.y = XY[numberOfDots].y;

return a;

}

else if (fig == 2)

{

a.x = XY2[numberOfDots].x;

a.y = XY2[numberOfDots].y;

return a;

}

}

void Quadrilateral::get\_RGB(int Whichcolor,int \*rgb)

{

if (Whichcolor == 1)

{

rgb[0] = GetRValue(RGB\_contour);

rgb[1] = GetGValue(RGB\_contour);

rgb[2] = GetBValue(RGB\_contour);

}

else

{

rgb[0] = GetRValue(RGB\_fill);

rgb[1] = GetGValue(RGB\_fill);

rgb[2] = GetBValue(RGB\_fill);

}

}

void Quadrilateral::save(std::ofstream &fout)

{

if (fout.is\_open())

{

for (int i(0); i < 4; i++)

fout << XY[i].x << " " << XY[i].y << std::endl;

for (int i(0); i < 4; i++)

fout << XY2[i].x << " " << XY2[i].y << std::endl;

fout << std::endl;

}

else throw 0;

}

void Quadrilateral::position(const int x, const int y)

{

for (int i(0); i < 4; i++)

{

XY[i].x += x;

XY[i].y += y;

}

}

void Quadrilateral::draw(HDC hdc,int type)

{

if (type == 1)

{

HPEN hPen1 = CreatePen(PS\_SOLID, 3, RGB\_contour);

XY[4] = { XY[0].x, XY[0].y };

SelectPen(hdc, hPen1);

do

{

Polyline(hdc, XY, 5);

} while (\_getch() != 27);

DeletePen(hPen1);

}

if (type == 2)

{

HPEN hPen1 = CreatePen(PS\_SOLID, 3, RGB\_contour);

HBRUSH hBrush1 = CreateSolidBrush(RGB\_fill);

SelectBrush(hdc, hBrush1);

SelectPen(hdc, hPen1);

do

{

Polygon(hdc, XY, 4);

} while (\_getch() != 27);

DeleteBrush(hBrush1);

DeletePen(hPen1);

}

if (type == 3)

{

HPEN hPen1 = CreatePen(PS\_SOLID, 3, RGB\_contour);

HBRUSH hBrush1 = CreateSolidBrush(RGB\_fill);

HBRUSH hBrush2 = GetStockBrush(BLACK\_BRUSH);

SelectPen(hdc, hPen1);

do

{

SelectBrush(hdc, hBrush1);

Polygon(hdc, XY, 4);

SelectBrush(hdc, hBrush2);

Polygon(hdc, XY2, 4);

} while (\_getch() != 27);

DeleteBrush(hBrush1);

DeleteBrush(hBrush2);

DeletePen(hPen1);

}

}

1. Набор тестов:

|  |  |  |  |
| --- | --- | --- | --- |
| № | Set.txt | Результат | Примечание |
| 1 | 700 700  1000 1000  800 1500  500 1500 | Фигура выходит за края экрана! | Фигура выходит за края экрана! |
| 2 |  | Файл пуст! | Файл пуст! |
| 3 | 1 1  200 1  200 200  1 200 |  | Контур фигуры |
| 4 | 100 100  300 100  100 400  90 250 |  | Заливка фигуры |
| 5 | 0 0  800 0  800 800  0 700  100 100  300 100  100 400  90 250 |  | Фигура в фигуре |
| 6 | 100 100  300 100  150 250  175 400 | Фигура не выпуклая! | Четырехугольник невыпуклый |
| 7 | 100 100  300 100  100 400  90 250  0 0  400 0  400 400  0 400 | Внутренняя фигура выходит за края внешней фигуры! | Фигура не в фигуре |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Функция | Было | Стало | Примечание |
| 1 | get\_data | Выбрана фигура 1 | Точка № 1  X: 1 Y: 1  Точка № 2  X: 200 Y: 1  Точка № 3  X: 200 Y: 200  Точка № 4  X: 1 Y: 200 | Функция работает правильно |
| 2 | set\_data | Координаты 1 фигуры  1 1  200 1  200 200  1 200 | Координаты 1 фигуры  0 100  200 100  200 200  1 200 | Функция работает правильно |
| 4 | save | Файл save.txt пуст  или занят устаревшей информацией | Файл заполнен новыми данными | Функция работает правильно |
| 6 | position | Координаты 1 фигуры  1 1  200 1  200 200  1 200  Сместить на 100 и 100 | Координаты 1 фигуры  101 101  300 101  300 300  101 300 | Функция работает правильно |

|  |  |  |  |
| --- | --- | --- | --- |
| № | Функция | Результат | Примечание |
| 1 | add\_elem(1,&Q)  0 0  400 0  400 400  0 400  0 0  300 100  300 400  200 300 | Элемент добавлен в таблицу  0 0  400 0  400 400  0 400  0 0  300 100  300 400  200 300 | Функция работает правильно |
| 2 | delete\_elem | Элемент удален из таблицы | Функция работает правильно |
| 3 | get\_elem(1,1,1) | Координаты точки были получены  0 0 | Функция работает правильно |
| 4 | Paint |  | Функция работает правильно |
| 5 | find | Если элемент найден, вернется истина, иначе ложь  1 или 0 | Функция работает правильно |
| 6 | saving | Фигуры были сохранены в файле  0 0  400 0  400 400  0 400  0 0  300 100  300 400  200 300  0 0  300 0  300 400  0 400  0 0  200 100  200 300  0 200 | Функция работает правильно |

1. Программа работает правильно, что подтверждают тесты.