

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 №1  
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 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. Разработать программу, демонстрирующую выполнение указанных функций. Ввод параметров фигур (координат и др.), параметров рисуемых линий и закраски осуществлять из файлов (отдельно для каждого теста). 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.

Метод решения: Запишем в файл необходимые фигуры в виде координат. При запуске программы запросим у пользователя тип фигуры, далее последует выбор одной из фигур, записанной в файле. Если пользователь запросит фигуру в фигуре, то так же выберем вторую фигуру. Проведем проверку фигур на выпуклость, нахождение на экране, проверку нахождения фигуры в фигуре. Запросим у пользователя цвет фигуры. Выведем фигуру в консоли.

## 2) Алгоритм:

```
Структура List
{
    целая x[4];
    целая y[4];
};

Структура Vector
{
    целая x;
    целая y;
};

Логическая функция Out_of_Window(целая right, целая bottom, целая x, целая y)
{
    Если (((right >= x) И (bottom >= y)) И ((0 <= x) И (0 <= y)))
    {
        Вернуть истину;
    }
    Вернуть ложь;
}

Логическая функция Test_Convex(Структура List figure)
{
    Вектор ab =
    {
        figure.x[1] - figure.x[0],
        figure.y[1] - figure.y[0]
    };

    Вектор bc =
    {
        figure.x[2] - figure.x[1],
        figure.y[2] - figure.y[1]
    };
}
```

```

Vector cd =
{
    figure.x[3] - figure.x[2],
    figure.y[3] - figure.y[2]
};

Вектор da =
{
    figure.x[0] - figure.x[3],
    figure.y[0] - figure.y[3]
};

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;

Если ((product1 >= 0) И (product2 >= 0) И (product3 >= 0) И (product4 >= 0))
    Вернуть истину;
иначе
    Вернуть ложь;
}

Целая функция AB_D(Целая x, Целая y, Целая xA, Целая yA, Целая xB, Целая yB)
{
    Вернуть (x - xA) * (yB - yA) - (y - yA) * (xB - xA);
}

Логическая функция Test_In_Figure(Структура List outside, Структура List inside)
{
    for (целая i(0); i < 4; i++)
    {
        for (целая j(0); j < 3; j++)
        {
            Если (AB_D(outside.x[i], inside.y[i], outside.x[j], outside.y[j],
outside.x[j + 1], outside.y[j + 1]) > 0)
                Вернуть ложь;
        }
    }
    Вернуть истину;
}

int main()
{
    Подключение русского языка

    Открыть файл("set.txt");
    Если (файл не открыт)
    {
        вывод << "Error 404" << endl;
        вернуть -1;
    }

    Беззнаковая целая count;
    ввод >> count;

    Если (count > 0)
    {
        List *Figure = new List[count];

        for (целая i(0); i < count; i++) //считывание из файла
            for (целая j(0); j < 4; j++)
            {
                ввод >> Figure[i].x[j];
            }
    }
}

```

```

        ввод >> Figure[i].y[j];
    }
Закрыть файл();

Получение контекста изображения
Определение размера окна

вывод << "Выберите функцию: " << endl;
вывод << endl << "1 - незакрашенная фигура(фигура-контур)" << endl;
вывод << "2 - закрашенная фигура" << endl;
вывод << "3 - две вложенные одна в другую фигуры" << endl;
беззнаковая целая Flag;
вывод << "|||||";
ввод >> Flag;

Если ((Flag == 1) или (Flag == 2))
{
    GetClientRect(hwnd, &rt);
    беззнаковая целая num(-1);
    пока ((0 > num) или (num >= count))
    {
        system("cls");
        вывод << "Доступны фигуры: ";
        for (целая i(1); i <= count; i++)
            вывод << i << " ";
        вывод << endl << "Введите номер фигуры: ";
        ввод >> num;
        num--;
    }

    Если (Test_Convex(Figure[num]))
    {
        логическая test(истина);
        for (целая i(0); (i < 4) И (test == истина); i++)
        {
            test = Out_of_Window(rt.right, rt.bottom,
Figure[num].x[i], Figure[num].y[i]);
        }
        Если (test == истина)
        {
            BOOL bxt;
            Если (Flag == 1)
            {
                вывод << "Введите цвет контура(RGB): " << endl;
                int R(0), G(0), B(0);
                вывод << "R = ";
                ввод >> R;
                вывод << endl << "G = ";
                ввод >> G;
                вывод << endl << "B = ";
                cin >> B;
                Выбрать перо(hdc, CreatePen(PS_SOLID, 3, RGB(R,
G, B)));
            do
            {
                system("cls");
                GetClientRect(hwnd, &rt);
                POINT points[5];
                points[0] = { Figure[num].x[0],
Figure[num].y[0] };
                points[1] = { Figure[num].x[1],
Figure[num].y[1] };
                points[2] = { Figure[num].x[2],
Figure[num].y[2] };

```

```

        points[3] = { Figure[num].x[3],
Figure[num].y[3] };
        points[4] = { Figure[num].x[0],
Figure[num].y[0] };
        bxt = Polyline(hdc, points, 5);
    } Пока (_getch() != 27);
}
Иначе
{
    вывод << "Введите цвет фигуры(RGB): " << endl;
    целая R(0), G(0), B(0);
    вывод << "R = ";
    ввод >> R;
    вывод << endl << "G = ";
    ввод >> G;
    вывод << endl << "B = ";
    ввод >> B;
    Выбрать перо(hdc, CreatePen(PS_SOLID, 3, RGB(R,
G, B)));
    Выбрать кисть(hdc, CreateSolidBrush(RGB(R, G,
B)));
    do
    {
        system("cls");
        GetClientRect(hwnd, &rt);
        POINT points[4];
        points[0] = { Figure[num].x[0],
Figure[num].y[0] };
        points[1] = { Figure[num].x[1],
Figure[num].y[1] };
        points[2] = { Figure[num].x[2],
Figure[num].y[2] };
        points[3] = { Figure[num].x[3],
Figure[num].y[3] };
        bxt = Polygon(hdc, points, 4);
    } Пока (_getch() != 27);
};
}
Иначе
{
    вывод << "Фигура выходит за края экрана!";
}
}
Иначе
{
    вывод << "Фигура не выпуклая!";
}
}
Иначе если (Flag == 3)
{
    Беззнаковая целая num1(-1);
    пока ((0 > num1) или (num1 >= count))
    {
        system("cls");
        вывод << "Доступны фигуры: ";
        for (целая i(1); i <= count; i++)
            вывод << i << " ";
        вывод << endl << "Введите номер внешней фигуры: ";
        ввод >> num1;
        num1--;
    }
    Беззнаковая целая num2(-1);
    Пока ((0 > num2) или (num2 >= count) и (num2 != num1))
    {
        system("cls");
    }
}

```

```

    вывод << "Доступны фигуры: ";
    for (целая i(1); i <= count; i++)
        Если ((num1 + 1) != i)
            вывод << i << " ";
    вывод << endl << "Введите номер внутренней фигуры: ";
    ввод >> num2;
    num2--;
}

GetClientRect(hwnd, &rt);
логическая test1(истина);
логическая test2(истина);
for (целая i(0); (i < 4) и (test1 == истина); i++)
{
    test1 = Out_of_Window(rt.right, rt.bottom, Figure[num1].x[i],
Figure[num1].y[i]);
}
for (целая i(0); (i < 4) && (test2 == истина); i++)
{
    test2 = Out_of_Window(rt.right, rt.bottom, Figure[num2].x[i],
Figure[num2].y[i]);
}
если (test1 * test2 == истина)
{
    Если (Test_Convex(Figure[num1]) && Test_Convex(Figure[num2])
== истина)
    {
        Если (Test_In_Figure(Figure[num1], Figure[num2]))
        {
            BOOL bxt;
            cout << "Введите цвет фигуры-1 (RGB): " << endl;
            целая R(0), G(0), B(0);
            вывод << "R = ";
            ввод >> R;
            вывод << endl << "G = ";
            ввод >> G;
            вывод << endl << "B = ";
            ввод >> B;
            HPEN hPen1 = Создать перо(PS_SOLID, 3, RGB(R, G,
B));
            HBRUSH hBrush1 = Создать кисть(RGB(R, G, B));
            system("cls");

            вывод << "Введите цвет фигуры-2 (RGB): " <<
endl;
            вывод << "R = ";
            ввод >> R;
            вывод << endl << "G = ";
            ввод >> G;
            вывод << endl << "B = ";
            ввод >> B;
            HPEN hPen2 = Создать перо(PS_SOLID, 3, RGB(R, G,
B));
            HBRUSH hBrush2 = Создать кисть(RGB(R, G, B));
            HBRUSH hBrush3 = Выбрать кисть(BLACK_BRUSH);
            do
            {
                system("cls");
                GetClientRect(hwnd, &rt);
                вывод << rt.right << "x" << rt.bottom <<
endl;
                POINT points1[4];
                points1[0] = { Figure[num1].x[0],
Figure[num1].y[0] };

```

```

    Figure[num1].y[1] };
    Figure[num1].y[2] };
    Figure[num1].y[3] };

    Figure[num2].y[0] };
    Figure[num2].y[1] };
    Figure[num2].y[2] };
    Figure[num2].y[3] };

    points1[1] = { Figure[num1].x[1],
    points1[2] = { Figure[num1].x[2],
    points1[3] = { Figure[num1].x[3],
    Выбрать перо(hdc, hPen1);
    Выбрать кисть(hdc, hBrush3);
    bxt = Polygon(hdc, points1, 4);

    POINT points2[4];
    points2[0] = { Figure[num2].x[0],
    points2[1] = { Figure[num2].x[1],
    points2[2] = { Figure[num2].x[2],
    points2[3] = { Figure[num2].x[3],
    Выбрать перо ( hdc, hPen2 );
    Выбрать кисть ( hdc, hBrush2 );
    bxt = Polygon(hdc, points2, 4);
    } пока (_getch() != 27);

    }
    иначе
    {
        вывод << "Внутренняя фигура выходит за края
внешней фигуры!";
    }
}

иначе
{
    вывод << "Фигура(ы) не выпуклая(ые)!";
}
иначе
{
    вывод << "Фигура выходит за края экрана!";
}
}

Удалить Figure;
}

Иначе
{
    вывод << "Файл пуст!";
}

вернуть 0;
}

```

### 3) Код программы:

```

#include <iostream>
#include <windows.h>
#include <windowsx.h>
#include <fstream>
#include <conio.h>

using namespace std;

struct List
{
    int x[4];

```

```

        int y[4];
};

struct Vector
{
    int x;
    int y;
};

bool Out_of_Window(int right, int bottom, int x, int y)
{

    if (((right >= x) && (bottom >= y)) && ((0 <= x) && (0 <= y)))
    {
        return true;
    }
    return false;
}

bool Test_Convex(struct List figure)
{
    Vector ab =
    {
        figure.x[1] - figure.x[0],
        figure.y[1] - figure.y[0]
    };

    Vector bc =
    {
        figure.x[2] - figure.x[1],
        figure.y[2] - figure.y[1]
    };

    Vector cd =
    {
        figure.x[3] - figure.x[2],
        figure.y[3] - figure.y[2]
    };

    Vector da =
    {
        figure.x[0] - figure.x[3],
        figure.y[0] - figure.y[3]
    };

    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;

    if ((product1 >= 0) && (product2 >= 0) && (product3 >= 0) && (product4 >= 0))
        return true;
    else
        return false;
}

int AB_D(int x, int y, int xA, int yA, int xB, int yB)
{
    return (x - xA) * (yB - yA) - (y - yA) * (xB - xA);
}

bool Test_In_Figure(struct List outside, struct List inside)
{
    for (int i(0); i < 4; i++)
    {

```

```

        for (int j(0); j < 3; j++)
        {
            if (AB_D(inside.x[i], inside.y[i], outside.x[j], outside.y[j],
outside.x[j + 1], outside.y[j + 1]) > 0)
                return false;
        }
    }

    return true;
}

int main()
{
    setlocale(LC_ALL, "rus");

    ifstream fin;
    fin.open("set.txt");
    if (!fin.is_open())
    {
        cout << "Error 404" << endl;
        return -1;
    }

    unsigned int count;
    fin >> count;

    if (count > 0)
    {
        List *Figure = new List[count];

        for (int i(0); i < count; i++)//считывание из файла
            for (int j(0); j < 4; j++)
            {
                fin >> Figure[i].x[j];
                fin >> Figure[i].y[j];
            }
        fin.close();

        HWND hwnd = GetConsoleWindow();
        HDC hdc = GetDC(hwnd);
        RECT rt;
        GetClientRect(hwnd, &rt);

        cout << "Выберите функцию: " << endl;
        cout << endl << "1 - незакрашенная фигура(фигура-контур)" << endl;
        cout << "2 - закрашенная фигура" << endl;
        cout << "3 - две вложенные одна в другую фигуры" << endl;
        unsigned int Flag;
        cout << ">>>>";
        cin >> Flag;

        if ((Flag == 1) || (Flag == 2))
        {
            GetClientRect(hwnd, &rt);
            unsigned int num(-1);
            while ((0 > num) || (num >= count))
            {
                system("cls");
                cout << "Доступны фигуры: ";
                for (int i(1); i <= count; i++)
                    cout << i << " ";
                cout << endl << "Введите номер фигуры: ";
                cin >> num;
                num--;
            }
        }
    }
}

```

```

        }

        if (Test_Convex(Figure[num]))
        {
            bool test(true);
            for (int i(0); (i < 4) && (test == true); i++)
            {
                test = Out_of_Window(rt.right, rt.bottom,
Figure[num].x[i], Figure[num].y[i]);
            }
            if (test == true)
            {
                BOOL bxt;
                if (Flag == 1)
                {
                    cout << "Введите цвет контура(RGB): " << endl;
                    int R(0), G(0), B(0);
                    cout << "R = ";
                    cin >> R;
                    cout << endl << "G = ";
                    cin >> G;
                    cout << endl << "B = ";
                    cin >> B;
                    SelectPen(hdc, CreatePen(PS_SOLID, 3, RGB(R, G,
B)));
                }
                do
                {
                    system("cls");
                    GetClientRect(hwnd, &rt);
                    POINT points[5];
                    points[0] = { Figure[num].x[0],
Figure[num].y[0] };
                    points[1] = { Figure[num].x[1],
Figure[num].y[1] };
                    points[2] = { Figure[num].x[2],
Figure[num].y[2] };
                    points[3] = { Figure[num].x[3],
Figure[num].y[3] };
                    points[4] = { Figure[num].x[0],
Figure[num].y[0] };
                    bxt = Polyline(hdc, points, 5);
                } while (_getch() != 27);
            }
            else
            {
                cout << "Введите цвет фигуры(RGB): " << endl;
                int R(0), G(0), B(0);
                cout << "R = ";
                cin >> R;
                cout << endl << "G = ";
                cin >> G;
                cout << endl << "B = ";
                cin >> B;
                SelectPen(hdc, CreatePen(PS_SOLID, 3, RGB(R, G,
B)));
                SelectBrush(hdc, CreateSolidBrush(RGB(R, G,
B)));
            }
            do
            {
                system("cls");
                GetClientRect(hwnd, &rt);
                POINT points[4];
                points[0] = { Figure[num].x[0],
Figure[num].y[0] };

```

```

        points[1] = { Figure[num].x[1],
Figure[num].y[1] };
        points[2] = { Figure[num].x[2],
Figure[num].y[2] };
        points[3] = { Figure[num].x[3],
Figure[num].y[3] };
        bxt = Polygon(hdc, points, 4);
    } while (_getch() != 27);
}
}
else
{
    cout << "Фигура выходит за края экрана!";
}
else
{
    cout << "Фигура не выпуклая!";
}
}
else if (Flag == 3)
{
    unsigned int num1(-1);
    while ((0 > num1) || (num1 >= count))
    {
        system("cls");
        cout << "Доступны фигуры: ";
        for (int i(1); i <= count; i++)
            cout << i << " ";
        cout << endl << "Введите номер внешней фигуры: ";
        cin >> num1;
        num1--;
    }
    unsigned int num2(-1);
    while ((0 > num2) || (num2 >= count) && (num2 != num1))
    {
        system("cls");
        cout << "Доступны фигуры: ";
        for (int i(1); i <= count; i++)
            if ((num1 + 1) != i)
                cout << i << " ";
        cout << endl << "Введите номер внутренней фигуры: ";
        cin >> num2;
        num2--;
    }

GetClientRect(hwnd, &rt);
bool test1(true);
bool test2(true);
for (int i(0); (i < 4) && (test1 == true); i++)
{
    test1 = Out_of_Window(rt.right, rt.bottom, Figure[num1].x[i],
Figure[num1].y[i]);
}
for (int i(0); (i < 4) && (test2 == true); i++)
{
    test2 = Out_of_Window(rt.right, rt.bottom, Figure[num2].x[i],
Figure[num2].y[i]);
}
if (test1 * test2 == true)
{
    if (Test_Convex(Figure[num1]) && Test_Convex(Figure[num2]) ==
true)
    {
        if (Test_In_Figure(Figure[num1], Figure[num2]))

```

```

    {
        BOOL bxt;
        cout << "Введите цвет фигуры-1 (RGB): " << endl;
        int R(0), G(0), B(0);
        cout << "R = ";
        cin >> R;
        cout << endl << "G = ";
        cin >> G;
        cout << endl << "B = ";
        cin >> B;
        HPEN hPen1 = CreatePen(PS_SOLID, 3, RGB(R, G,
        B));
        HBRUSH hBrush1 = CreateSolidBrush(RGB(R, G, B));
        system("cls");

        cout << "Введите цвет фигуры-2 (RGB): " << endl;
        cout << "R = ";
        cin >> R;
        cout << endl << "G = ";
        cin >> G;
        cout << endl << "B = ";
        cin >> B;
        HPEN hPen2 = CreatePen(PS_SOLID, 3, RGB(R, G,
        B));
        HBRUSH hBrush2 = CreateSolidBrush(RGB(R, G, B));
        HBRUSH hBrush3 = GetStockBrush(BLACK_BRUSH);

        do
        {
            system("cls");
            GetClientRect(hwnd, &rt);
            cout << rt.right << "x" << rt.bottom <<
            endl;

            Figure[num1].y[0] ;
            Figure[num1].y[1] ;
            Figure[num1].y[2] ;
            Figure[num1].y[3] ;

            Figure[num1].y[0] ;
            Figure[num1].y[1] ;
            Figure[num1].y[2] ;
            Figure[num1].y[3] ;

            Figure[num2].y[0] ;
            Figure[num2].y[1] ;
            Figure[num2].y[2] ;
            Figure[num2].y[3] ;

            Figure[num2].y[0] ;
            Figure[num2].y[1] ;
            Figure[num2].y[2] ;
            Figure[num2].y[3] ;

            SelectPen(hdc, hPen1);
            SelectBrush(hdc, hBrush1);
            bxt = Polygon(hdc, points1, 4);

            POINT points2[4];
            points2[0] = { Figure[num2].x[0],
            points2[1] = { Figure[num2].x[1],
            points2[2] = { Figure[num2].x[2],
            points2[3] = { Figure[num2].x[3],
            SelectPen(hdc, hPen2);
            SelectBrush(hdc, hBrush3);
            bxt = Polygon(hdc, points2, 4);
            } while (_getch() != 27);
        }
        else
        {
            cout << "Внутренняя фигура выходит за края
            внешней фигуры!";
        }
    }

```

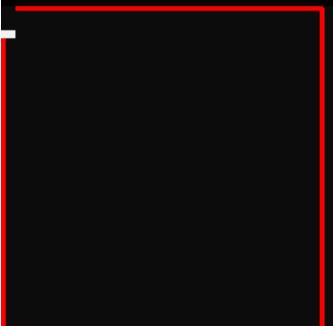
```

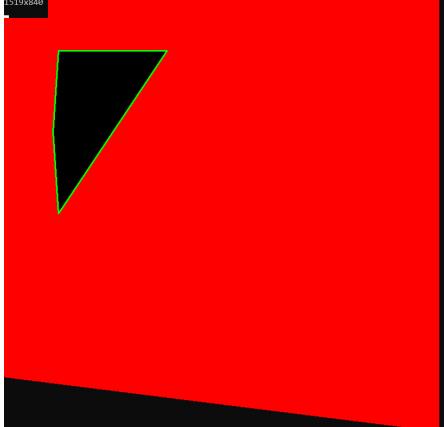
        }
    else
    {
        cout << "Фигура(ы) не выпуклая(ые)!";
    }
}
else
{
    cout << "Фигура выходит за края экрана!";
}
delete[] Figure;
}
else
{
    cout << "Файл пуст!";
}

return 0;
}

```

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	<pre> 0 0 800 0 800 800 0 700  100 100 300 100 100 400 90 250 </pre>		Фигура в фигуре
6	<pre> 100 100 300 100 150 250 175 400 </pre>	Фигура не выпуклая!	Четырехугольник невыпуклый
7	<pre> 100 100 300 100 100 400 90 250  0 0 400 0 400 400 0 400 </pre>	Внутренняя фигура выходит за края внешней фигуры!	Фигура не в фигуре

5) Программа работает правильно, что подтверждают тесты.