МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННО БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Учебный Центр Информационных Технологий «Информатика»

Лабораторная работа № 1

по дисциплине «Объектно-ориентированное программирование»

Направление подготовки: 230105 - «Программное обеспечение вычислительной техники и автоматизированных систем»

Выполнил слушатель: Сокотущенко Иван Игоревич
Вариант: 2
Дата сдачи:
Преподаватель: Силов Я. В.

Новосибирск, 2017г.

1. Задание

Изучить основные понятия и функции графического интерфейса операционной системы Microsoft Windows. По предложенному преподавателем варианту разработать функции, рисующие следующие геометрические фигуры:

- незакрашенную фигуру (фигуру-контур);
- закрашенную фигуру;
- две вложенные одна в другую фигуры, внешняя фигура закрашена, за исключением пространства внутренней фигуры.

Включить в программу проверки корректности данных (нулевой радиус окружности, нарушение неравенства треугольника и т. д.), в том числе проверки нахождения фигуры в пределах окна и вложенности двух фигур.

Разработать программу так, чтобы в ней был определен класс, реализующий понятие геометрической фигуры в графической системе. Определить ответственность класса. Учесть, что общение с пользователем (включая ввод данных с клавиатуры и вывод данных на экран, если их предполагается использовать), а также реакцию на возникающие при работе с функциями класса ошибки следует производить вне функций класса. Определить атрибуты, необходимые для реализации класса. Поместить атрибуты в закрытую часть описания. Определить функции, необходимые для реализации класса. Выделить интерфейс класса и поместить его в открытую часть описания.

Включить в разработанный класс функции:

- устанавливающие и изменяющие геометрические и графические характеристики фигуры (set-функции);
- возвращающие геометрические и графические характеристики фигуры (get-функции);
- рисующие фигуру на экране;
- изменяющие положение фигуры на экране;
- обеспечивающие сохраняемость объекта: сохранение набора атрибутов объекта класса в файле и считывание его из файла (файлы для сохранения и считывания должны иметь один формат).

Обработку ошибок (нулевой радиус окружности, нарушение неравенства треугольника, другие ошибки задания фигуры, ошибки работы с графикой и др.) осуществлять с использованием механизма возбуждения и обработки исключительных ситуаций.

Разработать функцию, демонстрирующую поведение класса. Поместить реализацию класса в один файл, а демонстрационную функцию – в другой. Обеспечить необходимые межмодульные связи.

Подготовить текстовый файл с разработанной программой, оттранслировать, собрать и выполнить программу с учетом требований операционных систем и программных оболочек, в которых эта программа выполняется. При необходимости исправить ошибки и вновь повторить технологический процесс решения задачи.

Оформить отчет по лабораторной работе. Отчет должен содержать постановку задачи, алгоритм, описание разработанных функций, текст разработанной программы и результаты тестирования.

Защитить лабораторную работу, ответив на вопросы преподавателя.

Вариант задания №2 – выпуклый четырёхугольник.

2. Структурное описание

В данной лабораторной работе мы реализуем класс **’’Quadrangle.h’’**. Для удобства будем называть его «класс».

Для реализации данного класса созданы три файла. Заголовочный файл и два файла исходного кода. В заголовочном файле ’’Quadrangle.h’’ описывается интерфейс (тело) класса, в файле “Quadrangle_methods.cpp” реализованы методы класса.

Класс содержит в себе закрытые (private) члены класса:

- атрибуты, необходимые для реализации класса:
  - структуру **POINT** (имеет два целочисленных поля x, y), в которой хранятся координаты текущего значения пера;
- функции проверки корректности данных:
  - **isBulge()** - проверка на выпуклость многоугольника;
  - **isOnNest()** – проверка на вложенность внутреннего многоугольника во внешний.

- функцию, устанавливающую и изменяющую геометрические характеристики фигуры **setQuadrangle()**;
- функцию возвращающую геометрические характеристики фигуры **getQuadrangle()**.

В открытые (public) члены класса входят:
- конструктор класса по умолчанию **Quadrangle()**;
- деструктор **~Quadrangle()**;
- конструктор с параметрами **Quadrangle(…)**;
- функцию получения данных (координат четырёхугольника) из файла **enterCoordinatesFromFiles()**;
- функцию сохранения данных (координат четырёхугольника) в файл **saveCoordinatesToFile ()**;
- функцию, сдвигающую фигуру на экране на заданное смещение **changePosition()**;
- функцию, задающие новые координаты для четырёхугольника **establishNewPosition()**.

Общение с пользователем, т. е. рисование фигур должно производиться вне функций класса. Для этого создадим отдельный класс **’’View.h’’**.

Для реализации данного класса созданы три файла. Заголовочный файл и два файла исходного кода. В заголовочном файле ’’View.h’’ описывается интерфейс (тело) класса, в файле “View _methods.cpp” реализованы методы класса.

Класс содержит в себе закрытые (private) члены класса:
- атрибуты, необходимые для реализации класса:
  - **HWND hwnd** - идентификатор окна;
  - **HDC hdc** - контекст изображения;
  - **RECT rt** - структура окна с полями left, top, right, buttom;
  - **HPEN hPen** – перо;
  - **HBRUSH hBrush** – кисть;
  - **int Rpen, Gpen, Bpen** - цвет пера;
  - **int Rbrush, Gbrush, Bbrush** - цвет кисти;
  - **int width** - ширина кисти.
- функции проверки корректности данных:
  - **isAffiliation()** - проверки нахождения фигуры в пределах окна;
  - **isRGB()** – проверка правильности установленного цвета;
  - **isWidth()** - проверка правильности установленной толщины пера;
- функцию, устанавливающая и изменяющую графические характеристики фигуры **setCharacteristicsView()**.

В открытые (public) члены класса входят:
- конструктор класса по умолчанию **View ()**;
- деструктор **~View ()**;
- конструктор с параметрами **View (…)**;
- функцию, выводящую на консоль не закрашенную фигуру **viewUnpaintQudrangle()**;
- функцию, выводящую на консоль закрашенную фигуру **viewPaintQudrangle()**;
- функцию, выводящую на консоль две вложенные одна в другую фигуры, внешняя фигура закрашена, за исключением пространства внутренней фигуры **viewTwoNestedPaintQuadrangle()**;
- функцию получения данных (графических параметров четырёхугольника) из файла **enterCharacteristicsFromFiles ()**;
- функцию сохранения данных (графических параметров четырёхугольника) в файл **saveCharacteristicsToFile ()**;
- функцию, задающую новую толщину пера для фигуры **establishNewWidth()**;
- функцию, задающую новый цвет пера для фигуры **establishNewRGBPen()**;
- функцию, задающую новый цвет заливки для фигуры **establishNewRGBBrush()**;

В файле “main.cpp” размещена функция int main(), которая демонстрирует поведение класса.

3. Функциональное описание

Рассмотрим реализацию методов класса.

Подключаем заголовочный файл ’’Quadrangle.h’’.

Private:

- **bool isBulge(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4)** - проверка на выпуклость многоугольника, выполняется как дружественная функция при выводе на консоль изображения четырёхуголника. Метод принимает в качестве параметров координаты четырёхуголника (8 целочисленных значений) и проверяет, чтобы все углы при последовательном обходе четырёхугольника имели один знак. Знак узнаётся через векторное произведение векторов двух соседних сторон;
- **bool isOnNest(Quadrangle* internalQuadrangle)** - проверка на вложенность внутреннего многоугольника во внешний. Проводится путём сравнения координат внутреннего и внешнего четырёхугольников. Метод получает в качестве фактического параметра объект класса – внутренний четырёхугольник. Далее, с помощью this получаются координаты текущего (внешнего) и полученного (внутреннего) четырёхугольников. Путём сравнения их между собой происходит проверка. При нарушении вложенности создаётся сообщение "Внутренний четырёхугольник выходит за границы внешнего!". Выполняется как дружественная функция при выводе на консоль изображения четырёхугольника;
- **void setQuadrangle(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4)** - принимает в качестве параметров 8 целочисленных значений (координаты четырёхугольника) и создаёт динамический массив структур POINT с помощью оператора new. Далее метод инициализирует поля этой структуры значениями, полученными в качестве фактических параметров метода. При этом происходит проверка на выпуклость полученного четырёхугольника. Выполняется как дружественная функция при выводе на консоль изображения четырёхугольника;
- **POINT getQuadrangle()** - метод не принимает никаких параметров, в качестве результата возвращает указатель на динамический массив структур POINT. Выполняется как дружественная функция при выводе на консоль изображения четырёхугольника;

Public:

- конструктор класса по умолчанию **Quadrangle ()** – запускается автоматически при создании нового объекта класса без параметров, создаёт динамический массив структур POINT и инициализирует поля этой структуры некими значениями по умолчанию;
- деструктор **~ Quadrangle ()** – запускается автоматически при удалении объекта класса и освобождает выделенную для создания объекта класса динамическую память;
- конструктор с параметрами **Quadrangle(int, int, int, int, int, int, int, int)** – запускается автоматически при создании нового объекта класса с параметрами. Данный конструктор принимает в качестве параметров 8 целочисленных значений (координаты четырёхугольника) и с помощью оператора new создаёт динамический массив структур POINT, который инициализирует поля этой структуры значениями, полученными в качестве фактических параметров метода;
- **void enterCoordinatesFromFiles(string fd)** – метод получает данные (координаты четырёхугольника) из файла, указанного в качестве фактического параметра. Для этого с помощью функций из стандартной библиотеки <fstream> файл открывается, из него считываются координаты, происходит проверка координат на условие выпуклости четырёхугольника, и файл закрывается. Если в файле отсутствуют требуемые координаты, создаётся сообщение " В этом файле нет нужных данных!";
- **void saveCoordinatesToFile (string fd)** – метод сохраняет данные (координаты четырёхугольника) в файл, название которого задаётся в качестве фактического параметра. Для этого с помощью функций из стандартной библиотеки <fstream> файл создаётся, в него записываются координаты через пробел, и файл закрывается;
- **void changePosition(int x, int y)** – метод сдвигает фигуру на экране на заданное смещение. В качестве фактических параметров метод принимает координаты, на которые нужно переместить фигуру. В теле метода вызывается метод setQuadrangle, которому в качестве фактических параметров передаются координаты, смещённые на указанные позиции;
- **void establishNewPosition(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4)** – метод изменяет положение фигуры на экране. В качестве фактических параметров метод принимает новые координаты фигуры. В теле метода вызывается метод setQuadrangle, которому в качестве фактических параметров передаются указанные новые координаты.

Подключаем заголовочный файл ’’View.h’’.

Private:

- **bool isAffiliation(Quadrangle* quadrangle)** – метод проверки нахождения фигуры в пределах окна. Для определения размера области окна, в которую можно осуществлять вывод (точнее, ее координат в пикселях), предназначена функция GetClientRect, входящая в стандартную библиотеку <Windows.h>. Вызвав её, определяем фактические размеры окна - int maxX= rt.right, maxY = rt.bottom. В качестве фактического параметра метод принимает объект класса. С помощью метода getQuadrangle() получим значение координат этого объекта класса. Сравнивая координаты объекта и координаты окна, можно выполнить проверку. В случае, если координаты объекта находятся вне области окна создаётся сообщение "Четырёхугольник выходит за границу экрана!";
- **bool isRGB(int R, int G, int B)** – метод проверки правильности установленного цвета. В качестве фактических параметров метод принимает значения цветов в формате RGB. В теле метода происходит проверка принадлежности значений цветов области значений 0..255. В случае, если указанные значения не входят в данную область создаётся сообщение "Такого цвета не существует!";
- **bool isWidth(int width)** – метод проверки правильности установленной толщины пера. В качестве фактических параметров метод принимает целочисленное значение пера. В теле метода происходит проверка принадлежности значения пера области значений 0..9. В случае, если указанное значение не входит в данную область создаётся сообщение "Такую толщину линий задать нельзя! Максимальная толщина - 9 пикселей!";
- **void setCharacteristicsView(int width, int Rpen, int Gpen, int Bpen, int Rbrush, int Gbrush, int Bbrush)** – метод устанавливает и изменяет графические характеристики фигуры. Принимает в качестве параметров 7 целочисленных значений (толщину пера, значение цвета пера, значение цвета заливки). Метод устанавливает полученные значения для текущего объекта. При этом происходит проверка корректности полученных значений – вызываются методы checkRGB() и checkWidth(). Далее метод устанавливает стиль, ширину и цвет пера и цвет заливки;

Public:

- конструктор класса по умолчанию **View()** – запускается автоматически при создании нового объекта класса без параметров, возвращает контекст изображения окна с идентификатором, инициализирует параметры кисти и заливки некими значениями по умолчанию;
- деструктор **~View()** – запускается автоматически при удалении объекта класса и освобождает контекст отображения, полученный для окна. Также выбирает в контекст отображения другие перо и заливку и удаляет текущие перо и заливку;
- конструктор с параметрами **View(HWND, HDC, int width, int Rpen, int Gpen, int Bpen, int Rbrush, int Gbrush, int Bbrush)** – запускается автоматически при создании нового объекта класса с параметрами. Данный конструктор принимает в качестве параметров окно, контекст отображения и 7 целочисленных значений (толщину пера, значение цвета пера, значение цвета заливки) и инициализирует ими объект. При этом происходит проверка корректности полученных значений – вызываются методы checkRGB() и checkWidth(). Далее метод устанавливает стиль, ширину и цвет пера и цвет заливки;
- **void viewUnpaintQudrangle(Quadrangle* quadrangle)** – метод, выводит на консоль не закрашенную фигуру. Сначала происходит проверка нахождения фигуры в пределах окна. Далее указывается прямоугольник для перерисовки окна и обновляется содержимое окна. Потом выбираются кисть и заливка в контекст отображения. Заливка выбирается пустая, типа NULL_BRUSH. Затем получаются координаты объекта, получаемого в качестве фактического параметра с помощью дружественной функции getQuadrangle(). И, наконец, происходит вывод на консоль изображения четырёхугольника с помощью стандартной функции Polygon() из стандартной библиотеки <Windows.h>;
- **void viewPaintQudrangle(Quadrangle* quadrangle)** – метод, выводит на консоль изакрашенную фигуру. Сначала происходит проверка нахождения фигуры в пределах окна. Далее указывается прямоугольник для перерисовки окна и обновляется содержимое окна. Потом выбираются кисть и заливка в контекст отображения. Затем получаются координаты объекта, получаемого в качестве фактического параметра с помощью дружественной функции getQuadrangle(). И, наконец, происходит вывод на консоль изображения четырёхугольника с помощью стандартной функции Polygon() из стандартной библиотеки <Windows.h>;
- **void viewTwoNestedPaintQuadrangle(Quadrangle *externalQuadrangle, Quadrangle *internalQuadrangle)** – метод, выводит на консоль две вложенные одна в другую фигуры, внешняя фигура закрашена, за исключением пространства внутренней фигуры. Сначала происходит проверка нахождения фигур в пределах окна. Далее указывается прямоугольник для перерисовки окна и об-

новляется содержимое окна. Потом выбираются кисть и заливка в контекст отображения. При этом для обоих четырёхугольников заливка выбирается пустая, типа NULL_BRUSH. Затем получаются координаты объекта, получаемого в качестве фактического параметра с помощью дружественной функции getQuadrangle(). И, наконец, происходит вывод на консоль изображения четырёхугольника с помощью стандартной функции Polygon() из стандартной библиотеки <Windows.h>. Далее происходит закраска внутренней области внешнего четырёхугольника с помощью стандартной функции FloodFill(), для которой указывается точка, находящаяся внутри замкнутой области;

- **void enterCharacteristicsFromFiles(string fd)** – метод получает данные (графических параметров четырёхугольника) из файла, указанного в качестве фактического параметра. Для этого с помощью функций из стандартной библиотеки <fstream> файл открывается, из него считываются координаты, происходит проверка координат на условия корректности данных, и файл закрывается. Если в файле отсутствуют требуемые координаты, создаётся сообщение " В этом файле нет нужных данных!";
- **void saveCharacteristicsToFile(string fd)** – метод сохраняет данные (графических параметров четырёхугольника) в файл, название которого задаётся в качестве фактического параметра. Для этого с помощью функций из стандартной библиотеки <fstream> файл создаётся, в него записываются координаты через пробел, и файл закрывается;
- **void establishNewWidth(int width)** – метод задаёт новую толщину пера для фигуры. Принимает в качестве параметра целочисленное значений (толщину пера). Метод устанавливает полученное значения для текущего объекта. В теле метода вызывается метод setCharacteristicsView, которому в качестве фактического параметра передаётся новое значение толщины пера;
- **void establishNewRGBPen(int Rpen, int Gpen, int Bpen)** – метод задаёт новый цвет пера для фигуры. Принимает в качестве параметра целочисленные значения (значения цвета пера в формате RGB). Метод устанавливает полученное значение для текущего объекта. В теле метода вызывается метод setCharacteristicsView, которому в качестве фактического параметра передаются новые значения цвета пера;
- **void establishNewRGBBrush(int Rbrush, int Gbrush, int Bbrush)** – метод задаёт новый цвет заливки для фигуры. Принимает в качестве параметра целочисленные значения (значения цвета заливки в формате RGB). Метод устанавливает полученное значение для текущего объекта. В теле метода вызывается метод setCharacteristicsView, которому в качестве фактического параметра передаются новые значения цвета заливки.

4. Описание работы программы

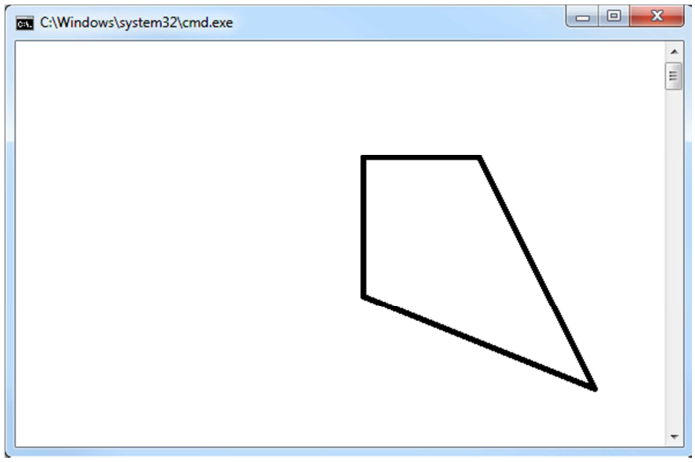

Рис. 1 – Вывод в консоль не закрашенной фигуры (фигуры-контура).

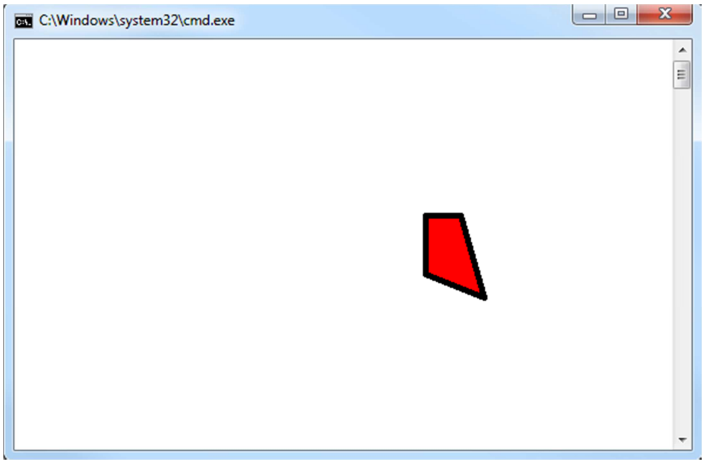

Рис. 2 – Вывод в консоль закрашенной фигуры.

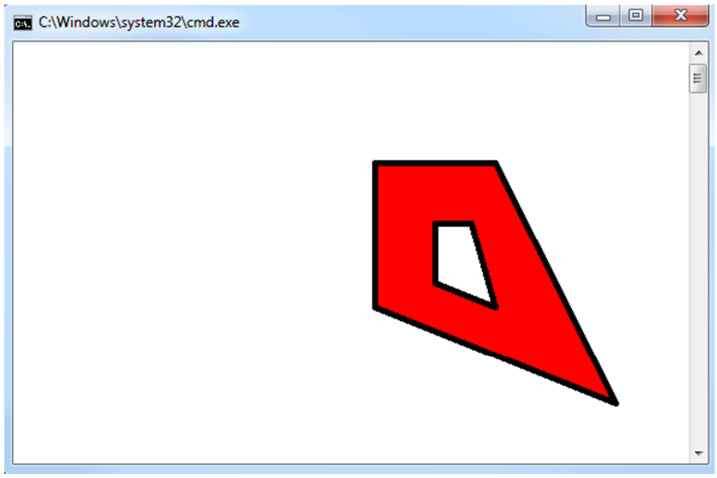

Рис. 3 - Вывод в консоль двух вложенных одна в другую фигуры (внешняя фигура закрашена за исключением пространства внутренней фигуры).

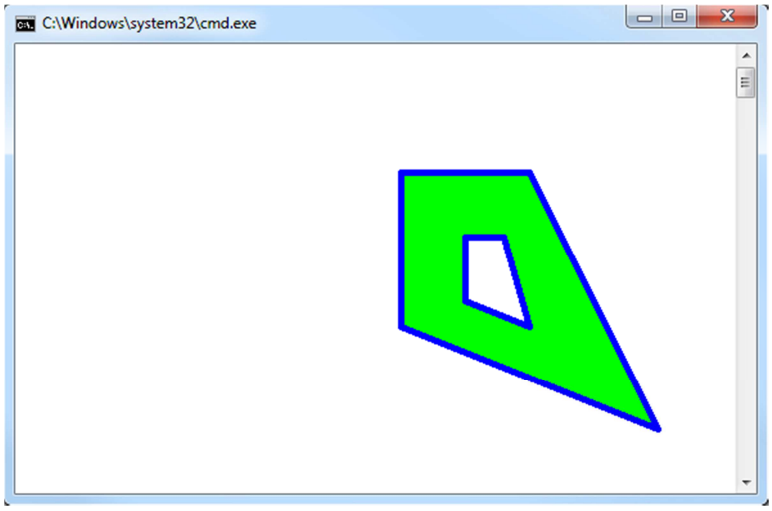

Рис. 4 - Вывод в консоль двух вложенных одна в другую фигуры (внешняя фигура закрашена за исключением пространства внутренней фигуры) с новыми графическими параметрами, полученными из файла.

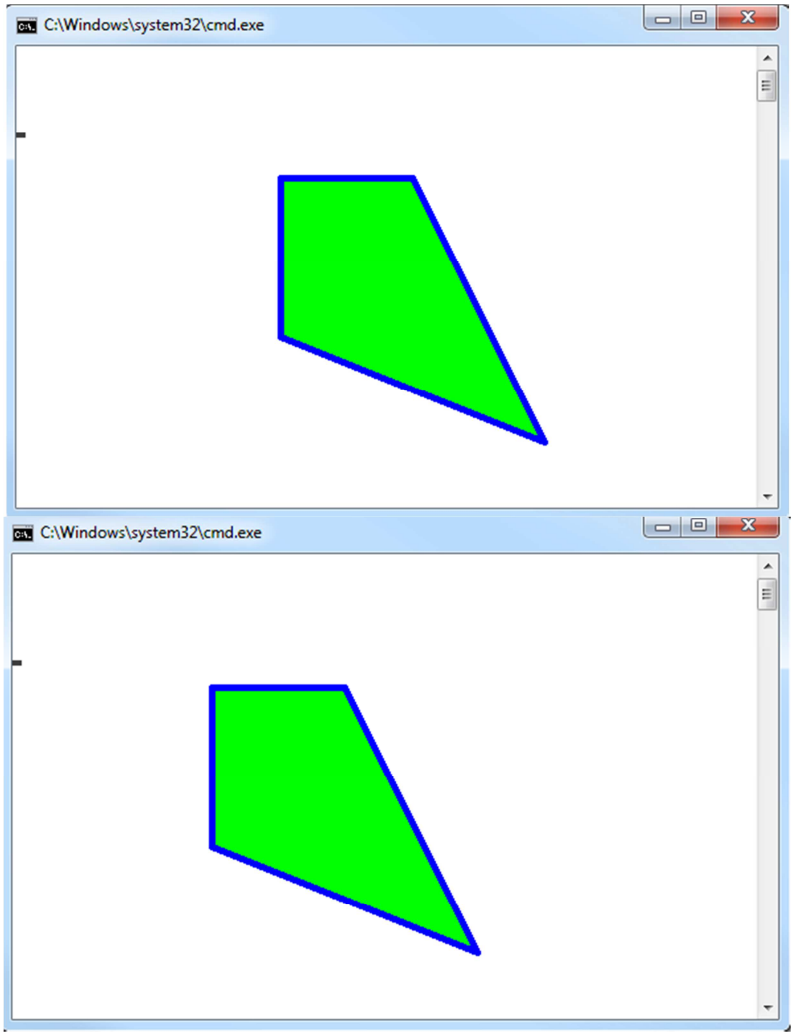

Рис. 5 - Демонстрация работы функции сдвига фигуры.

Выводы

В ходе выполнения лабораторной работы были изучены основные понятия и функции графического интерфейса операционной системы Microsoft Windows.

Также были изучены следующие понятия: определение типа данных пользователя с помощью конструкции class, открытая и закрытая части описания класса, определение набора функций класса, интерфейс и реализация класса, реализация понятия модульности в языке C++.

Ссылка на github.com: **https://github.com/IvanSokotushenko/OOP/tree/master/Lab1var2v10**.

Приложение. Текст программы

```
#pragma once
#include <windows.h>
#include <iostream>
#include <fstream>
using namespace std;

class Quadrangle
{
	POINT *points;
	bool isBulge(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4);
	bool isOnTheNested(Quadrangle *internalQuadrangle);
	POINT *setQuadrangle(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4);
	POINT *getQuadrangle();
	friend class View;
public:
	Quadrangle();
	~Quadrangle();
	Quadrangle(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4);
	void enterCoordinatesFromFiles(string fd);
	void saveCoordinatesToFile(string fd);
	void changePosition(int x, int y);
	void establishNewPosition(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4);
};

#include "Quadrangle.h"
bool Quadrangle::isBulge(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4)
{
	if (((x2 - x1)*(y3 - y1) - (y2 - y1)*(x3 - x1))*((x2 - x1)*(y4 - y1) - (y2 - y1)*(x4 - x1))<0 ||
		((x4 - x3)*(y2 - y3) - (y4 - y3)*(x2 - x3))*((x4 - x3)*(y1 - y3) - (y4 - y3)*(x1 - x3))<0)
		return false;
	else
		return true;
}

bool Quadrangle::isOnTheNested(Quadrangle *internalQuadrangle)
{
	POINT *points1 = this->getQuadrangle();
	POINT *points2 = internalQuadrangle->getQuadrangle();
	if ((points2[0].x < points1[0].x) || (points2[0].y < points1[0].y) ||
		(points2[1].x > points1[1].x) || (points2[1].y < points1[1].y) ||
		(points2[2].x > points1[2].x) || (points2[2].y > points1[2].y) ||
		(points2[3].x < points1[3].x) || (points2[3].y > points1[3].y))
		return false;
	else
		return true;
}

POINT *Quadrangle::setQuadrangle(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4)
{
	if (isBulge(x1, y1, x2, y2, x3, y3, x4, y4) == false)
		throw exception("Четырёхугольник не является выпуклым!");
	points = new POINT[4];
	points[0] = { x1, y1 };
	points[1] = { x2, y2 };
	points[2] = { x3, y3 };
	points[3] = { x4, y4 };
}

POINT* Quadrangle::getQuadrangle()
{
	return this->points;
}

Quadrangle::Quadrangle()
{
	points = new POINT[4];
```

```
	points[0] = { 300, 100 };
	points[1] = { 400, 100 };
	points[2] = { 300, 200 };
	points[3] = { 500, 300 };
}

Quadrangle::~Quadrangle()
{
	delete[] getQuadrangle();
}

Quadrangle::Quadrangle(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4)
{
	if (isBulge(x1, y1, x2, y2, x3, y3, x4, y4) == false)
		throw exception("Четырёхугольник не является выпуклым!");
	points = new POINT[4];
	points[0] = { x1, y1 };
	points[1] = { x2, y2 };
	points[2] = { x3, y3 };
	points[3] = { x4, y4 };
}

void Quadrangle::enterCoordinatesFromFiles(string fd)
{
	ifstream in(fd, ios::in);
	if (in.is_open())
	{
		POINT *points = this->getQuadrangle();
		in >> points[0].x;
		in >> points[0].y;
		in >> points[1].x;
		in >> points[1].y;
		in >> points[2].x;
		in >> points[2].y;
		in >> points[3].x;
		in >> points[3].y;
		if (isBulge(points[0].x, points[0].y, points[1].x, points[1].y, points[2].x, points[2].y, points[3].x, points[3].y) ==
false)
			throw exception("Четырёхугольник не является выпуклым!");
		in.close();
	}
	else
		throw exception("В этом файле нет нужных данных!");
}

void Quadrangle::saveCoordinatesToFile(string fd)
{
	ofstream out(fd, ios::out);
	out << this->points[0].x;
	out << " ";
	out << this->points[0].y;
	out << " ";
	out << this->points[1].x;
	out << " ";
	out << this->points[1].y;
	out << " ";
	out << this->points[2].x;
	out << " ";
	out << this->points[2].y;
	out << " ";
	out << this->points[3].x;
	out << " ";
	out << this->points[3].y;
	out << " ";
	out.close();
}
```

```
void Quadrangle::changePosition(int x, int y)
{
	POINT *points = this->getQuadrangle();
	setQuadrangle(points[0].x + x, points[0].y + y, points[1].x + x, points[1].y + y,
		points[2].x + x, points[2].y + y, points[3].x + x, points[3].y + y);
}

void Quadrangle::establishNewPosition(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4)
{
	POINT *points = this->getQuadrangle();
	setQuadrangle(points[0].x = x1, points[0].y = y1, points[1].x = x2, points[1].y = y2,
		points[2].x = x3, points[2].y = y3, points[3].x = x4, points[3].y = y4);
}

#pragma once
#include <windowsx.h>
#include "Quadrangle.h"

class View
{
	HWND hwnd;
	HDC hdc;
	RECT rt;
	HPEN hPen;
	HBRUSH hBrush;
	int Rpen, Gpen, Bpen;
	int Rbrush, Gbrush, Bbrush;
	int width;

	bool isAffiliation(Quadrangle *quadrangle);
	bool isRGB(int R, int G, int B);
	bool isWidth(int width);
	void setCharacteristicsView(int width, int Rpen, int Gpen, int Bpen, int Rbrush, int Gbrush, int Bbrush);
public:
	View();
	~View();
	View(HWND, HDC, int width, int Rpen, int Gpen, int Bpen, int Rbrush, int Gbrush, int Bbrush);
	void viewUnpaintQudrangle(Quadrangle *quadrangle);
	void viewPaintQudrangle(Quadrangle *quadrangle);
	void viewTwoNestedPaintQuadrangle(Quadrangle *externalQuadrangle, Quadrangle *internalQuadrangle);
	void enterCharacteristicsFromFiles(string fd);
	void saveCharacteristicsToFile(string fd);
	void establishNewWidth(int width);
	void establishNewRGBPen(int Rpen, int Gpen, int Bpen);
	void establishNewRGBBrush(int Rbrush, int Gbrush, int Bbrush);
};
#include "View.h"

bool View::isAffiliation(Quadrangle *quadrangle)
{
	GetClientRect(this->hwnd, &(this->rt));
	POINT *points = quadrangle->getQuadrangle();
	if ((points[0].x < rt.left) || (points[0].y < rt.top) || (points[0].x > rt.right) || (points[0].y > rt.bottom) ||
		(points[1].x > rt.right) || (points[1].y < rt.top) || (points[1].x < rt.left) || (points[1].y > rt.bottom) ||
		(points[2].x > rt.right) || (points[2].y > rt.bottom) || (points[2].x < rt.left) || (points[2].y < rt.top) ||
		(points[3].x < rt.left) || (points[3].y > rt.bottom) || (points[3].x > rt.right) || (points[3].y < rt.top))
		return false;
	else
		return true;
}
bool View::isRGB(int R, int G, int B)
{
	if ((R < 0 || R > 255) || (G < 0 || G > 255) || (B < 0 || B> 255))
		return false;
	else
		return true;
}
```

```
bool View::isWidth(int width)
{
	if (width < 0 || width > 9)
		return false;
	else
		return true;
}

void View::setCharacteristicsView(int width, int Rpen, int Gpen, int Bpen, int Rbrush, int Gbrush, int Bbrush)
{
	this->width = width;
	if (isWidth(this->width) == false)
		throw exception("Такую толщину линий задать нельзя!\nМаксимальная толщина - 9 пикселей.");
	this->Rpen = Rpen;
	this->Gpen = Gpen;
	this->Bpen = Bpen;
	if (isRGB(this->Rpen, this->Gpen, this->Bpen) == false)
		throw exception("Такого цвета не существует!");
	this->Rbrush = Rbrush;
	this->Gbrush = Gbrush;
	this->Bbrush = Bbrush;
	if (isRGB(this->Rbrush, this->Gbrush, this->Bbrush) == false)
		throw exception("Такого цвета не существует!");
	hPen = CreatePen(PS_SOLID, this->width, RGB(this->Rpen, this->Gpen, this->Bpen));
	hBrush = CreateSolidBrush(RGB(this->Rbrush, this->Gbrush, this->Bbrush));
}

View::View()
{
	hwnd = GetConsoleWindow();
	hdc = GetDC(hwnd);
	Rpen = Gpen = Bpen = 0;
	Rbrush = Gbrush = Bbrush = 255;
	hPen = CreatePen(PS_SOLID, 5, RGB(this->Rpen, this->Gpen, this->Bpen));
	hBrush = CreateSolidBrush(RGB(this->Rbrush, this->Gbrush, this->Bbrush));
}

View::~View()
{
	SelectPen(hdc, hPen);
	SelectBrush(hdc, hBrush);
	DeletePen(hPen);
	DeleteBrush(hBrush);
	ReleaseDC(this->hwnd, this->hdc);
}

View::View(HWND hwnd, HDC hdc, int width, int Rpen, int Gpen, int Bpen, int Rbrush, int Gbrush, int Bbrush)
{
	this->hwnd = hwnd;
	this->hdc = hdc;
	this->width = width;
	if (isWidth(this->width) == false)
		throw exception("Такую толщину линий задать нельзя!\nМаксимальная толщина - 9 пикселей.");
	this->Rpen = Rpen;
	this->Gpen = Gpen;
	this->Bpen = Bpen;
	if (isRGB(this->Rpen, this->Gpen, this->Bpen) == false)
		throw exception("Такого цвета не существует!");
	this->Rbrush = Rbrush;
	this->Gbrush = Gbrush;
	this->Bbrush = Bbrush;
	if (isRGB(this->Rbrush, this->Gbrush, this->Bbrush) == false)
		throw exception("Такого цвета не существует!");
	hPen = CreatePen(PS_SOLID, this->width, RGB(this->Rpen, this->Gpen, this->Bpen));
	hBrush = CreateSolidBrush(RGB(this->Rbrush, this->Gbrush, this->Bbrush));
}
```

```
void View::viewUnpaintQudrangle(Quadrangle *quadrangle)
{
	if (isAffiliation(quadrangle) == false)
		throw exception("Четырёхугольник выходит за границу экрана!");
	InvalidateRect(this->hwnd, 0, TRUE);
	UpdateWindow(this->hwnd);
	HPEN SelectPen(this->hdc, this->hPen);
	HBRUSH SelectBrush(this->hdc, GetStockBrush(NULL_BRUSH));
	POINT *points = quadrangle->getQuadrangle();
	Polygon(this->hdc, points, 4);
}

void View::viewPaintQudrangle(Quadrangle *quadrangle)
{
	if (isAffiliation(quadrangle) == false)
		throw exception("Четырёхугольник выходит за границу экрана!");
	InvalidateRect(this->hwnd, 0, TRUE);
	UpdateWindow(this->hwnd);
	HPEN SelectPen(this->hdc, this->hPen);
	HBRUSH SelectBrush(this->hdc, this->hBrush);
	POINT *points = quadrangle->getQuadrangle();
	Polygon(this->hdc, points, 4);
}

void View::viewTwoNestedPaintQuadrangle(Quadrangle *externalQuadrangle, Quadrangle *internalQuadrangle)
{
	if (isAffiliation(externalQuadrangle) == false)
		throw exception("Четырёхугольник выходит за границу экрана!");
	InvalidateRect(this->hwnd, 0, TRUE);
	UpdateWindow(this->hwnd);
	if (externalQuadrangle->isOnTheNested(internalQuadrangle) == false)
		throw exception("Внутренний четырёхугольник выходит за границы внешнего!");
	HPEN SelectPen(this->hdc, this->hPen);
	HBRUSH SelectBrush(this->hdc, GetStockBrush(NULL_BRUSH));
	POINT *points1 = externalQuadrangle->getQuadrangle();
	POINT *points2 = internalQuadrangle->getQuadrangle();
	Polygon(this->hdc, points1, 4);
	Polygon(this->hdc, points2, 4);
	HBRUSH SelectBrush(this->hdc, this->hBrush);
	FloodFill(this->hdc, points1[0].x + 10, points1[0].y + 10, RGB(this->Rpen, this->Gpen, this->Bpen));
}

void View::enterCharacteristicsFromFiles(string fd)
{
	ifstream in(fd, ios::in);
	if (in.is_open())
	{
		in >> this->width;
		if (isWidth(this->width) == false)
			throw exception("Такую толщину линий задать нельзя!\nМаксимальная толщина - 9 пикселей.");
		in >> this->Rpen;
		in >> this->Gpen;
		in >> this->Bpen;
		if (isRGB(this->Rpen, this->Gpen, this->Bpen) == false)
			throw exception("Такого цвета не существует!");
		in >> this->Rbrush;
		in >> this->Gbrush;
		in >> this->Bbrush;
		if (isRGB(this->Rbrush, this->Gbrush, this->Bbrush) == false)
			throw exception("Такого цвета не существует!");
		this->hPen = CreatePen(PS_SOLID, 5, RGB(this->Rpen, this->Gpen, this->Bpen));
		this->hBrush = CreateSolidBrush(RGB(this->Rbrush, this->Gbrush, this->Bbrush));
		in.close();
	}
	else
		throw exception("В этом файле нет нужных данных");
```

```
}

void View::saveCharacteristicsToFile(string fd)
{
	ofstream out(fd, ios::out);
	out << this->width;
	out << " ";
	out << this->Rpen;
	out << " ";
	out << this->Gpen;
	out << " ";
	out << this->Bpen;
	out << " ";
	out << this->Rbrush;
	out << " ";
	out << this->Gbrush;
	out << " ";
	out << this->Bbrush;
	out.close();
}

void View::establishNewWidth(int width)
{
	this->width = width;
	if ((this->width) == false)
		throw exception("Такую толщину линий задать нельзя!\nМаксимальная толщина - 9 пикселей.");
	setCharacteristicsView(this->width, Rpen, Gpen, Bpen, Rbrush, Gbrush, Bbrush);
}

void View::establishNewRGBPen(int Rpen, int Gpen, int Bpen)
{
	this->Rpen = Rpen;
	this->Gpen = Gpen;
	this->Bpen = Bpen;
	if (isRGB(this->Rpen, this->Gpen, this->Bpen) == false)
		throw exception("Такого цвета не существует!");
	setCharacteristicsView(width, this->Rpen, this->Gpen, this->Bpen, Rbrush, Gbrush, Bbrush);
}

void View::establishNewRGBBrush(int Rbrush, int Gbrush, int Bbrush)
{
	this->Rbrush = Rbrush;
	this->Gbrush = Gbrush;
	this->Bbrush = Bbrush;
	if (isRGB(this->Rbrush, this->Gbrush, this->Bbrush) == false)
		throw exception("Такого цвета не существует!");
	setCharacteristicsView(width, Rpen, Gpen, Bpen, this->Rbrush, this->Gbrush, this->Bbrush);
}


#include "Quadrangle.h"
#include "View.h"

int main()
{
	setlocale(LC_ALL, "rus");
	try {
		HWND hwnd = GetConsoleWindow();
		HDC hdc = GetDC(hwnd);
		int width = 5;
		int Rpen = 0, Gpen = 0, Bpen = 0;
		int Rbrush = 255, Gbrush = 0, Bbrush = 0;

		// Создаём три четырёхугольника
		Quadrangle first(300, 100, 400, 100, 500, 300, 300, 220);
		Quadrangle second(350, 150, 380, 150, 400, 220, 350, 200);
		Quadrangle third(0, 0, 0, 0, 0, 0, 0, 0);
```

```
		third.enterCoordinatesFromFiles("Quadrangle_CoordinatesFromFile.txt");

		// Вызываем конструктор с параметрами
		View quadrangle1(hwnd, hdc, width, Rpen, Gpen, Bpen, Rbrush, Gbrush, Bbrush);

		// Выводим изображения
		quadrangle1.viewUnpaintQudrangle(&first);
		getchar();
		quadrangle1.viewPaintQudrangle(&second);
		getchar();
		quadrangle1.viewTwoNestedPaintQuadrangle(&first, &second);
		getchar();

		// Выводим изображение двух вложенных четырёхугольников с новыми параметрами, полученными из
		файла
		quadrangle1.enterCharacteristicsFromFiles("Quadrangle_CharacteristicsFromFiles.txt");
		quadrangle1.viewTwoNestedPaintQuadrangle(&first, &second);
		getchar();

		// Демонстрация работы функции сдвига фигуры
		quadrangle1.viewPaintQudrangle(&third);
		getchar();
		third.changePosition(-50, 0);
		quadrangle1.viewPaintQudrangle(&third);
		getchar();
		third.saveCoordinatesToFile("Quadrangle_CoordinatesInFile.txt");
		quadrangle1.saveCharacteristicsToFile("Quadrangle_CharacteristicsInFiles.txt");

		// Демонстрация работы функций изменения геометрических и графических свойств фигуры
		View quadrangle2(hwnd, hdc, width, Rpen, Gpen, Bpen, Rbrush, Gbrush, Bbrush);
		Quadrangle fourth(300, 100, 400, 100, 500, 300, 300, 220);
		fourth.establishNewPosition(350, 100, 450, 100, 550, 300, 350, 220);
		quadrangle2.viewPaintQudrangle(&fourth);
		getchar();
		quadrangle2.establishNewWidth(1);
		quadrangle2.viewPaintQudrangle(&fourth);
		getchar();
		quadrangle2.establishNewRGBPen(0, 0, 255);
		quadrangle2.viewPaintQudrangle(&fourth);
		getchar();
		quadrangle2.establishNewRGBBrush(0, 255, 0);
		quadrangle2.viewPaintQudrangle(&fourth);
		getchar();
	}
	catch (exception exeption)
	{
		cout << exeption.what() << endl;
	}
	return 0;
}
```