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 Win32 API. Оконное приложение Win32 с минимальной функциональной достаточностью. Обработка основных оконных сообщений

Выполнил:

студент гр. 153504

Климкович Н.В.

Проверил:

Гриценко Н.Ю.

Минск 2023

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc146631498)

[2 Краткие теоретические сведения 4](#_Toc146631499)

[3 Результаты выполнения лабораторной работы 5](#_Toc146631500)

[Выводы 6](#_Toc146631501)

[Список использованных источников 7](#_Toc146631502)

[Приложение А (обязательное) Листинг кода 8](#_Toc146631503)

## 1 ПОСТАНОВКА ЗАДАЧИ

Целью выполнения лабораторной работы является создание оконного приложения на Win32 API с минимальной достаточной функциональностью, позволяющим отработать базовые навыки написания программы на Win32 API. Реализовать графический редактор для рисования и редактирования графических фигур с помощью мыши и клавиш клавиатуры, которое обрабатывает основные оконные сообщения, такие как WM\_CREATE, WM\_PAINT, WM\_LBUTTONUP, WM\_ HSCROLL, WM\_MOUSEMOVE, WM\_KEYDOWN, WM\_LBUTTONDOWN, и WM\_COMMAND.

## 2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Win32 API (Windows API) представляет собой набор функций и интерфейсов, предоставляемых операционной системой Windows для разработки приложений. Этот мощный набор инструментов обеспечивает доступ к различным функциональным возможностям Windows, включая создание и управление окнами, обработку сообщений, работу с файлами и реестром, а также многие другие операции. Win32 API играет ключевую роль в разработке приложений для Windows и обеспечивает высокую степень контроля над поведением приложений.

Для выполнения данной лабораторной работы, были использованы следующие теоретические сведения и концепции:

– Win32 API (Application Programming Interface): это набор функций и структур, предоставляемых операционной системой Windows для разработки приложений под Windows. Для создания приложения было использовано множество функций Win32 API для создания окна, обработки сообщений и других задач.

– Обработка событий: код обрабатывает различные события, такие как нажатия кнопок. Это обеспечивает реакцию приложения на действия пользователя.

## 3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

В ходе выполнения лабораторной работы было разработано приложение, которое позволяет пользователю рисовать и редактировать графические фигуры с помощью мыши и клавиш клавиатуры. Приложение предоставляет пользователю возможность изменять толщину контура, масштаб, цвет заливки и линии геометрических фигур. Также пользователь может изменять положение объектов, изменять порядок отображения, удалять объекты и выбрать количество углов в многоугольнике (Рисунок 1).

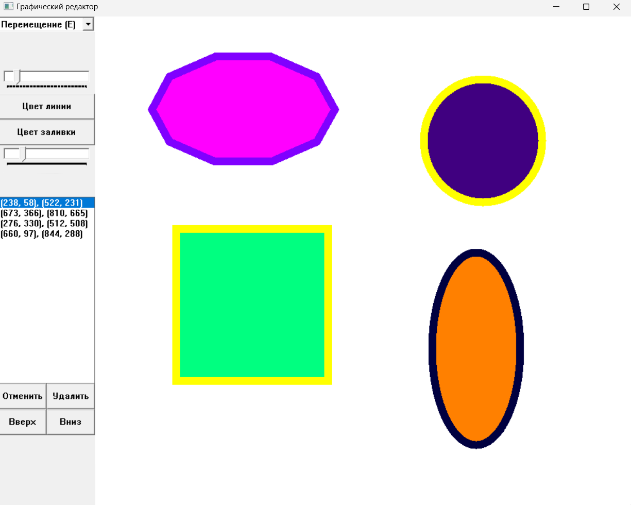


Рисунок 1 - Главное окно

## ВЫВОДЫ

В ходе выполнения лабораторной работы было успешно создано оконное приложение на платформе Win32 API с минимально необходимой функциональностью. Приложение предоставляет пользователю удобный инструмент для рисования и редактирования графических фигур с использованием мыши и клавиатуры. Оно обрабатывает основные оконные сообщения, что позволяет гарантировать стабильную работу программы. Данное приложение является полезным инструментом для работы с изображениями. В итоге, выполнение лабораторной работы позволило укрепить базовые навыки разработки на платформе Win32 API и создать функциональное оконное приложение.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Создание классических приложений для Windows с использованием API Win32 [Электронный ресурс]. ─ Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/api ─ Дата доступа 12.09.2023

[2] Щупак Ю. Win32 API. Разработка приложений для Windows. ─ СПб: Питер, 2008. ─ 592 с.: ип.

## ПРИЛОЖЕНИЕ А

## (обязательное)

## Листинг кода

**Lab1.cpp**

#include <windows.h>

#include <windows.h>

#include <vector>

#include <CommCtrl.h>

#include <commdlg.h>

#include <string>

#define M\_PI 3.141592653589793238462643383279

struct Shape

{

RECT rect;

bool isCorrect;

int n;

int selectedShape;

int Thickness;

COLORREF selectedColorThickness;

COLORREF selectedColorBrush;

std::vector<POINT> pen;

};

struct PaintWindow

{

int x1 = 150;

int y1 = 0;

int x2 = 1000;

int y2 = 800;

int Width = x2 - x1;

int Height = y2 - y1;

};

HINSTANCE hInst;

HDC hdcBuffer;

HBITMAP hBitmap;

HWND hwndMain, hwndComboBox, hSlider, hSliderThickness, hwndList, hwndDeleteItem, hwndUpItem, hwndDownItem;

COLORREF customColorsThickness[16]{ 0 };

COLORREF customColorsBrush[16]{ 0 };

CHOOSECOLOR ccThickness, ccBrush;

COLORREF selectedColorThickness, selectedColorBrush;

PaintWindow PW;

int selectedItemIndex = -1;

std::vector<Shape> shapes;

RECT currentShape;

bool isDrawing = false;

bool isMove = false;

int selectedShape = 1;

int n = 3;

int Thickness = 1;

std::vector<POINT> pen;

POINT startPos;

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

void DrawShape(HDC\* hdc, bool isCorrect);

void FillRectWindow();

void RePaint(bool ctrlZ, bool del);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow)

{

WNDCLASSEX wc = { sizeof(WNDCLASSEX), CS\_HREDRAW | CS\_VREDRAW, WndProc, 0, 0, GetModuleHandle(NULL), NULL, NULL, NULL, NULL, L"MyWindowClass", NULL };

RegisterClassEx(&wc);

hwndMain = CreateWindow(L"MyWindowClass", L"WS\_OVERLAPPEDWINDOW | WS\_CLIPCHILDREN, NULL, NULL, PW.x2, PW.y2, NULL, NULL, hInstance, NULL);

SetWindowLong(hwndMain, GWL\_STYLE, GetWindowLong(hwndMain, GWL\_STYLE) & ~WS\_THICKFRAME);

HDC hdc = GetDC(hwndMain);

hdcBuffer = CreateCompatibleDC(hdc);

hBitmap = CreateCompatibleBitmap(hdc, PW.Width, PW.Height);

SelectObject(hdcBuffer, hBitmap);

ShowWindow(hwndMain, nCmdShow);

RECT rectSlider{ 0,0,PW.x1,PW.y2 };

FillRect(hdc, &rectSlider, (HBRUSH)(COLOR\_WINDOW));

ReleaseDC(hwndMain, hdc);

UpdateWindow(hwndMain);

MSG msg;

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

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

DeleteObject(hBitmap);

DeleteDC(hdcBuffer);

return msg.wParam;

}

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

{

switch (msg)

{

case WM\_CREATE:

{

#pragma region Elements

hwndComboBox = CreateWindow(L"COMBOBOX", NULL, WS\_CHILD | WS\_VISIBLE | CBS\_DROPDOWNLIST | WS\_VSCROLL, 0, 00, PW.x1, 200, hwnd, NULL, NULL, NULL);

SendMessage(hwndComboBox, CB\_ADDSTRING, 0, (LPARAM)L" (E)");

SendMessage(hwndComboBox, CB\_ADDSTRING, 0, (LPARAM)L" (A)");

SendMessage(hwndComboBox, CB\_ADDSTRING, 0, (LPARAM)L"N- (B)");

SendMessage(hwndComboBox, CB\_ADDSTRING, 0, (LPARAM)L" (C)");

SendMessage(hwndComboBox, CB\_ADDSTRING, 0, (LPARAM)L" (D)");

SendMessage(hwndComboBox, CB\_SETCURSEL, 1, 0);

#pragma endregion

#pragma region SliderN

hSlider = CreateWindowEx(0, TRACKBAR\_CLASS, NULL, TBS\_AUTOTICKS | TBS\_ENABLESELRANGE | WS\_CHILD | WS\_VISIBLE, 0, 40, PW.x1, 40, hwnd, NULL, hInst, NULL);

SendMessage(hSlider, TBM\_SETRANGE, TRUE, MAKELPARAM(3, 20));

SendMessage(hSlider, WM\_SETREDRAW, FALSE, 0);

ShowWindow(hSlider, SW\_HIDE);

#pragma endregion

#pragma region Slider Thickness

hSliderThickness = CreateWindowEx(0, TRACKBAR\_CLASS, NULL, TBS\_AUTOTICKS | TBS\_ENABLESELRANGE | WS\_CHILD | WS\_VISIBLE, 0, 80, PW.x1, 40, hwnd, NULL, hInst, NULL);

SendMessage(hSliderThickness, TBM\_SETRANGE, TRUE, MAKELPARAM(1, 100));

#pragma endregion

#pragma region Color Choose Thickness

HWND hButton1 = CreateWindow(L"BUTTON", L"", WS\_CHILD | WS\_VISIBLE, 0, 120, PW.x1, 40, hwnd, (HMENU)1001, GetModuleHandle(NULL), NULL);

ZeroMemory(&ccThickness, sizeof(ccThickness));

ccThickness.lStructSize = sizeof(ccThickness);

ccThickness.hwndOwner = hwndMain;

ccThickness.lpCustColors = (LPDWORD)customColorsThickness;

ccThickness.rgbResult = RGB(255, 0, 0);

ccThickness.Flags = CC\_FULLOPEN | CC\_RGBINIT;

#pragma endregion

#pragma region Color Choose Brush

HWND hButton2 = CreateWindow(L"BUTTON", L"", WS\_CHILD | WS\_VISIBLE, 0, 160, PW.x1, 40, hwnd, (HMENU)1002, GetModuleHandle(NULL), NULL);

ZeroMemory(&ccBrush, sizeof(ccBrush));

ccBrush.lStructSize = sizeof(ccBrush);

ccBrush.hwndOwner = hwndMain;

ccBrush.lpCustColors = (LPDWORD)customColorsThickness;

ccBrush.rgbResult = RGB(255, 0, 0);

ccBrush.Flags = CC\_FULLOPEN | CC\_RGBINIT;

#pragma endregion

#pragma region Color Choose Brush

hwndList = CreateWindowEx(0, L"LISTBOX", NULL, WS\_CHILD | WS\_VISIBLE | WS\_BORDER | LBS\_NOTIFY | WS\_VSCROLL, 0, 240, PW.x1, 300, hwnd, (HMENU)100, GetModuleHandle(NULL), NULL);

hwndDeleteItem = CreateWindow(L"BUTTON", L", WS\_CHILD | WS\_VISIBLE, 0, 530, PW.x1, 40, hwnd, (HMENU)101, GetModuleHandle(NULL), NULL);

hwndUpItem = CreateWindow(L"BUTTON", L"", WS\_CHILD | WS\_VISIBLE, 0, 570, PW.x1 / 2, 40, hwnd, (HMENU)102, GetModuleHandle(NULL), NULL);

hwndDownItem = CreateWindow(L"BUTTON", L"", WS\_CHILD | WS\_VISIBLE, PW.x1 / 2, 570, PW.x1 / 2, 40, hwnd, (HMENU)103, GetModuleHandle(NULL), NULL);

#pragma endregion

break;

}

case WM\_PAINT:

{

if (isDrawing)

{

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hwnd, &ps);

HPEN hPen = CreatePen(PS\_SOLID, Thickness, selectedColorThickness);

HBRUSH hBrush = CreateSolidBrush(selectedColorBrush);

SelectObject(hdc, hPen);

SelectObject(hdc, hBrush);

BitBlt(hdc, PW.x1, PW.y1, PW.Width, PW.Height, hdcBuffer, 0, 0, SRCCOPY);

DrawShape(&hdc, GetKeyState(VK\_SHIFT) & 0x8000);

DeleteObject(hPen);

DeleteObject(hBrush);

EndPaint(hwnd, &ps);

}

break;

}

case WM\_LBUTTONDOWN:

{

if (LOWORD(lParam) >= PW.x1 && selectedShape != 0) {

isDrawing = true;

currentShape.left = LOWORD(lParam);

currentShape.top = HIWORD(lParam);

currentShape.right = LOWORD(lParam);

currentShape.bottom = HIWORD(lParam);

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hwnd, &ps);

BitBlt(hdcBuffer, 0, 0, PW.Width, PW.Height, hdc, PW.x1, PW.y1, SRCCOPY);

EndPaint(hwnd, &ps);

if (selectedShape == 4) {

pen.push\_back({ LOWORD(lParam) ,HIWORD(lParam) });

}

}

if (selectedShape == 0)

{

isMove = true;

startPos = { LOWORD(lParam), HIWORD(lParam) };

}

break;

}

case WM\_MOUSEMOVE:

{

if (isDrawing)

{

if (LOWORD(lParam) >= PW.x1) {

currentShape.right = LOWORD(lParam);

currentShape.bottom = HIWORD(lParam);

if (selectedShape == 4) {

pen.push\_back({ LOWORD(lParam) ,HIWORD(lParam) });

}

InvalidateRect(hwnd, NULL, TRUE);

}

}

if (isMove)

{

int selectedIndex = SendMessage(hwndList, LB\_GETCURSEL, 0, 0);

if (selectedIndex != LB\_ERR) {

selectedIndex = shapes.size() - selectedIndex - 1;

int x = LOWORD(lParam) - startPos.x;

int y = HIWORD(lParam) - startPos.y;

if (shapes[selectedIndex].selectedShape == 4)

{

std::vector<POINT> bufpoints = shapes[selectedIndex].pen;

for (long a = 0;a < shapes[selectedIndex].pen.size();a++)

{

shapes[selectedIndex].pen[a].x += x;

shapes[selectedIndex].pen[a].y += y;

}

RePaint(false, false);

shapes[selectedIndex].pen = bufpoints;

bufpoints.clear();

}

else

{

RECT bufrect = shapes[selectedIndex].rect;

shapes[selectedIndex].rect.left += x;

shapes[selectedIndex].rect.right += x;

shapes[selectedIndex].rect.top += y;

shapes[selectedIndex].rect.bottom += y;

RePaint(false, false);

shapes[selectedIndex].rect = bufrect;

}

}

}

break;

}

case WM\_LBUTTONUP:

{

if (isDrawing) {

isDrawing = false;

if (selectedShape == 4) {

pen.push\_back({ LOWORD(lParam) ,HIWORD(lParam) });

}

shapes.push\_back({ currentShape, bool(GetKeyState(VK\_SHIFT) & 0x8000),n,selectedShape, Thickness,selectedColorThickness,selectedColorBrush,pen });

pen.clear();

wchar\_t buffer[30];

swprintf(buffer, 30, L"(%d, %d), (%d, %d)", currentShape.left, currentShape.top, currentShape.right, currentShape.bottom);

SendMessage(hwndList, LB\_INSERTSTRING, (WPARAM)0, (LPARAM)buffer);

currentShape = { 0, 0, 0, 0 };

}

if (isMove) {

isMove = false;

int selectedIndex = SendMessage(hwndList, LB\_GETCURSEL, 0, 0);

if (selectedIndex != LB\_ERR) {

selectedIndex = shapes.size() - selectedIndex - 1;

int x = LOWORD(lParam) - startPos.x;

int y = HIWORD(lParam) - startPos.y;

if (shapes[selectedIndex].selectedShape == 4)

{

for (long a = 0;a < shapes[selectedIndex].pen.size();a++)

{

shapes[selectedIndex].pen[a].x += x;

shapes[selectedIndex].pen[a].y += y;

}

RePaint(false, false);

}

else

{

shapes[selectedIndex].rect.left += x;

shapes[selectedIndex].rect.right += x;

shapes[selectedIndex].rect.top += y;

shapes[selectedIndex].rect.bottom += y;

RePaint(false, false);

}

}

SetFocus(hwndMain);

}

break;

}

case WM\_KEYDOWN:

{

int lower = tolower((unsigned char)wParam);

if (wParam == 'Z' && GetKeyState(VK\_CONTROL) < 0) {

RePaint(true, true);

SetFocus(hwnd);

}

else

{

switch (lower)

{

case L'a':

case L'c':

case L'd':

selectedShape = lower - L'a' + 1;

SendMessage(hwndComboBox, CB\_SETCURSEL, selectedShape, 0);

ShowWindow(hSlider, SW\_HIDE);

FillRectWindow();

break;

case L'b':

selectedShape = lower - L'a' + 1;

SendMessage(hwndComboBox, CB\_SETCURSEL, selectedShape, 0);

ShowWindow(hSlider, SW\_SHOW);

break;

case L'e':

selectedShape = 0;

SendMessage(hwndComboBox, CB\_SETCURSEL, selectedShape, 0);

ShowWindow(hSlider, SW\_HIDE);

FillRectWindow();

break;

}

}

break;

}

case WM\_COMMAND:

{

if (LOWORD(wParam) == 0 && HIWORD(wParam) == CBN\_SELCHANGE) {

int selectedIndex = SendMessage(hwndComboBox, CB\_GETCURSEL, 0, 0);

if (selectedIndex != CB\_ERR) {

if (selectedIndex == 2)

{

ShowWindow(hSlider, SW\_SHOW);

}

else

{

ShowWindow(hSlider, SW\_HIDE);

FillRectWindow();

}

selectedShape = selectedIndex;

}

SetFocus(hwnd);

}

if (LOWORD(wParam) == 1001)

{

if (ChooseColor(&ccThickness))

{

selectedColorThickness = ccThickness.rgbResult;

}

}

if (LOWORD(wParam) == 1002)

{

if (ChooseColor(&ccBrush))

{

selectedColorBrush = ccBrush.rgbResult;

}

}

if (LOWORD(wParam) == 100) {

if (HIWORD(wParam) == LBN\_SELCHANGE)

{

int selectedIndex = SendMessage(hwndList, LB\_GETCURSEL, 0, 0);

SetFocus(hwndMain);

SendMessage(hwndList, LB\_SETSEL, selectedIndex, 0);

}

}

if (LOWORD(wParam) == 101)

{

int selectedIndex = SendMessage(hwndList, LB\_GETCURSEL, 0, 0);

if (selectedIndex != LB\_ERR) {

RePaint(false, true);

}

else

{

MessageBox(NULL, L"!", L"", MB\_ICONERROR | MB\_OK);

}

SetFocus(hwndMain);

}

if (LOWORD(wParam) == 102)

{

int selectedIndex = SendMessage(hwndList, LB\_GETCURSEL, 0, 0);

if (selectedIndex != LB\_ERR) {

if (selectedIndex > 0)

{

Shape buf = shapes[shapes.size() - selectedIndex];

shapes[shapes.size() - selectedIndex] = shapes[shapes.size() - selectedIndex - 1];

shapes[shapes.size() - selectedIndex - 1] = buf;

wchar\_t buffer1[30];

wchar\_t buffer2[30];

SendMessage(hwndList, LB\_GETTEXT, selectedIndex - 1, (LPARAM)buffer1);

SendMessage(hwndList, LB\_GETTEXT, selectedIndex, (LPARAM)buffer2);

SendMessage(hwndList, LB\_DELETESTRING, selectedIndex - 1, 0);

SendMessage(hwndList, LB\_DELETESTRING, selectedIndex - 1, 0);

SendMessage(hwndList, LB\_INSERTSTRING, selectedIndex - 1, (LPARAM)buffer1);

SendMessage(hwndList, LB\_INSERTSTRING, selectedIndex - 1, (LPARAM)buffer2);

SendMessage(hwndList, LB\_SETCURSEL, selectedIndex - 1, 0);

RePaint(false, false);

}

}

else

{

MessageBox(NULL, L"!", L"", MB\_ICONERROR | MB\_OK);

}

SetFocus(hwndMain);

}

if (LOWORD(wParam) == 103)

{

int selectedIndex = SendMessage(hwndList, LB\_GETCURSEL, 0, 0);

if (selectedIndex != LB\_ERR) {

if (selectedIndex < shapes.size() - 1)

{

Shape buf = shapes[shapes.size() - selectedIndex - 1];

shapes[shapes.size() - selectedIndex - 1] = shapes[shapes.size() - selectedIndex - 2];

shapes[shapes.size() - selectedIndex - 2] = buf;

wchar\_t buffer1[30];

wchar\_t buffer2[30];

SendMessage(hwndList, LB\_GETTEXT, selectedIndex + 1, (LPARAM)buffer1);

SendMessage(hwndList, LB\_GETTEXT, selectedIndex, (LPARAM)buffer2);

SendMessage(hwndList, LB\_DELETESTRING, selectedIndex, 0);

SendMessage(hwndList, LB\_DELETESTRING, selectedIndex, 0);

SendMessage(hwndList, LB\_INSERTSTRING, selectedIndex, (LPARAM)buffer2);

SendMessage(hwndList, LB\_INSERTSTRING, selectedIndex, (LPARAM)buffer1);

SendMessage(hwndList, LB\_SETCURSEL, selectedIndex + 1, 0);

RePaint(false, false);

}

}

else

{

MessageBox(NULL, L"!", L"", MB\_ICONERROR | MB\_OK);

}

SetFocus(hwndMain);

}

break;

}

case WM\_HSCROLL:

{

if (lParam == (LPARAM)hSlider)

n = SendMessage(hSlider, TBM\_GETPOS, 0, 0);

if (lParam == (LPARAM)hSliderThickness)

Thickness = SendMessage(hSliderThickness, TBM\_GETPOS, 0, 0);

if (lParam == (LPARAM)hSliderThickness)

Thickness = SendMessage(hSliderThickness, TBM\_GETPOS, 0, 0);

SetFocus(hwnd);

break;

}

case WM\_DESTROY:

{

PostQuitMessage(0);

break;

}

}

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

}

void FillRectWindow() {

HDC hdchwndMain = GetDC(hwndMain);

RECT rect1{ 0,0,PW.x1,PW.y2 };

FillRect(hdchwndMain, &rect1, (HBRUSH)(COLOR\_WINDOW));

ReleaseDC(hwndMain, hdchwndMain);

}

void RePaint(bool ctrlZ, bool del)

{

if (shapes.size() > 0) {

int indexItem;

if (del) {

if (ctrlZ)

{

shapes.pop\_back();

indexItem = 0;

}

else

{

indexItem = SendMessage(hwndList, LB\_GETCURSEL, 0, 0);

shapes.erase(shapes.begin() + shapes.size() - indexItem - 1);

}

SendMessage(hwndList, LB\_DELETESTRING, indexItem, 0);

}

COLORREF bufSelectedColorThickness = selectedColorThickness;

COLORREF bufSelectedColorBrush = selectedColorBrush;

RECT bufCurrentShape = currentShape;

int bufSelectedShape = selectedShape;

int bufn = n;

std::vector<POINT> bufpen = pen;

HDC hdc = GetDC(hwndMain);

RECT rect1{ PW.x1, PW.y1, PW.x2, PW.y2, };

FillRect(hdc, &rect1, (HBRUSH)(COLOR\_WINDOW + 1));

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

HPEN hPen = CreatePen(PS\_SOLID, shapes[a].Thickness, shapes[a].selectedColorThickness);

HBRUSH hBrush = CreateSolidBrush(shapes[a].selectedColorBrush);

SelectObject(hdc, hPen);

SelectObject(hdc, hBrush);

n = shapes[a].n;

selectedShape = shapes[a].selectedShape;

pen = shapes[a].pen;

currentShape = shapes[a].rect;

DrawShape(&hdc, shapes[a].isCorrect);

DeleteObject(hPen);

DeleteObject(hBrush);

}

ReleaseDC(hwndMain, hdc);

selectedColorThickness = bufSelectedColorThickness;

selectedColorBrush = bufSelectedColorBrush;

currentShape = bufCurrentShape;

selectedShape = bufSelectedShape;

n = bufn;

pen = bufpen;

bufpen.clear();

}

}

void DrawShape(HDC\* hdc, bool isCorrect)

{

long\* x1 = &currentShape.left;

long\* y1 = &currentShape.top;

long\* x2 = &currentShape.right;

long\* y2 = &currentShape.bottom;

switch (selectedShape) {

case 1:

if (isCorrect) {

int centerX = (\*x1 + \*x2) / 2;

int centerY = (\*y1 + \*y2) / 2;

int radius = (\*x2 - \*x1) / 2;

Ellipse(\*hdc, centerX - radius, centerY - radius, centerX + radius, centerY + radius);

}

else {

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

}

break;

case 2:

if (isCorrect)

{

double angle = 2 \* M\_PI / n;

int radius, x;

int y = \*y1;

std::vector<POINT> vertices;

if (\*x2 >= \*x1)

{

if (\*y2 >= \*y1)

{

radius = (\*x2 - \*x1) / 2;

}

else

{

radius = (\*x1 - \*x2) / 2;

}

x = \*x1 + (\*x2 - \*x1) / 2 - radius \* tan(M\_PI / n);

}

else

{

if (\*y2 >= \*y1)

{

radius = (\*x1 - \*x2) / 2;

}

else

{

radius = (\*x2 - \*x1) / 2;

}

x = \*x1 - abs(\*x2 - \*x1) / 2 - radius \* tan(M\_PI / n);

}

for (int i = 0; i < n; i++) {

vertices.push\_back({ x,y });

x += static\_cast<int>(radius \* 2 \* cos(angle \* i));

y += static\_cast<int>(radius \* 2 \* sin(angle \* i));

}

Polygon(\*hdc, vertices.data(), n);

}

else

{

int width = \*x2 - \*x1;

int height = \*y2 - \*y1;

int centerX = (\*x1 + \*x2) / 2;

int centerY = (\*y1 + \*y2) / 2;

double angle = 2 \* M\_PI / n;

std::vector<POINT> vertices;

for (int i = 0; i < n; i++)

{

int x = static\_cast<int>(centerX + width / 2 \* cos(i \* angle));

int y = static\_cast<int>(centerY + height / 2 \* sin(i \* angle));

vertices.push\_back({ x, y });

}

if (!vertices.empty())

{

Polygon(\*hdc, vertices.data(), static\_cast<int>(vertices.size()));

}

}

break;

case 3:

MoveToEx(\*hdc, \*x1, \*y1, NULL);

LineTo(\*hdc, \*x2, \*y2);

break;

case 4:

Polyline(\*hdc, (POINT\*)&pen[0], pen.size());

break;

}

FillRectWindow();

}