Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные среды и системное программирование

# ОТЧЁТ
к лабораторной работе №1
на тему

# Основы программирования в Win 32 API. Оконное приложение Win 32 с минимальной достаточной функциональностью. Обработка основных оконных сообщений.

Студент: гр.153502
Сачивко В.Г.

Проверил: Гриценко Н.Ю.

Минск 2023

# СОДЕРЖАНИЕ

# 1 ЦЕЛЬ РАБОТЫ

Целью выполнения лабораторной работы является создание оконного приложения на Win32 API, обладающее минимальным функционалом, позволяющим отработать базовые навыки написания программы на Win32 API, таких как обработка оконных сообщений.

В качестве задачи необходимо разработать оконное приложение, которое позволяет пользователю рисовать и редактировать графические фигуры (круги, прямоугольники) с помощью мыши и клавиш клавиатуры.

# 2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Win32 API (Application Programming Interface) – это набор функций и процедур, предоставляемых операционной системой Windows для разработки приложений на языке программирования C/C++. Оконное приложение Win32 – это приложение, которое состоит из одного или нескольких окон, в которых происходит взаимодействие с пользователем. Для создания окна необходимо зарегистрировать класс окна с помощью функции RegisterClassEx и создать окно с помощью функции CreateWindowEx. Окно может иметь различные свойства, такие как заголовок, размеры, стиль и обработчики сообщений. Важным аспектом программирования в Win32 API является обработка оконных сообщений. Оконные сообщения – это события, которые происходят в окне, например, нажатие кнопки мыши или клавиши, изменение размера окна и другие действия пользователя. Для обработки оконных сообщений необходимо определить функцию оконной процедуры (WndProc), которая будет вызываться системой при возникновении сообщения. В функции WndProc нужно обрабатывать различные типы сообщений с помощью условных операторов и выполнять соответствующие действия.

# 2 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

Согласно формулировке задачи, были спроектированы следующие функции программы:

– Создание фигур (кругов, прямоугольников);

– Редактирование фигур.

## 1. Создание фигур

Создание фигур (рисунок 1) доступно изначально при запуске программы. Для создания фигуры необходимо зажать ЛКМ и провести желаемую фигуру, после отпустить ЛКМ. Для выбора другой фигуры необходимо нажать пробел.

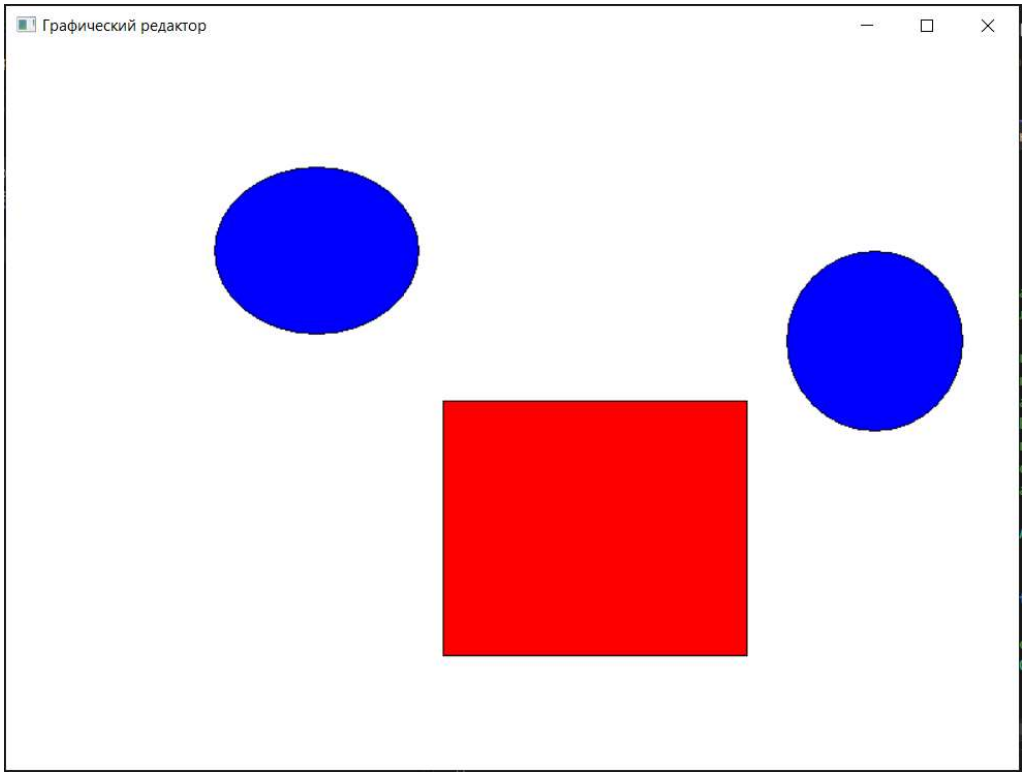


Рисунок 1 – Режим создания фигур

## 2. Редактирование фигур

Для изменения фигуры необходимо навести курсор рядом с границей фигуры и зажать ЛКМ, после чего навести курсор в желаемое положение и отпустить ЛКМ (см. рисунок 2). Для изменения положения фигуры необходимо навести курсор ближе к центру и зажать ЛКМ, после чего фигуру можно переместить в другое место (см. рисунок 3).

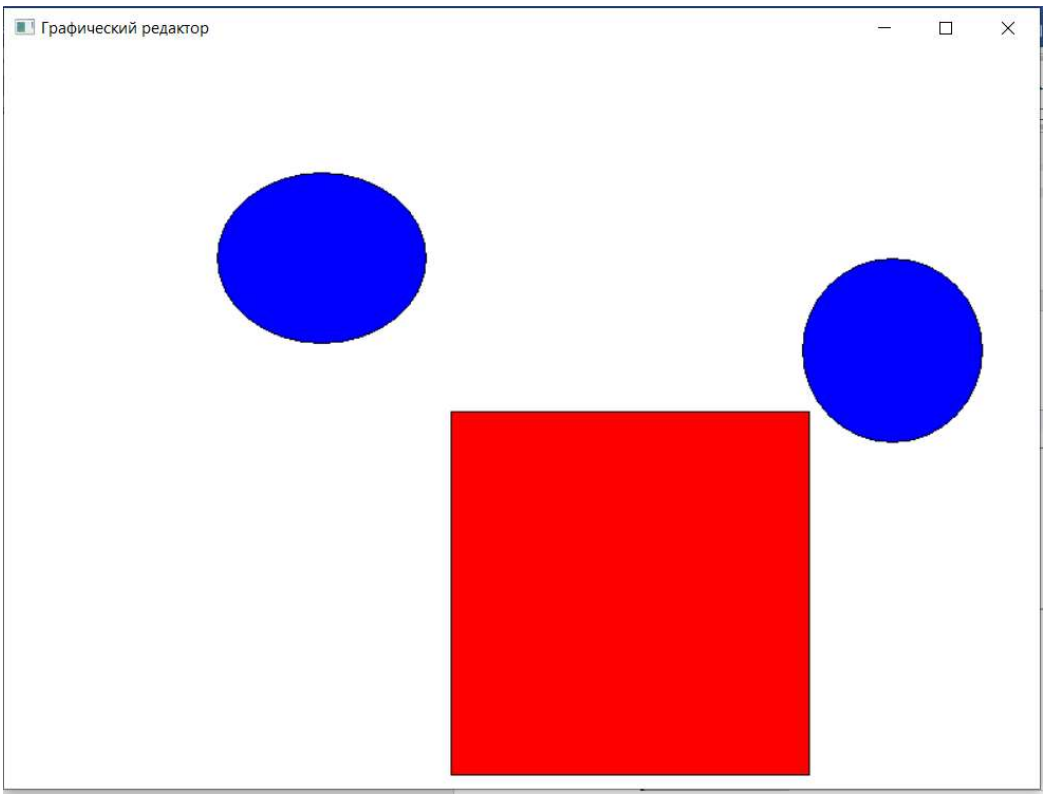


Рисунок 2 – Результат изменения размера фигуры

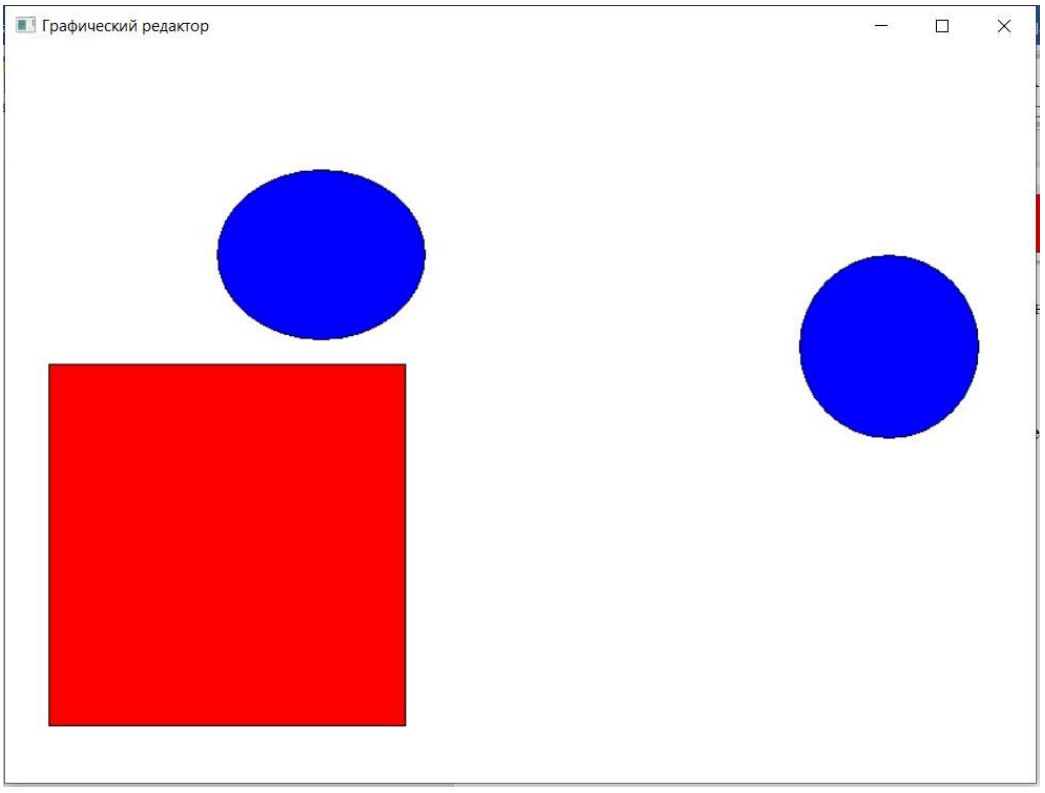


Рисунок 3 – Результат изменения положения фигуры

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Начало работы с классическими приложениями для Windows, которые используют API Win32 [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/desktop-programming

# ПРИЛОЖЕНИЕ А
## Исходный код программы
## Файл Lab1.cpp

```
#include <windows.h>
#include <vector>
#include <utility>
HINSTANCE hInst;
HWND hWnd;
bool isDrawing = false;
bool isRectangleMode = false;
POINT startPoint;
POINT endPoint;
RECT currentRectangle;
bool currentIsRectangle;
std::vector<std::pair<RECT, bool>> figures;
bool isDragging = false;
int selectedFigureIndex = -1;
bool isResizing = false;
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
void DrawFigures(HDC hdc);
void DrawEllipse(HDC hdc, const RECT& rect);
bool IsCursorOnBorder(const POINT& cursor, const RECT& rect);
// Основная функция WinMain, точка входа в приложение
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
LPSTR lpCmdLine, int nCmdShow)
{
    hInst = hInstance;
    WNDCLASSEX wcex;
    wcex.cbSize = sizeof(WNDCLASSEX);
    wcex.style = CS_HREDRAW | CS_VREDRAW;
    wcex.lpfnWndProc = WndProc;
    wcex.cbClsExtra = 0;
    wcex.cbWndExtra = 0;
    wcex.hInstance = hInstance;
    wcex.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wcex.hCursor = LoadCursor(NULL, IDC_ARROW);
    wcex.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
    wcex.lpszMenuName = NULL;
    wcex.lpszClassName = L"GraphicsEditor";
    wcex.hIconSm = LoadIcon(NULL, IDI_APPLICATION);
    if (!RegisterClassEx(&wcex))
    {
```

```
        MessageBox(NULL, L"Не удалось зарегистрировать класс окна",
        L"Ошибка", MB_ICONERROR);
        return 1;
    }
    hWnd = CreateWindow(L"GraphicsEditor", L"Графический редактор",
        WS_OVERLAPPEDWINDOW, CW_USEDEFAULT,
        CW_USEDEFAULT, 800, 600, NULL, NULL, hInstance, NULL);
    if (!hWnd)
    {
        MessageBox(NULL, L"Не удалось создать окно", L"Ошибка",
MB_ICONERROR);
        return 1;
    }
    ShowWindow(hWnd, nCmdShow);
    UpdateWindow(hWnd);
    MSG msg;
    while (GetMessage(&msg, NULL, 0, 0))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    return (int)msg.wParam;
}
// Функция обработки сообщений
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM
wParam, LPARAM lParam)
{
    switch (message)
    {
    case WM_PAINT:
    {
        PAINTSTRUCT ps;
        HDC hdc = BeginPaint(hWnd, &ps);
        DrawFigures(hdc);
        EndPaint(hWnd, &ps);
    }
    break;
    case WM_LBUTTONDOWN:
        isDrawing = true;
        startPoint.x = LOWORD(lParam);
        startPoint.y = HIWORD(lParam);
        endPoint = startPoint;
        currentIsRectangle = isRectangleMode;
        for (int i = 0; i < figures.size(); ++i)
```

```
        {
            if (PtInRect(&figures[i].first, startPoint))
            {
                isDragging = true;
                selectedFigureIndex = i;
                if (IsCursorOnBorder(startPoint, figures[i].first))
                {
                    isResizing = true;
                }
                break;
            }
        }
        break;
    case WM_MOUSEMOVE:
        if (isDrawing)
        {
            endPoint.x = LOWORD(lParam);
            endPoint.y = HIWORD(lParam);
            currentRectangle.left = startPoint.x;
            currentRectangle.top = startPoint.y;
            currentRectangle.right = endPoint.x;
            currentRectangle.bottom = endPoint.y;
            HDC hdc = GetDC(hWnd);
            DrawFigures(hdc);
            if (isRectangleMode)
            {
                HBRUSH hBrush = CreateSolidBrush(RGB(255, 0, 0));
                SelectObject(hdc, hBrush);
                Rectangle(hdc, currentRectangle.left, currentRectangle.top,
                    currentRectangle.right, currentRectangle.bottom);
                DeleteObject(hBrush);
            }
            else
                DrawEllipse(hdc, currentRectangle);
            ReleaseDC(hWnd, hdc);
        }
        if (isDragging && selectedFigureIndex != -1)
        {
            endPoint.x = LOWORD(lParam);
            endPoint.y = HIWORD(lParam);
            // Обновляем координаты выбранной фигуры для перетаскивания
            if (!isResizing)
            {
                int deltaX = endPoint.x - startPoint.x;
```

```
            int deltaY = endPoint.y - startPoint.y;
            figures[selectedFigureIndex].first.left += deltaX;
            figures[selectedFigureIndex].first.right += deltaX;
            figures[selectedFigureIndex].first.top += deltaY;
            figures[selectedFigureIndex].first.bottom += deltaY;
        }
        else
        {
            // Изменение размера фигуры
            figures[selectedFigureIndex].first.right = endPoint.x;
            figures[selectedFigureIndex].first.bottom = endPoint.y;
        }
        startPoint = endPoint;
        InvalidateRect(hWnd, NULL, TRUE);
    }
    break;
case WM_LBUTTONUP:
    if (isDrawing)
    {
        isDrawing = false;
        endPoint.x = LOWORD(lParam);
        endPoint.y = HIWORD(lParam);
        currentRectangle.left = startPoint.x;
        currentRectangle.top = startPoint.y;
        currentRectangle.right = endPoint.x;
        currentRectangle.bottom = endPoint.y;
        figures.push_back(std::make_pair(currentRectangle, currentIsRectangle));
        InvalidateRect(hWnd, NULL, TRUE);
    }
    if (isDragging)
    {
        isDragging = false; // Завершаем перетаскивание
        isResizing = false; // Завершаем изменение размера
        selectedFigureIndex = -1; // Отменяем выбор фигуры
    }
    break;
case WM_KEYDOWN:
    if (wParam == VK_SPACE)
    {
        isRectangleMode = !isRectangleMode;
    }
    break;
case WM_DESTROY:
    PostQuitMessage(0);
```

```
        break;
    default:
        return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}
// Функция для отрисовки фигур
void DrawFigures(HDC hdc)
{
    for (int i = 0; i < figures.size(); ++i)
    {
        if (figures[i].second) // Если фигура - прямоугольник
        {
            HBRUSH hBrush = CreateSolidBrush(RGB(255, 0, 0));
            SelectObject(hdc, hBrush);
                Rectangle(hdc, figures[i].first.left, figures[i].first.top,
                figures[i].first.right, figures[i].first.bottom);
            DeleteObject(hBrush);
        }
        else // Если фигура - круг
        {
            DrawEllipse(hdc, figures[i].first);
        }
    }
}
// Функция для отрисовки круга
void DrawEllipse(HDC hdc, const RECT& rect)
{
    HBRUSH hBrush = CreateSolidBrush(RGB(0, 0, 255));
    SelectObject(hdc, hBrush);
    Ellipse(hdc, rect.left, rect.top, rect.right, rect.bottom);
    DeleteObject(hBrush);
}
// Функция для проверки, находится ли курсор на границе фигуры
bool IsCursorOnBorder(const POINT& cursor, const RECT& rect)
{
    const int borderSize = 15; // Размер области для изменения размера
    if (cursor.x >= rect.left - borderSize && cursor.x <= rect.left + borderSize)
        return true; // Левая граница
    if (cursor.x >= rect.right - borderSize && cursor.x <= rect.right + borderSize)
        return true; // Правая граница
    if (cursor.y >= rect.top - borderSize && cursor.y <= rect.top + borderSize)
        return true; // Верхняя граница
```

```
    if (cursor.y >= rect.bottom - borderSize && cursor.y <= rect.bottom +
borderSize)
        return true; // Нижняя граница
    return false;
}
```