Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные системы и системное программирование

Отчёт

к лабораторной работе

на тему

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

Студент: гр.153502

Даниленко И.А.

Проверил: Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[Цель работы 2](#_gjdgxs)

[Теоретические сведения 3](#_30j0zll)

[Результат выполнения программы 3](#_1fob9te)

[Список использованных источников 4](#_3znysh7)

[Приложение А 5](#_2et92p0)

# **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 удалять 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 RegisterClassExW и создать окно с помощью функции CreateWindowExW. 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.

# 

# **3 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ ПРОГРАММЫ**

Создание фигур доступно сразу при запуске программы. Чтобы нарисовать прямоугольник, необходимо зажать ЛКМ, провести фигуру и отпустить ЛКМ. Для того, чтобы нарисовать круг, нужно сперва зажать клавишу Shift, повторить предыдущие действия и отпустить Shift.

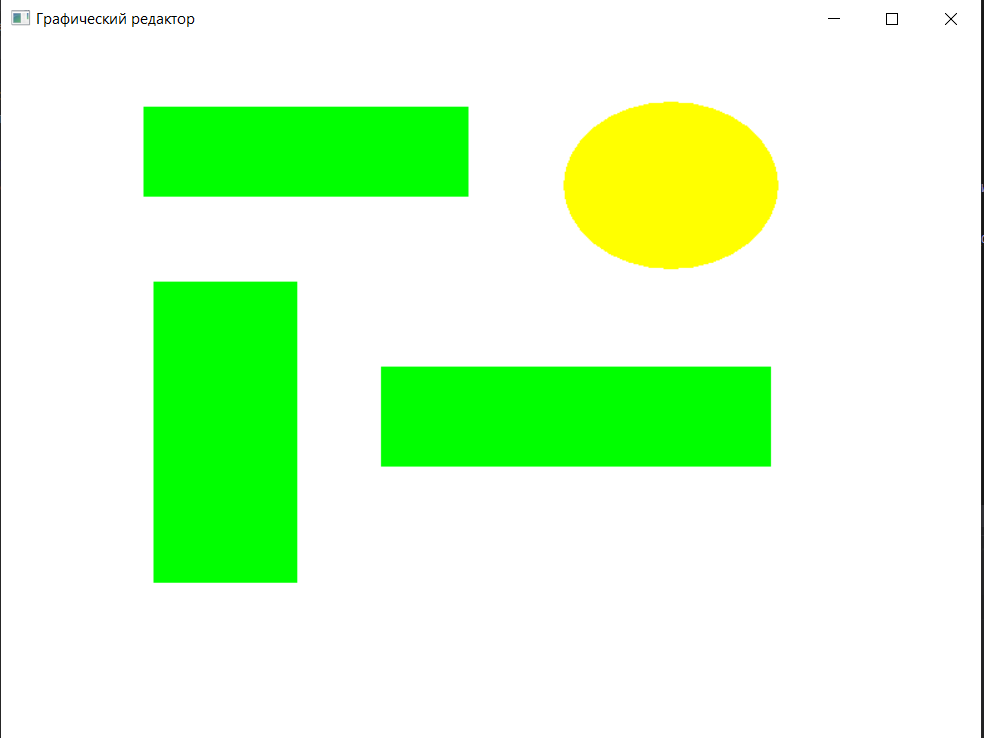


Рисунок 1 – Создание фигуры

Кроме рисования фигур, в программе доступно удаление последней нарисованной фигуры. Для этого нужно нажать ПКМ по фигуре.

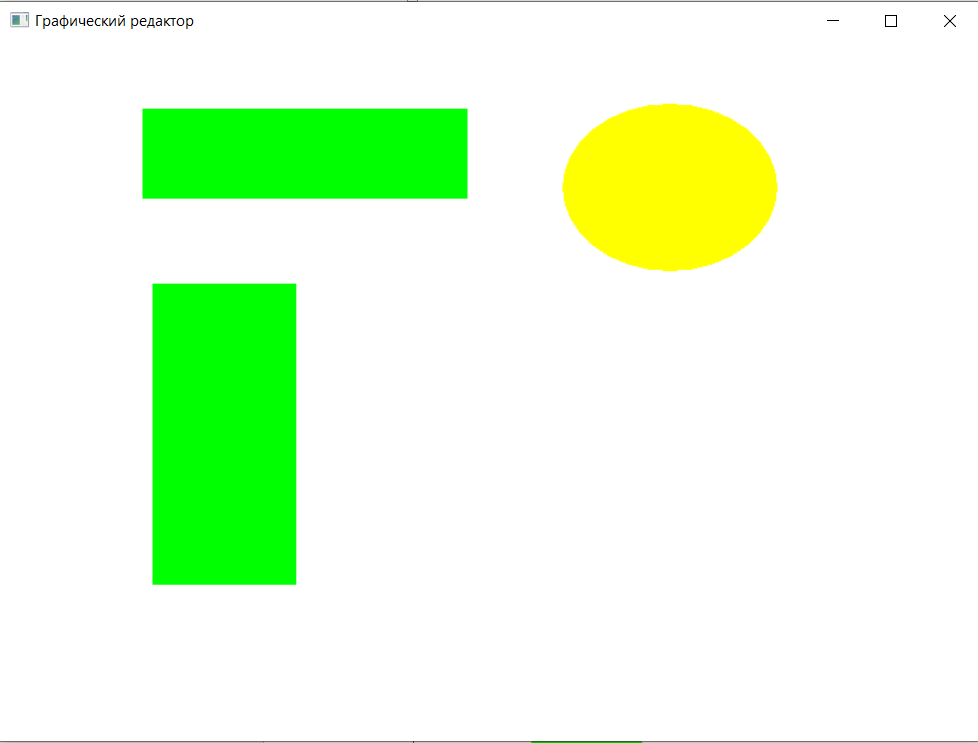


Рисунок 1 – Удаление фигуры

# **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

[2] Н.А. Литвиненко, Технология программирования на С++ Win-32 API-приложения, 2010

# **ПРИЛОЖЕНИЕ А**

**Исходный код программы**

#include <windows.h>

#include <tchar.h>

#include <vector>

#include <algorithm>

struct Shape {

int startX, startY, endX, endY;

bool isEllipse;

Shape(int x1, int y1, int x2, int y2, bool ellipse)

: startX(x1), startY(y1), endX(x2), endY(y2), isEllipse(ellipse) {}

};

std::vector<Shape> shapes; // Вектор для хранения фигур

int selectedShapeIndex = -1; // Индекс выбранной фигуры

LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam);

HBRUSH yellowBrush = NULL;

HPEN yellowPen = NULL;

HBRUSH greenBrush = NULL;

HPEN greenPen = NULL;

bool isEllipse = false;

int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPWSTR lpCmdLine, int nCmdShow) {

const wchar\_t\* className = L"MyWindowClass";

WNDCLASSEXW wc = { 0 };

wc.cbSize = sizeof(WNDCLASSEXW);

wc.style = CS\_HREDRAW | CS\_VREDRAW;

wc.lpfnWndProc = WndProc;

wc.hInstance = hInstance;

wc.hCursor = LoadCursor(NULL, IDC\_ARROW);

wc.hbrBackground = (HBRUSH)(COLOR\_WINDOW + 1);

wc.lpszClassName = className;

if (!RegisterClassExW(&wc)) {

MessageBoxW(NULL, L"Window Registration Failed!", L"Error!", MB\_ICONEXCLAMATION | MB\_OK);

return 0;

}

HWND hwnd = CreateWindowExW(0, className, L"Графический редактор", WS\_OVERLAPPEDWINDOW, CW\_USEDEFAULT, CW\_USEDEFAULT, 800, 600, NULL, NULL, hInstance, NULL);

if (hwnd == NULL) {

MessageBoxW(NULL, L"Window Creation Failed!", L"Error!", MB\_ICONEXCLAMATION | MB\_OK);

return 0;

}

ShowWindow(hwnd, nCmdShow);

UpdateWindow(hwnd);

MSG Msg;

while (GetMessageW(&Msg, NULL, 0, 0) > 0) {

TranslateMessage(&Msg);

DispatchMessageW(&Msg);

}

return Msg.wParam;

}

LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam) {

HDC hdc;

PAINTSTRUCT ps;

static int startX, startY, endX, endY;

static bool isDrawing = false;

switch (msg) {

case WM\_LBUTTONDOWN:

startX = LOWORD(lParam);

startY = HIWORD(lParam);

isDrawing = true;

break;

case WM\_LBUTTONUP:

endX = LOWORD(lParam);

endY = HIWORD(lParam);

isDrawing = false;

// Определяем, какую фигуру рисовать

isEllipse = (GetAsyncKeyState(VK\_SHIFT) & 0x8000) != 0;

// Сохраняем фигуру в векторе

shapes.push\_back(Shape(startX, startY, endX, endY, isEllipse));

// Вызываем функцию для перерисовки окна

InvalidateRect(hwnd, NULL, TRUE);

break;

case WM\_MOUSEMOVE:

if (isDrawing) {

// Обновляем координаты конечной точки при движении мыши

endX = LOWORD(lParam);

endY = HIWORD(lParam);

// Перерисовываем окно в режиме реального времени

hdc = GetDC(hwnd);

// Определяем, какую фигуру рисовать

isEllipse = (GetAsyncKeyState(VK\_SHIFT) & 0x8000) != 0;

if (isEllipse) {

Ellipse(hdc, startX, startY, endX, endY);

}

else {

Rectangle(hdc, startX, startY, endX, endY);

}

ReleaseDC(hwnd, hdc);

}

selectedShapeIndex = -1;

for (int i = static\_cast<int>(shapes.size()) - 1; i >= 0; --i) {

const Shape& shape = shapes[i];

if (endX >= shape.startX && endX <= shape.endX && endY >= shape.startY && endY <= shape.endY) {

// Мышь находится над фигурой

selectedShapeIndex = i;

break;

}

}

break;

case WM\_RBUTTONDOWN:

// Удаляем фигуру, над которой находится мышь

if (selectedShapeIndex != -1 && selectedShapeIndex < static\_cast<int>(shapes.size())) {

shapes.erase(shapes.begin() + selectedShapeIndex);

selectedShapeIndex = -1; // Очищаем выбранную фигуру

// Вызываем функцию для перерисовки окна

InvalidateRect(hwnd, NULL, TRUE);

}

break;

case WM\_PAINT:

// Перерисовываем окно

hdc = BeginPaint(hwnd, &ps);

// Отрисовываем все сохраненные фигуры

for (const Shape& shape : shapes) {

if (shape.isEllipse) {

// Если фигура - круг, то устанавливаем желтую кисть и контур

yellowBrush = CreateSolidBrush(RGB(255, 255, 0)); // Желтая кисть

yellowPen = CreatePen(PS\_SOLID, 1, RGB(255, 255, 0)); // Желтый контур

SelectObject(hdc, yellowBrush);

SelectObject(hdc, yellowPen);

Ellipse(hdc, shape.startX, shape.startY, shape.endX, shape.endY);

DeleteObject(yellowBrush);

DeleteObject(yellowPen);

}

else {

// Если фигура - прямоугольник, то устанавливаем зеленую кисть и контур

greenBrush = CreateSolidBrush(RGB(0, 255, 0)); // Зеленая кисть

greenPen = CreatePen(PS\_SOLID, 1, RGB(0, 255, 0)); // Зеленый контур

SelectObject(hdc, greenBrush);

SelectObject(hdc, greenPen);

Rectangle(hdc, shape.startX, shape.startY, shape.endX, shape.endY);

DeleteObject(greenBrush);

DeleteObject(greenPen);

}

}

EndPaint(hwnd, &ps);

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

return DefWindowProcW(hwnd, msg, wParam, lParam);

}

return 0;

}