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.

Выполнил студент гр.153502 Леоненко А.О.

Проверил ассистент кафедры информатики Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Формулировка задачи 3](#_Toc145669228)

[2 Описание функций программы 4](#_Toc145669229)

[Список использованных источников 6](#_Toc145669230)

[Приложение А 7](#_Toc145669231)

1 ФОРМУЛИРОВКА ЗАДАЧИ

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 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

Согласно формулировке задачи, были спроектированы следующие функции программы:

– Создание фигур (кругов, прямоугольников);

– Редактирование фигур.

2.1 Изменение режима

Для изменения режима (создание фигур, редактирования) необходимо нажать клавишу Shift.

2.2 Режим создания фигур

Режим создания фигур (рисунок 1) доступен изначально при запуске программы. Для создания фигуры необходимо зажать ЛКМ и провести желаемую фигуру, после отжать ЛКМ. Для изменения фигуры, необходимо нажать пробел.

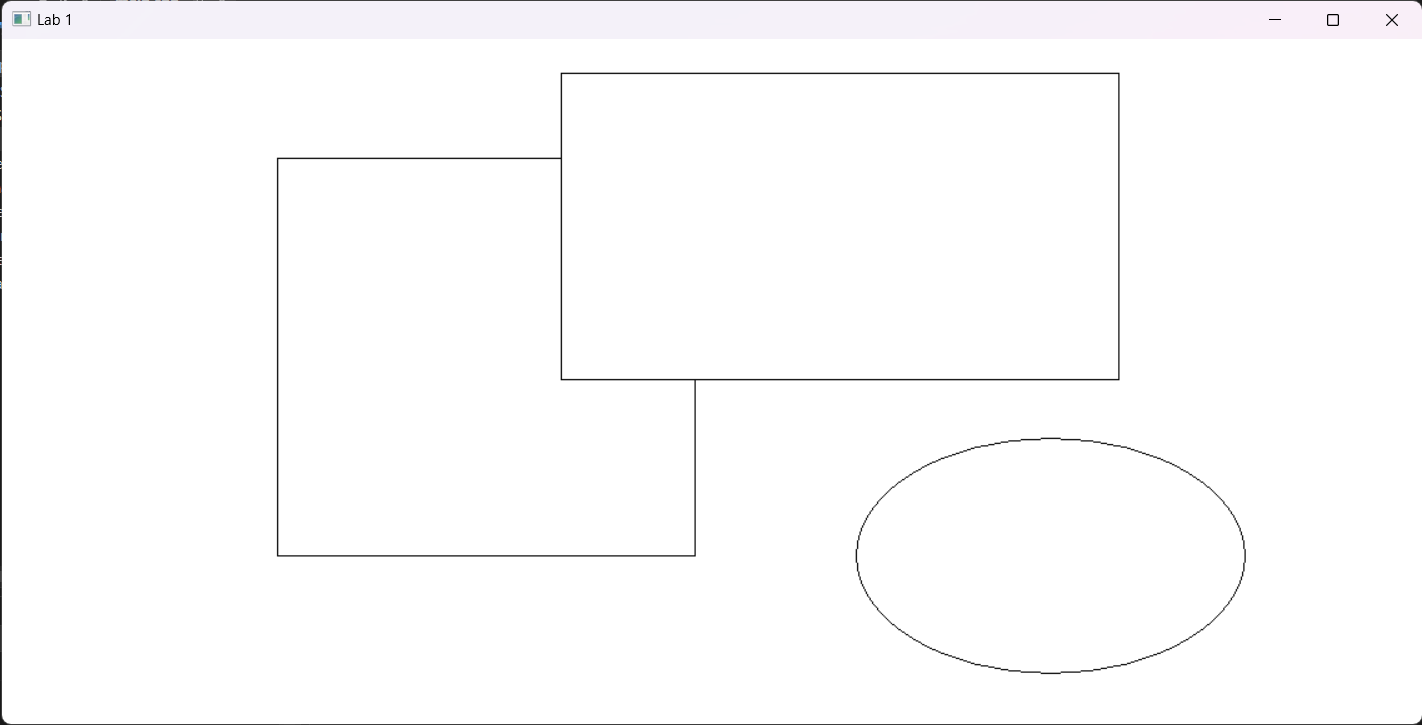


Рисунок 1 – Режим создания фигур

2.3 Режим редактирования фигур

Для изменения фигуры необходимо перейти в режим редактирования. Далее необходимо навести курсор на желаемую геометрическую форму, после чего можно изменить положение фигуры в окне с помощью клавишей влево, вправо, вверх и вниз (рисунок 2). Так же в этот момент появляется возможность изменить саму фигуру. Для этого необходимо навести курсор на правый нижний либо левый верхний угол, нажать ЛКМ, после чего навести курсор в желаемое положение и нажать ЛКМ (рисунок 3).

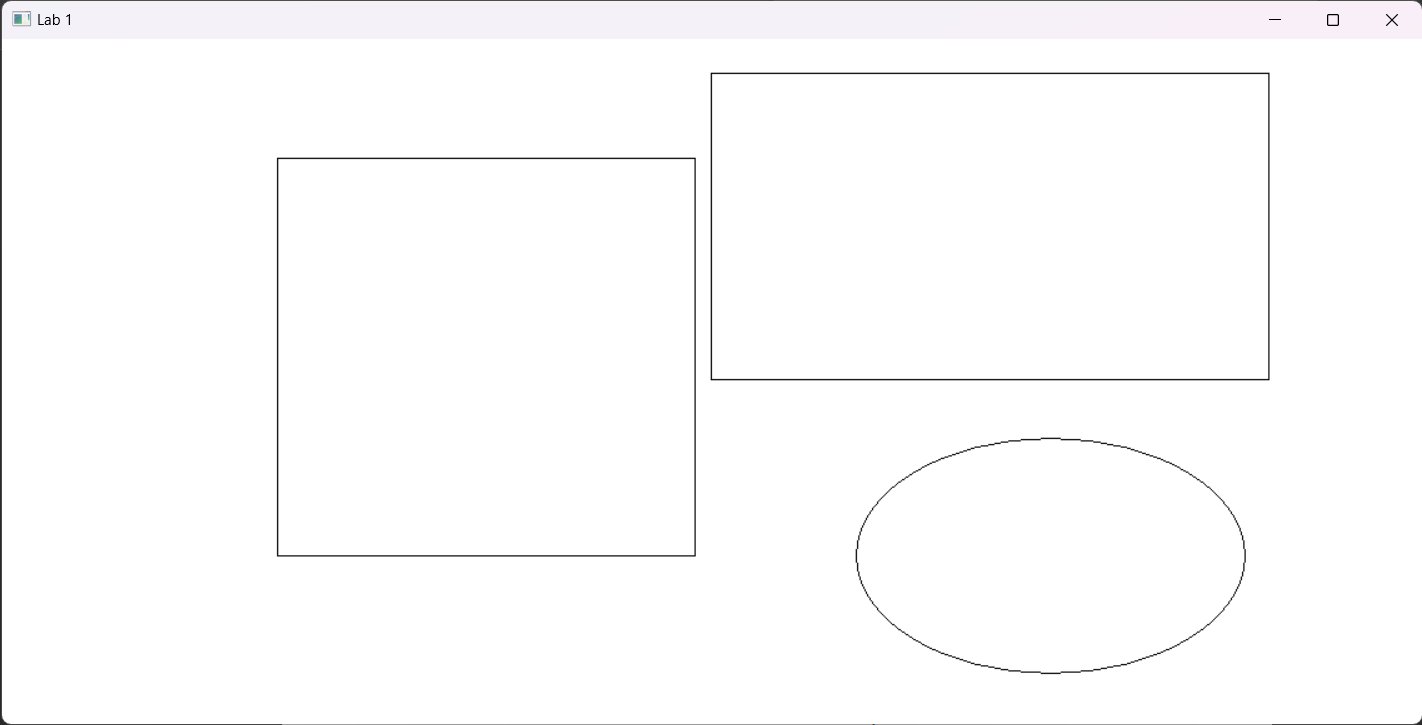


Рисунок 2 – Результат изменения положения прямоугольника в режиме редактирования

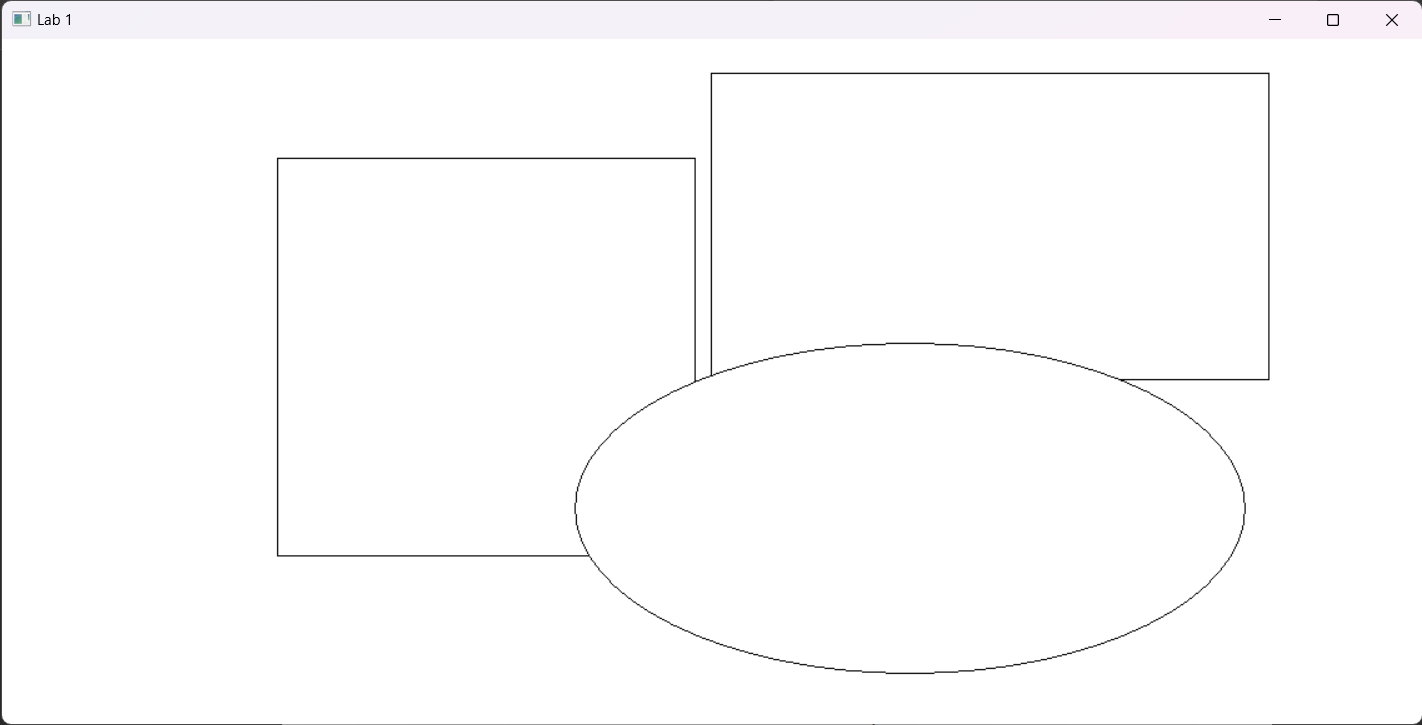


Рисунок 3 – Результат изменения размера фигуры в режиме редактирования

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Build desktop Windows apps using the Win32 API [Электронный ресурс]. – Электронные данные. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/

[2] Основные сообщения ОС Windows (Win32 API). Программирование в ОС Windows. Лекция 1. – Электронные данные. – Режим доступа: https://www.youtube.com/watch?v=wTArIolxch0

[3] КАК рисовать в Win32 API? [Электронный ресурс]. – Электронные данные. – Режим доступа: https://radiofront.narod.ru/htm/prog/htm/winda/api/paint.html#s

ПРИЛОЖЕНИЕ А

Листинг кода

Файл main.cpp

#ifndef UNICODE

#define UNICODE

#endif

#include <windows.h>

#include <tchar.h>

#include <vector>

RECT rc = { 0 };

int WindowPosX = 0;

int WindowPosY = 0;

int lastShapeId = 0;

class Shape {

public:

int shapeId;

int x1, y1, x2, y2;

int shapeType; // 0 - rectangle, 1 - circle

COLORREF color;

Shape() {}

Shape(int \_shapeType)

{

shapeId = ++lastShapeId;

shapeType = \_shapeType;

}

void checkCoord()

{

if (this->x1 > this->x2)

{

int buff = this->x1;

this->x1 = this->x2;

this->x2 = buff;

}

if (this->y1 > this->y2)

{

int buff = this->y1;

this->y1 = this->y2;

this->y2 = buff;

}

}

void moveFigure(int dx, int dy)

{

x1 += dx;

x2 += dx;

y1 += dy;

y2 += dy;

}

};

int choosenShapeType = 0;

std::vector<Shape\*> shapes;

Shape\* currentShape = nullptr;

bool isDrawing = false; // while LMB clicked

bool isEditing = false;

// Editing staff

int selectedShapeIndex = -1;

bool isCornerSelected = false;

bool isLtCornerSelected = false;

int deltaWay = 20;

void DrawShape(HDC hdc, int shapeType, int x1, int y1, int x2, int y2)

{

if (shapeType == 0)

Rectangle(hdc, x1, y1, x2, y2);

else

Ellipse(hdc, x1, y1, x2, y2);

}

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam);

int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, PWSTR pCmdLine, int nCmdShow)

{

const wchar\_t CLASS\_NAME[] = L"Sample Window Class";

WNDCLASS wc = { };

wc.lpfnWndProc = WindowProc;

wc.hInstance = hInstance;

wc.lpszClassName = CLASS\_NAME;

wc.style = CS\_HREDRAW | CS\_VREDRAW; // Window redraw after sizing

RegisterClass(&wc);

HWND hwnd = CreateWindowEx(

0, // Optional window styles.

CLASS\_NAME, // Window class

L"Lab 1", // Window text

WS\_OVERLAPPEDWINDOW, // Window style

CW\_USEDEFAULT, CW\_USEDEFAULT, CW\_USEDEFAULT, CW\_USEDEFAULT,

NULL, // Parent window

NULL, // Menu

hInstance, // Instance handle

NULL // Additional application data

);

if (hwnd == NULL)

{

return 0;

}

ShowWindow(hwnd, nCmdShow);

MSG msg = { };

while (GetMessage(&msg, NULL, 0, 0) > 0)

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return 0;

}

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)

{

switch (uMsg)

{

case WM\_DESTROY:

shapes.clear();

PostQuitMessage(0);

return 0;

case WM\_PAINT:

{

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hwnd, &ps);

FillRect(hdc, &ps.rcPaint, (HBRUSH)(COLOR\_WINDOW + 1));

for (int i = 0; i < shapes.size(); i++)

{

Shape shape = \*shapes[i];

DrawShape(hdc, shape.shapeType, shape.x1, shape.y1, shape.x2, shape.y2);

}

if (currentShape != nullptr)

DrawShape(hdc, currentShape->shapeType, currentShape->x1, currentShape->y1, currentShape->x2, currentShape->y2);

EndPaint(hwnd, &ps);

}

return 0;

case WM\_LBUTTONDOWN:

if (!isEditing)

{

isDrawing = true;

currentShape = new Shape(choosenShapeType);

currentShape->x1 = currentShape->x2 = LOWORD(lParam);

currentShape->y1 = currentShape->y2 = HIWORD(lParam);

}

else if (isEditing && !isCornerSelected)

{

if (selectedShapeIndex == -1)

{

for (int i = shapes.size() - 1; i >= 0; i--)

{

Shape shape = \*shapes[i];

int x = LOWORD(lParam);

int y = HIWORD(lParam);

if (shape.x1 <= x && shape.x2 >= x && shape.y1 <= y && shape.y2 >= y)

{

selectedShapeIndex = i;

break;

}

}

}

else

{

Shape shape = \*shapes[selectedShapeIndex];

int x1 = shape.x1;

int y1 = shape.y1;

int x2 = shape.x2;

int y2 = shape.y2;

int x = LOWORD(lParam);

int y = HIWORD(lParam);

double lt\_corner\_way = sqrt(pow((x1 - x), 2) + pow((y1 - y), 2));

double rd\_corner\_way = sqrt(pow((x2 - x), 2) + pow((y2 - y), 2));

isCornerSelected = true;

if (lt\_corner\_way < rd\_corner\_way)

isLtCornerSelected = true;

else

isLtCornerSelected = false;

}

}

else // corner selected

{

if (isLtCornerSelected)

{

shapes[selectedShapeIndex]->x1 = LOWORD(lParam);

shapes[selectedShapeIndex]->y1 = HIWORD(lParam);

}

else

{

shapes[selectedShapeIndex]->x2 = LOWORD(lParam);

shapes[selectedShapeIndex]->y2 = HIWORD(lParam);

}

shapes[selectedShapeIndex]->checkCoord();

isCornerSelected = false;

selectedShapeIndex = -1;

InvalidateRect(hwnd, NULL, TRUE);

}

break;

case WM\_MOUSEMOVE:

if (isDrawing)

{

currentShape->x2 = LOWORD(lParam);

currentShape->y2 = HIWORD(lParam);

InvalidateRect(hwnd, NULL, TRUE);

}

break;

case WM\_LBUTTONUP:

if (isDrawing)

{

isDrawing = false;

currentShape->checkCoord();

shapes.push\_back(currentShape);

}

case WM\_SIZE:

rc.right = LOWORD(lParam);

rc.bottom = HIWORD(lParam);

break;

case WM\_MOVE:

WindowPosX = (int)(short)LOWORD(lParam); // horizontal position

WindowPosY = (int)(short)HIWORD(lParam); // vertical position

InvalidateRect(hwnd, 0, TRUE); // update window after moving

break;

case WM\_KEYDOWN:

if (wParam == VK\_ESCAPE)

PostMessage(hwnd, WM\_DESTROY, 0, 0);

else if (wParam == VK\_SPACE)

{

if (choosenShapeType == 0)

choosenShapeType = 1;

else

choosenShapeType = 0;

}

else if (wParam == VK\_SHIFT)

{

if (!isEditing)

{

if (isDrawing)

{

isDrawing = false;

shapes.push\_back(currentShape);

}

isEditing = true;

}

else

{

isEditing = false;

selectedShapeIndex = -1;

isCornerSelected = false;

}

}

else if (wParam == VK\_LEFT)

{

if (isEditing && selectedShapeIndex != -1)

shapes[selectedShapeIndex]->moveFigure(-deltaWay, 0);

InvalidateRect(hwnd, 0, TRUE);

}

else if (wParam == VK\_RIGHT)

{

if (isEditing && selectedShapeIndex != -1)

shapes[selectedShapeIndex]->moveFigure(deltaWay, 0);

InvalidateRect(hwnd, 0, TRUE);

}

else if (wParam == VK\_UP)

{

if (isEditing && selectedShapeIndex != -1)

shapes[selectedShapeIndex]->moveFigure(0, -deltaWay);

InvalidateRect(hwnd, 0, TRUE);

}

else if (wParam == VK\_DOWN)

{

if (isEditing && selectedShapeIndex != -1)

shapes[selectedShapeIndex]->moveFigure(0, deltaWay);

InvalidateRect(hwnd, 0, TRUE);

}

break;

}

return DefWindowProc(hwnd, uMsg, wParam, lParam);

}