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 №4
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:

Путем модификации программ, разработанных в лабораторных работах № 1, 2, разработать программу такую, чтобы в ней были определены несколько классов, реализующих понятие геометрической фигуры в графической системе: абстрактный класс «Фигура», содержащий чисто виртуальные функции; класс «Закрашенный», позволяющий задать кисть, ее параметры и, возможно, осуществить закраску; класс «Фигура-контур» – потомок класса «Фигура»; класс «Закрашенная фигура» – потомок класса «Фигура-контур», класс «Закрашенный» при этом использовать либо как второго родите-ля (множественное наследование), либо как часть класса «Закрашенная фигура» (агрегация); класс «Комбинированная фигура», реализующий две вложенные фигуры с закраской между ними.

Описание функций:

```
class Figure
{
protected:

public:
	virtual void Draw() = 0;
	Виртуальная функция
	virtual void reader(ifstream &fin, HDC new_hdc, RECT new_rt) = 0;
	Виртуальная функция

};

class Fill_Brush
{
protected:
	int mass_rgb_f[3];
public:
	void createbrush(HBRUSH &newcolorfill);
	Создать кисть
};

class Painted_Counter : public Figure
{
protected:
	POINT pt[5];
	int mass_rgb_c[3];
public:
	void Draw();
	Нарисовать контур
	void reader(ifstream &fin, HDC new_hdc, RECT new_rt);
	Загрузить контур фигуры из файла
	void save(std::ofstream &fout);
	Сохранение координат в файл
	void position(int x, int y);
	Изменение позиции фигуры
	void set_data(RECT rt, HDC new_hdc, POINT *figure, int *rgb);
	Загрузить контур фигуры
	POINT get_data(int numberDots);
	Получить контур фигуры
	bool Test_Convex(const POINT *outside);
	Проверка на выпуклость

};

class Quadrangle_fill : public Painted_Counter, public Fill_Brush
{
```

```
protected:

public:
	void Draw();
	Нарисовать закрашенный четырехугольник
	void reader(ifstream &fin, HDC new_hdc, RECT new_rt);
	Загрузить закрашенный четырехугольник из файла
	void save(std::ofstream &fout);
	Сохранение координат в файл
	void position(int x, int y);
	Изменение позиции фигуры
	void set_data(RECT rt, HDC new_hdc, POINT *figure, int *rgb, int *rgbFill);
	Загрузить закрашенный четырехугольник
	void getpen(int *clr);
	Получить перо
	void getbrush(int *clr);
	Получить кисть
	POINT get_data(int numberDots);
	Получить закрашенный четырехугольник
	POINT *get_pt()
	{
		return pt;
	}

};

class Two_Quadrangle : public Quadrangle_fill
{
protected:
	Quadrangle_fill trap;
public:
	void Draw();
	Нарисовать фигуру в фигуре
	void reader(ifstream &fin, HDC new_hdc, RECT new_rt);
	Загрузить фигуру в фигуре из файла
	void save(std::ofstream &fout);
	Сохранение координат в файл
	void position(int x, int y);
	Изменение позиции фигуры
	void set_data(RECT new_rt, HDC new_hdc, POINT *figure1, POINT *figure2, int *rgb,
int *rgbFill);
	Загрузить фигуру в фигуре
	POINT get_data(int numberDots, int numberFigure);
	Получить фигуру в фигуре
	int AB_D(int x, int y, int xA, int yA, int xB, int yB);
	Проверка на выпуклость
	bool Test_In_Figure();
	Проверка на выпуклость

	};
```

2) Программа:

Quadrangle.cpp:

```
#include "Quadrangle.h"
POINT Painted_Counter:: get_data(int numberDots)
{
	POINT a;
	a.x = pt[numberDots].x;
	a.y = pt[numberDots].y;
	return a;
}
bool Painted_Counter::Test_Convex(const POINT *outside)
{
	POINT ab =
```

```
	{
		outside[1].x - outside[0].x,
		outside[1].y - outside[0].y
	};

	POINT bc =
	{
		outside[2].x - outside[1].x,
		outside[2].y - outside[1].y
	};

	POINT cd =
	{
		outside[3].x - outside[2].x,
		outside[3].y - outside[2].y
	};

	POINT da =
	{
		outside[0].x - outside[3].x,
		outside[0].y - outside[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));
}
void Painted_Counter::Draw(HDC hdc, RECT new_rt)
{
	HPEN color;
	color = CreatePen(PS_SOLID, 3, RGB(mass_rgb_c[0], mass_rgb_c[1], mass_rgb_c[2]));
	SelectPen(hdc, color);
	while (_getch() != 27)
	{
		Polyline(hdc, pt, 5);
	}
	DeletePen(color);
}
void Painted_Counter::reader(ifstream &fin)
{
	POINT check[4];
	if (!fin.is_open()) throw 1;
	else
	{
		for (int i(0); i < 4; i++)
		{
			fin >> check[i].x;
			fin >> check[i].y;

		}
		for (int i = 0; i < 3; ++i)
		{
			fin >> mass_rgb_c[i];
		}
		if (!Test_Convex(check)) throw 2;
		for (int i(0); i < 4; i++)
		{
			pt[i].x = check[i].x;
			pt[i].y = check[i].y;
		}
	}
	pt[4].x= pt[0].x;
```

```
	pt[4].y= pt[0].x;
}
void Painted_Counter::set_data(POINT *figure, int *rgb)
{
	POINT check[4];
	for (int i(0); i < 4; i++)
	{
		check[i].x = figure[i].x;
		check[i].y = figure[i].y;
	}
	if (!Test_Convex(check)) throw 2;
	for (int i(0); i < 4; i++)
	{
		pt[i].x = check[i].x;
		pt[i].y = check[i].y;
	}
	for (int i = 0; i < 3; ++i)
	{
		mass_rgb_c[i] = rgb[i];
	}
	pt[4].x = pt[0].x;
	pt[4].y = pt[0].x;
}
void Painted_Counter::position(int x, int y)
{
	for (int i(0); i < 4; i++)
	{
		pt[i].x = pt[i].x + x;
		pt[i].y = pt[i].y + y;

	}
}
void Painted_Counter::save(std::ofstream &fout)
{
	for (int i(0); i < 4; i++)
	{
		fout << pt[i].x << " " << pt[i].y << std::endl;
	}
	fout << std::endl;
	for (int i(0); i < 4; i++)
	{
		fout << pt[i].x << " " << pt[i].y << std::endl;
	}

}
void Fill_Brush::createbrush(HBRUSH &newcolorfill)
{
	newcolorfill = CreateSolidBrush(RGB(mass_rgb_f[0], mass_rgb_f[1], mass_rgb_f[2]));
}
void Quadrangle_fill::Draw(HDC hdc, RECT new_rt)
{
	HBRUSH  colorfill;
	createbrush(colorfill);
	SelectBrush(hdc, colorfill);
	HPEN color= CreatePen(PS_SOLID, 3, RGB(mass_rgb_c[0], mass_rgb_c[1],
mass_rgb_c[2]));
	SelectPen(hdc, color);
	while (_getch() != 27)
	{
		Polygon(hdc, pt, 4);
	}
	DeleteBrush(colorfill);
	DeletePen(color);

}
```

```
void Quadrangle_fill::reader(ifstream &fin)
{
	Painted_Counter::reader(fin);
	for (int i = 0; i < 3; ++i)
	{
		fin >> mass_rgb_f[i];
	}
}
void Quadrangle_fill::set_data(POINT *figure, int *rgb, int *rgbFill)
{
	Painted_Counter::set_data(figure, rgb);
	for (int i = 0; i < 3; ++i)
	{
		mass_rgb_f[i] = rgbFill[i];
	}
}
void Quadrangle_fill::getpen(int *clr)
{
	for (int i = 0; i < 3; i++)
	{
		clr[i] = mass_rgb_c[i];
	}

}
void Quadrangle_fill::getbrush(int *clr)
{
	for (int i = 0; i < 3; i++)
	{
		clr[i] = mass_rgb_f[i];
	}

}
void Quadrangle_fill::position(int x, int y)
{
	Painted_Counter::position(x, y);
}
void Quadrangle_fill::save(std::ofstream &fout)
{
	Painted_Counter::save(fout);
}
POINT Quadrangle_fill::get_data(int numberDots)
{
	POINT a;
	a.x = pt[numberDots].x;
	a.y = pt[numberDots].y;
	return a;
}
int Two_Quadrangle::AB_D(int x, int y, int xA, int yA, int xB, int yB)
{
	return (x - xA) * (yB - yA) - (y - yA) * (xB - xA);
}
bool Two_Quadrangle::Test_In_Figure()
{
	POINT *pptt;
	pptt = trap.get_pt();
	for (int i(0); i < 4; i++)
	{
		for (int j(0); j < 3; j++)
		{
			if (AB_D(pptt[i].x, pptt[i].y, pt[j].x, pt[j].y, pt[j + 1].x, pt[j +
1].y) > 0)
				return false;
		}
	}
	return true;
```

```
}
void Two_Quadrangle::reader(ifstream &fin)
{
	Quadrangle_fill::reader(fin);
	trap.reader(fin);
	if (!Test_In_Figure()) throw 3;

}
void Two_Quadrangle::Draw(HDC hdc, RECT new_rt)
{
	//Quadrangle_fill::Draw();
	//trap.Draw();
	POINT *pptt;
	pptt = trap.get_pt();
	HBRUSH colorfill, colorfill2;
	createbrush(colorfill);
	trap.createbrush(colorfill2);
	HPEN color, color2;
	color = CreatePen(PS_SOLID, 3, RGB(mass_rgb_c[0], mass_rgb_c[1], mass_rgb_c[2]));
	int clr[3];
	trap.getpen(clr);
	color2 = CreatePen(PS_SOLID, 3, RGB(clr[0], clr[1], clr[2]));
	while (_getch() != 27)
	{
		SelectPen(hdc, color);
		SelectBrush(hdc, colorfill);
		Polygon(hdc, pt, 4);
		SelectPen(hdc, color2);
		SelectBrush(hdc, colorfill2);
		Polygon(hdc, pptt, 4);
	}
	DeleteBrush(colorfill);
	DeleteBrush(colorfill2);
	DeletePen(color);
	DeletePen(color2);
}
void Two_Quadrangle::set_data(POINT *figure1, POINT *figure2, int *rgb, int *rgbFill)
{
	Quadrangle_fill::set_data(figure1,rgb, rgbFill);
	trap.set_data(figure2, rgb, rgbFill);
	if (!Test_In_Figure()) throw 3;

}
POINT Two_Quadrangle::get_data(int numberDots, int numberFigure)
{
	POINT a;
	if (numberFigure == 2)
	{

		POINT *pptt;
		pptt = trap.get_pt();
		a.x = pptt[numberDots].x;
		a.y = pptt[numberDots].y;
		return a;
	}
	else
	{
		a.x = pt[numberDots].x;
		a.y = pt[numberDots].y;
		return a;

	}

}
void Two_Quadrangle::position(int x, int y)
```

```
{
	Painted_Counter::position(x, y);
}
void Two_Quadrangle::save(std::ofstream &fout)
{
	Painted_Counter::save(fout);
	fout << std::endl;
	POINT *pptt;
	pptt = trap.get_pt();
	for (int i(0); i < 4; i++)
	{
		fout << pptt[i].x << " " << pptt[i].y << std::endl;
	}
	fout << std::endl;
	for (int i(0); i < 4; i++)
	{
		fout << pptt[i].x << " " << pptt[i].y << std::endl;
	}

	 }
```

Quadrangle.h:

```
#pragma once
#include <windows.h>
#include <windowsx.h>
#include <conio.h>
#include <iostream>
#include <fstream>

using namespace std;

class Figure
{
protected:
	POINT Center;
public:
	virtual void Draw(HDC hdc, RECT new_rt) = 0;
	virtual void reader(ifstream &fin) = 0;
};

class Fill_Brush
{
protected:
	int mass_rgb_f[3];
public:
	void createbrush(HBRUSH &newcolorfill);
};

class Painted_Counter : public Figure
{
protected:
	POINT pt[5];
	int mass_rgb_c[3];
public:
	void Draw(HDC hdc, RECT new_rt);
	void reader(ifstream &fin);
	void save(std::ofstream &fout);
	void set_data(POINT *figure, int *rgb);
	void position(int x, int y);
	POINT get_data(int numberDots);
	bool Test_Convex(const POINT *outside);
};

class Quadrangle_fill : public Painted_Counter, public Fill_Brush
{
```

```
protected:

public:
	void Draw(HDC hdc, RECT new_rt);
	void reader(ifstream &fin);
	void save(std::ofstream &fout);
	void set_data(POINT *figure, int *rgb, int *rgbFill);
	void position(int x, int y);
	void getpen(int *clr);
	void getbrush(int *clr);
	POINT get_data(int numberDots);
	POINT *get_pt()
	{
		return pt;
	}
};

class Two_Quadrangle : public Quadrangle_fill
{
protected:
	Quadrangle_fill trap;
public:
	void Draw(HDC hdc, RECT new_rt);
	void reader(ifstream &fin);
	void save(std::ofstream &fout);
	void set_data(POINT *figure1, POINT *figure2, int *rgb, int *rgbFill);
	POINT get_data(int numberDots, int numberFigure);
	void position(int x, int y);
	int AB_D(int x, int y, int xA, int yA, int xB, int yB);
	bool Test_In_Figure();
	 };

	 Main.cpp:
#include "Quadrangle.h"
int main()
{
	setlocale(LC_CTYPE, "Russian");
	HWND hwnd = GetConsoleWindow();
	HDC hdc = GetDC(hwnd);
	RECT rt;
	GetClientRect(hwnd, &rt);
	ifstream fin;
	ofstream fout("save.txt");
	Figure* figure;
	Painted_Counter PC;
	Quadrangle_fill trap_fill;
	Two_Quadrangle two_trap;
	POINT Check[4];


	try
	{
		figure = &PC;
		fin.open("Quadrangle.txt");
		figure->reader(fin);
		figure->Draw(hdc, rt);
		fin.close();



		figure = &trap_fill;
		fin.open("Quadrangle.txt");
		figure->reader(fin);
		figure->Draw(hdc, rt);
		fin.close();
```

```
			figure = &two_trap;
			fin.open("Quadrangle.txt");
			figure->reader(fin);
			figure->Draw(hdc, rt);
			fin.close();


			fout.close();

		}
		catch (int error)
		{
			if (error == 1) cout << "Проблемы с файлом!" << endl;
			if (error == 2) cout << "Фигура не является выпуклым четырехугольником" <<
endl;
			if (error == 3) cout << "Вторая фигура выходит за границы первой" << endl;
		}
		system("pause");
		 }
```

3) Алгоритм:

Quadrangle.cpp:

```
ТОЧКА Класс контур:: get_data(Целая numberDots)
{
	ТОЧКА a;
	a.x = pt[numberDots].x;
	a.y = pt[numberDots].y;
	Вернуть a;
}
Логическая функция Класс контур::Test_Convex(константа ТОЧКА *outside)
{
	ТОЧКА ab =
	{
		outside[1].x - outside[0].x,
		outside[1].y - outside[0].y
	};

	ТОЧКА bc =
	{
		outside[2].x - outside[1].x,
		outside[2].y - outside[1].y
	};

	ТОЧКА cd =
	{
		outside[3].x - outside[2].x,
		outside[3].y - outside[2].y
	};

	ТОЧКА da =
	{
		outside[0].x - outside[3].x,
		outside[0].y - outside[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));
}
Процедура Класс контур::Draw()
{
	HPEN color;
	color = CreatePen(PS_SOLID, 3, RGB(mass_rgb_c[0], mass_rgb_c[1], mass_rgb_c[2]));
	Выбрать перо(hdc, color);
	Пока (_getch() != 27)
	{
		Polyline(hdc, pt, 5);
	}
	Удалить перо(color);
}
Процедура Класс контур::reader(ifstream &ввод, HDC new_hdc, RECT new_rt)
{
	hdc = new_hdc;
	rt = new_rt;
	ТОЧКА check[4];
	Если (!ввод.is_open()) Выбросить исключение 1;
	Иначе
	{
		for (Целая i(0); i < 4; i++)
		{
			ввод >> check[i].x;
			ввод >> check[i].y;

		}
		for (Целая i = 0; i < 3; ++i)
		{
			ввод >> mass_rgb_c[i];
		}
		Если (!Test_Convex(check)) Выбросить исключение 2;
		for (Целая i(0); i < 4; i++)
		{
			pt[i].x = check[i].x;
			pt[i].y = check[i].y;
		}
	}
	pt[4].x= pt[0].x;
	pt[4].y= pt[0].x;
}
Процедура Класс контур::set_data(RECT new_rt, HDC new_hdc, ТОЧКА *figure, Целая *rgb)
{
	hdc = new_hdc;
	rt = new_rt;
	ТОЧКА check[4];
	for (Целая i(0); i < 4; i++)
	{
		check[i].x = figure[i].x;
		check[i].y = figure[i].y;
	}
	Если (!Test_Convex(check)) Выбросить исключение 2;
	for (Целая i(0); i < 4; i++)
	{
		pt[i].x = check[i].x;
		pt[i].y = check[i].y;
	}
	for (Целая i = 0; i < 3; ++i)
	{
		mass_rgb_c[i] = rgb[i];
```

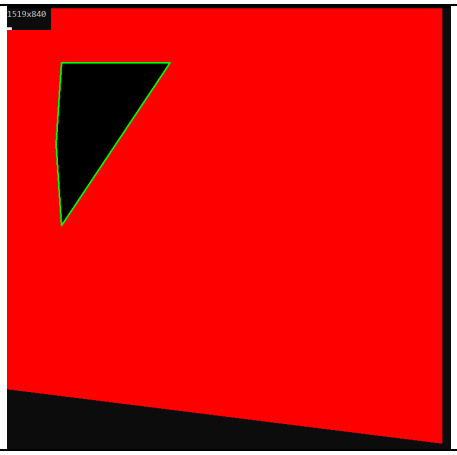
```
	}
	pt[4].x = pt[0].x;
	pt[4].y = pt[0].x;
}
Процедура Класс закраски::Создать кисть(HBRUSH &newcolorfill)
{
	newcolorfill = CreateSolidBrush(RGB(mass_rgb_f[0], mass_rgb_f[1], mass_rgb_f[2]));
}
Процедура Класс закрашенный четырехугольник::Draw()
{
	HBRUSH  colorfill;
	Создать кисть(colorfill);
	Выбрать кисть(hdc, colorfill);
	HPEN color= CreatePen(PS_SOLID, 3, RGB(mass_rgb_c[0], mass_rgb_c[1],
mass_rgb_c[2]));
	Выбрать перо(hdc, color);
	Пока (_getch() != 27)
	{
		Polygon(hdc, pt, 4);
	}
	Удалить кисть(colorfill);
	Удалить перо(color);

}
Процедура Класс закрашенный четырехугольник::reader(Еслиstream &ввод, HDC new_hdc, RECT
new_rt)
{
	hdc = new_hdc;
	rt = new_rt;
	Класс контур::reader(ввод, hdc, rt);
	for (Целая i = 0; i < 3; ++i)
	{
		ввод >> mass_rgb_f[i];
	}
}
Процедура Класс закрашенный четырехугольник::set_data(RECT new_rt, HDC new_hdc, ТОЧКА
*figure, Целая *rgb, Целая *rgbFill)
{
	hdc = new_hdc;
	rt = new_rt;
	Класс контур::set_data(rt, hdc, figure, rgb);
	for (Целая i = 0; i < 3; ++i)
	{
		mass_rgb_f[i] = rgbFill[i];
	}
}
Процедура Класс закрашенный четырехугольник::getpen(Целая *clr)
{
	for (Целая i = 0; i < 3; i++)
	{
		clr[i] = mass_rgb_c[i];
	}

}
Процедура Класс закрашенный четырехугольник::getbrush(Целая *clr)
{
	for (Целая i = 0; i < 3; i++)
	{
		clr[i] = mass_rgb_f[i];
	}

}
ТОЧКА Класс закрашенный четырехугольник::get_data(Целая numberDots)
{
	ТОЧКА a;
```

```
	a.x = pt[numberDots].x;
	a.y = pt[numberDots].y;
	Вернуть a;
}
Целая функция Класс фигура в фигуре::AB_D(Целая x, Целая y, Целая xA, Целая yA, Целая xB,
Целая yB)
{
	Вернуть (x - xA) * (yB - yA) - (y - yA) * (xB - xA);
}
Логическая Класс фигура в фигуре::Test_In_Figure()
{
	ТОЧКА *pptt;
	pptt = trap.get_pt();
	for (Целая i(0); i < 4; i++)
	{
		for (Целая j(0); j < 3; j++)
		{
			Если (AB_D(pptt[i].x, pptt[i].y, pt[j].x, pt[j].y, pt[j + 1].x, pt[j
+ 1].y) > 0)
				Вернуть false;
		}
	}
	Вернуть true;
}
Процедура Класс фигура в фигуре::reader(Еслиstream &ввод, HDC new_hdc, RECT new_rt)
{
	hdc = new_hdc;
	rt = new_rt;
	Класс закрашенный четырехугольник::reader(ввод, hdc, rt);
	trap.reader(ввод, hdc, rt);
	Если (!Test_In_Figure()) Выбросить исключение 3;

}
Процедура Класс фигура в фигуре::Draw()
{
	//Класс закрашенный четырехугольник::Draw();
	//trap.Draw();
	ТОЧКА *pptt;
	pptt = trap.get_pt();
	HBRUSH colorfill, colorfill2;
	Создать кисть(colorfill);
	trap.Создать кисть(colorfill2);
	HPEN color, color2;
	color = CreatePen(PS_SOLID, 3, RGB(mass_rgb_c[0], mass_rgb_c[1], mass_rgb_c[2]));
	Целая clr[3];
	trap.getpen(clr);
	color2 = CreatePen(PS_SOLID, 3, RGB(clr[0], clr[1], clr[2]));
	Пока (_getch() != 27)
	{
		Выбрать перо(hdc, color);
		Выбрать кисть(hdc, colorfill);
		Polygon(hdc, pt, 4);
		Выбрать перо(hdc, color2);
		Выбрать кисть(hdc, colorfill2);
		Polygon(hdc, pptt, 4);
	}
	Удалить кисть(colorfill);
	Удалить кисть(colorfill2);
	Удалить перо(color);
	Удалить перо(color2);
}
Процедура Класс фигура в фигуре::set_data(RECT new_rt, HDC new_hdc, ТОЧКА *figure1, ТОЧКА
*figure2, Целая *rgb, Целая *rgbFill)
{
	hdc = new_hdc;
```

```
        rt = new_rt;
        Класс закрашенный четырехугольник::set_data(rt, hdc, figure1,rgb, rgbFill);
        trap.set_data(rt, hdc, figure2, rgb, rgbFill);
        Если (!Test_In_Figure()) Выбросить исключение 3;

}
ТОЧКА Класс фигура в фигуре::get_data(Целая numberDots, Целая numberFigure)
{
        ТОЧКА a;
        Если (numberFigure == 2)
        {

                ТОЧКА *pptt;
                pptt = trap.get_pt();
                a.x = pptt[numberDots].x;
                a.y = pptt[numberDots].y;
                Вернуть a;
        }
        Иначе
        {
                a.x = pt[numberDots].x;
                a.y = pt[numberDots].y;
                Вернуть a;

        }

       }
```

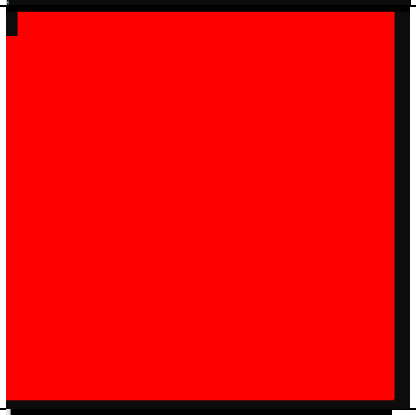
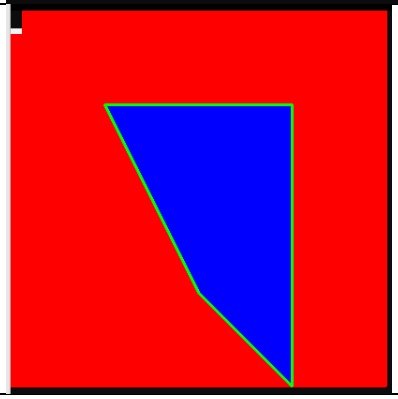
4) Набор тестов:

№	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	Координаты	Функция

		1 фигуры 1 1 200 1 200 200 1 200 Сместить на 100 и 100	1 фигуры 101 101 300 101 300 300 101 300	работает правильно
7	reader	0 0 0 0 0 0 0 0	0 100 200 100 200 200 1 200	Функция работает правильно

№	Функция	Результат	Примечание
1	Painted_Counter::Draw()		Функция работает правильно
2	Quadrangle_fill::Draw()		Функция работает правильно
3	Two_Quadrangle::Draw()		Функция работает правильно

5) Программа работает правильно, что подтверждают тесты.