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

ОТЧЕТ

к лабораторной работе №3

на тему

Управление памятью и вводом выводом, расширенные возможности ввода-вывода *Windows*. Функции *API* подсистемы памяти *Win32*. Организация и контроль асинхронных операций ввода-вывода. Отображение файлов в память

Студент Н. В. Климкович

Преподаватель Н. Ю. Гриценко

Минск 2023

СОДЕРЖАНИЕ

[1 Цель работы 3](#_Toc146752068)

[2 Теоретические сведения 4](#_Toc146752069)

[3 Результат выполнения 5](#_Toc146752070)

[Заключение 6](#_Toc146752071)

[Список использованных источников 7](#_Toc146752072)

[Приложение А (обязательное) Листинг кода 8](#_Toc146752073)

1. **ЦЕЛЬ РАБОТЫ**

Целью данной работы является создание графического редактора, способного записывать и читать файлы в двоичном формате с использованием маппинга файлов в память. Графический редактор предоставляет пользователю возможность создавать и редактировать графические объекты, а также сохранять текущее состояние холста в файл и восстанавливать его из него. Маппинг файлов в память используется для эффективной работы с файлами, позволяя обращаться к ним как к данным в памяти, что способствует оптимизации операций ввода-вывода.

1. **ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Для работы с файлами и операциями ввода-вывода мы использовали *Win32 API*, часть операционной системы *Windows*. Этот набор функций предоставляет доступ к различным функциональным возможностям *Windows*, включая создание, открытие и запись файлов, а также работу с памятью.

Двоичный формат файлов был выбран для оптимизированного хранения графических данных. Этот формат позволяет компактно записывать информацию о графических объектах в виде последовательности байтов.

Механизм *Memory-Mapped Files* был использован для эффективного взаимодействия с файлами на диске. Он позволяет отображать содержимое файла непосредственно в виртуальное адресное пространство приложения, что снижает нагрузку на операции ввода-вывода и улучшает производительность.

Механизм маппинга файлов в память используется для удобного и эффективного доступа к содержимому файла. Ваш код использует функцию *CreateFileMapping* для создания *memory-mapped* файла, который отображается в виртуальное адресное пространство вашего приложения. Это позволяет вашей программе работать с файлом, как если бы он был в оперативной памяти.

Расширенные возможности ввода-вывода в *Windows* включают функции *ReadFile* и *WriteFile*, которые позволяют считывать и записывать данные в файлы и другие устройства.

Для отладки и мониторинга операций с памятью и вводом-выводом в *Win32* существует набор инструментов, включая отладчики и профилировщики. Эти инструменты позволяют отслеживать процессы, мониторить использование памяти и анализировать производительность приложений. Управление памятью и вводом, выводом важные аспекты при разработке приложений под *Windows*, и правильное использование функций и механизмов *Win32* позволяет создавать эффективные и надежные программы.

1. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

В ходе выполнения лабораторной работы было разработано приложение, которое позволяет пользователю рисовать и редактировать графические фигуры (Рисунок 1).

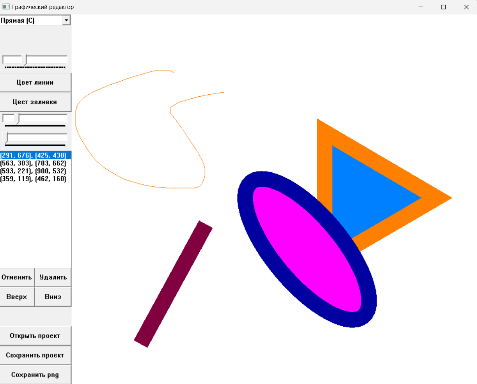


Рисунок 1 – Окно приложения

При выборе опции открытия проекта, программа автоматически загружает графические данные из файла. Пользователь видит, как графические объекты восстанавливаются на холсте, что позволяет ему моментально продолжить работу над проектом.

При выборе опции сохранения проекта, программа автоматически сохраняет текущее состояние проекта в файл. Пользователь видит сообщение об успешном сохранении (Рисунок 2), что подтверждает, что его проект сохранен и готов к дальнейшей работе.

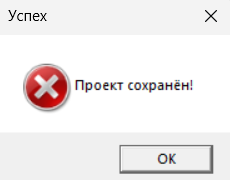


Рисунок 2 – Окно успешного сохранении

# ЗАКЛЮЧЕНИЕ

В результате лабораторной работы было создано графическое приложение, способное записывать и читать файлы в двоичном формате с использованием маппинга файлов в память. Это приложение предоставляет пользователю удобные средства для создания и редактирования графических объектов, а также для сохранения и загрузки проектов.

С использованием функционала *Win32 API* для работы с файлами и операциями ввода-вывода, мы выбрали двоичный формат файлов, что позволило компактно хранить графические данные в виде последовательности байтов. Механизм *Memory-Mapped Files* использовался для эффективной работы с файлами на диске, обеспечивая быстрый доступ и улучшение производительности.

Получившийся результат представляет собой полнофункциональное приложение, готовое к использованию пользователем. Оно обеспечивает удобство и надежность при работе с графическими данными, а также гарантирует сохранность проектов при использовании маппинга файлов в память. Созданное приложение является надежным инструментом для работы с графическими проектами.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Щупак Ю. *Win32 API*. Разработка приложений для *Windows*. ─ СПБ: Питер, 2008. ─ 592 с.: ип.
2. Создание классических приложений для *Windows* с использованием *API Win32* [Электронный ресурс]. ─ Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/api

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

Листинг 1 – Файл Lab1.cpp

#include <windows.h>

#include <vector>

#include <CommCtrl.h>

#include <commdlg.h>

#include <string>

#include <GdiPlus.h>

#include <iostream>

#pragma comment(lib, "gdiplus.lib")

#pragma comment(lib, "user32.lib")

#define M\_PI 3.141592653589793238462643383279

struct Shape

{

RECT rect;

bool isCorrect;

int n;

int selectedShape;

// 1 - круг

// 2 - n-угольник

// 3 - Прямая

// 4 - Карандашь

int Thickness;

COLORREF selectedColorThickness;

COLORREF selectedColorBrush;

std::vector<Gdiplus::Point> pen;

int scale = 100;

int rotation = 0;

};

struct PaintWindow

{

int x1 = 150;

int y1 = 0;

int x2 = 1000;

int y2 = 800;

int Width = x2 - x1;

int Height = y2 - y1;

};

// Глобальные переменные

HINSTANCE hInst;

HDC hdcBuffer;

HBITMAP hBitmap;

HWND hwndMain, hwndComboBox, hSlider, hSliderThickness, hSliderScale, hSliderRotation, hwndList, hwndDeleteItem, hwndUpItem, hwndDownItem, hwndSave;

COLORREF customColorsThickness[16]{ 0 };

COLORREF customColorsBrush[16]{ 0 };

CHOOSECOLOR ccThickness, ccBrush;

COLORREF selectedColorThickness, selectedColorBrush;

HHOOK MouseHook;

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<Gdiplus::Point> pen;

Gdiplus::Point startPos;

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

LRESULT CALLBACK MouseHookProc(int nCode, WPARAM wParam, LPARAM lParam);

void DrawShape(Gdiplus::Graphics& graphics, Gdiplus::Pen\* penPlus, Gdiplus::SolidBrush\* brush, bool isCorrect, int scale = 100, int rotation = 0);

void FillRectWindow();

void RePaint(bool ctrlZ, bool del, bool list = false);

int GetEncoderClsid(const WCHAR\* format, CLSID\* pClsid);

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

{

Gdiplus::GdiplusStartupInput gdiplusStartupInput;

ULONG\_PTR gdiplusToken;

Gdiplus::GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);

MouseHook = SetWindowsHookEx(WH\_MOUSE\_LL, MouseHookProc, NULL, 0);

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"Графический редактор", 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);

}

UnhookWindowsHookEx(MouseHook);

Gdiplus::GdiplusShutdown(gdiplusToken);

DeleteObject(hBitmap);

DeleteDC(hdcBuffer);

return msg.wParam;

}

LRESULT CALLBACK MouseHookProc(int nCode, WPARAM wParam, LPARAM lParam)

{

if (nCode == HC\_ACTION)

{

// wParam содержит информацию о событии мыши (WM\_LBUTTONDOWN, WM\_LBUTTONUP, WM\_MOUSEMOVE и т. д.)

if (wParam == WM\_MOUSEMOVE)

{

if (isDrawing || isMove) {

POINT mousePos;

GetCursorPos(&mousePos);

RECT windowRect;

GetWindowRect(hwndMain, &windowRect);

if (mousePos.x < windowRect.left + PW.x1 || mousePos.x > windowRect.right ||

mousePos.y < windowRect.top || mousePos.y > windowRect.bottom)

{

if (isDrawing) {

isDrawing = false;

InvalidateRect(hwndMain, NULL, TRUE);

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);

}

}

}

}

}

return CallNextHookEx(MouseHook, nCode, wParam, lParam);

}

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"Перемещение (E)");

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

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

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

SendMessage(hwndComboBox, CB\_ADDSTRING, 0, (LPARAM)L"Карандашь (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(0, 100));

#pragma endregion

#pragma region Color Choose Thickness

HWND hButton1 = CreateWindow(L"BUTTON", L"Цвет линии", 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; // Окно-владелец диалога

ccThickness.lpCustColors = (LPDWORD)customColorsThickness;

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

ccThickness.Flags = CC\_FULLOPEN | CC\_RGBINIT; // Флаги диалога (полный выбор цвета и начальное значение)

#pragma endregion

#pragma region Color Choose Brush

HWND hButton2 = CreateWindow(L"BUTTON", L"Цвет заливки", 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 Slider Scale

hSliderScale = CreateWindowEx(0, TRACKBAR\_CLASS, NULL, TBS\_AUTOTICKS | TBS\_ENABLESELRANGE | WS\_CHILD | WS\_VISIBLE, 0, 200, PW.x1, 40, hwnd, NULL, hInst, NULL);

SendMessage(hSliderScale, TBM\_SETRANGE, TRUE, MAKELPARAM(1, 500));

#pragma endregion

#pragma region Slider Rotation

hSliderRotation = CreateWindowEx(0, TRACKBAR\_CLASS, NULL, TBS\_AUTOTICKS | TBS\_ENABLESELRANGE | WS\_CHILD | WS\_VISIBLE, 0, 240, PW.x1, 40, hwnd, NULL, hInst, NULL);

SendMessage(hSliderRotation, TBM\_SETRANGE, TRUE, MAKELPARAM(0, 360));

#pragma endregion

#pragma region Color Choose Brush

hwndList = CreateWindowEx(0, L"LISTBOX", NULL, WS\_CHILD | WS\_VISIBLE | WS\_BORDER | LBS\_NOTIFY | WS\_VSCROLL, 0, 280, PW.x1, 250, hwnd, (HMENU)100, GetModuleHandle(NULL), NULL);

hwndDeleteItem = CreateWindow(L"BUTTON", L"Отменить", WS\_CHILD | WS\_VISIBLE, 0, 520, PW.x1 / 2, 40, hwnd, (HMENU)99, GetModuleHandle(NULL), NULL);

hwndDeleteItem = CreateWindow(L"BUTTON", L"Удалить", WS\_CHILD | WS\_VISIBLE, PW.x1 / 2, 520, PW.x1 / 2, 40, hwnd, (HMENU)101, GetModuleHandle(NULL), NULL);

hwndUpItem = CreateWindow(L"BUTTON", L"Вверх", WS\_CHILD | WS\_VISIBLE, 0, 560, PW.x1 / 2, 40, hwnd, (HMENU)102, GetModuleHandle(NULL), NULL);

hwndDownItem = CreateWindow(L"BUTTON", L"Вниз", WS\_CHILD | WS\_VISIBLE, PW.x1 / 2, 560, PW.x1 / 2, 40, hwnd, (HMENU)103, GetModuleHandle(NULL), NULL);

#pragma endregion

#pragma region Open/Save

hwndSave = CreateWindow(L"BUTTON", L"Открыть проект", WS\_CHILD | WS\_VISIBLE, 0, PW.y2 - 160, PW.x1, 40, hwnd, (HMENU)104, GetModuleHandle(NULL), NULL);

hwndSave = CreateWindow(L"BUTTON", L"Сохранить проект", WS\_CHILD | WS\_VISIBLE, 0, PW.y2 - 120, PW.x1, 40, hwnd, (HMENU)105, GetModuleHandle(NULL), NULL);

hwndSave = CreateWindow(L"BUTTON", L"Сохранить png", WS\_CHILD | WS\_VISIBLE, 0, PW.y2 - 80, PW.x1, 40, hwnd, (HMENU)106, GetModuleHandle(NULL), NULL);

#pragma endregion

break;

}

case WM\_PAINT:

{

if (isDrawing)

{

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hwnd, &ps);

Gdiplus::Graphics graphics(hdc);

Gdiplus::Pen pen(Gdiplus::Color({ GetRValue(selectedColorThickness),GetGValue(selectedColorThickness),GetBValue(selectedColorThickness) }));

pen.SetWidth(Thickness);

Gdiplus::SolidBrush brush(Gdiplus::Color({ GetRValue(selectedColorBrush),GetGValue(selectedColorBrush),GetBValue(selectedColorBrush) }));

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

DrawShape(graphics, &pen, &brush, GetKeyState(VK\_SHIFT) & 0x8000);

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) });

}

InvalidateRect(hwnd, NULL, TRUE);

}

if (selectedShape == 0)

{

isMove = true;

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

InvalidateRect(hwnd, NULL, TRUE);

}

break;

}

case WM\_MOUSEMOVE:

{

if (isDrawing)

{

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<Gdiplus::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;

InvalidateRect(hwnd, NULL, TRUE);

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;

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

if (selectedIndex != LB\_ERR)

{

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

RePaint(false, false);

}

}

}

if (LOWORD(wParam) == 1002)

{

if (ChooseColor(&ccBrush))

{

selectedColorBrush = ccBrush.rgbResult;

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

if (selectedIndex != LB\_ERR)

{

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

RePaint(false, false);

}

}

}

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);

SendMessage(hSliderScale, TBM\_SETPOS, TRUE, shapes[shapes.size() - selectedIndex - 1].scale);

SendMessage(hSliderRotation, TBM\_SETPOS, TRUE, shapes[shapes.size() - selectedIndex - 1].rotation);

SendMessage(hSliderThickness, TBM\_SETPOS, TRUE, shapes[shapes.size() - selectedIndex - 1].Thickness);

SendMessage(hSlider, TBM\_SETPOS, TRUE, shapes[shapes.size() - selectedIndex - 1].n);

}

}

if (LOWORD(wParam) == 99)

{

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

if (selectedIndex != LB\_ERR) {

SetFocus(hwndMain);

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

}

else

{

MessageBox(NULL, L"Не выбран элемент в списке!", L"Ошибка", MB\_ICONERROR | MB\_OK);

}

}

if (LOWORD(wParam) == 101)

{

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

if (selectedIndex != LB\_ERR) {

RePaint(false, true);

}

else

{

MessageBox(NULL, L"Не выбран элемент в списке!", L"Ошибка", 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"Не выбран элемент в списке!", L"Ошибка", 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"Не выбран элемент в списке!", L"Ошибка", MB\_ICONERROR | MB\_OK);

}

SetFocus(hwndMain);

}

if (LOWORD(wParam) == 104)

{

// Открываем файл для чтения в двоичном режиме

HANDLE fileHandle = CreateFile(L"shapes.dat", GENERIC\_READ, 0, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (fileHandle == INVALID\_HANDLE\_VALUE) {

MessageBox(NULL, L"Error 1: Не удалось открыть файл для чтения!", L"Ошибка", MB\_ICONERROR | MB\_OK);

return 1;

}

// Получаем размер файла

DWORD fileSize = GetFileSize(fileHandle, NULL);

// Создаем memory-mapped файл для чтения

HANDLE mapHandle = CreateFileMapping(fileHandle, NULL, PAGE\_READONLY, 0, fileSize, NULL);

if (mapHandle == NULL) {

CloseHandle(fileHandle);

MessageBox(NULL, L"Error 2: Не удалось создать memory-mapped файл для чтения!", L"Ошибка", MB\_ICONERROR | MB\_OK);

return 1;

}

CloseHandle(fileHandle);

// Отображаем файл в память для чтения

LPVOID mapView = MapViewOfFile(mapHandle, FILE\_MAP\_READ, 0, 0, 0);

if (mapView == NULL) {

CloseHandle(mapHandle);

MessageBox(NULL, L"Error 3: Не удалось отобразить файл в память для чтения!", L"Ошибка", MB\_ICONERROR | MB\_OK);

return 1;

}

CloseHandle(mapHandle);

// Определяем количество элементов (Shapes) в файле

size\_t numShapes = fileSize / sizeof(Shape);

// Очищаем вектор, если он уже содержит данные

shapes.clear();

// Копируем данные из memory-mapped файла в вектор

for (size\_t i = 0; i < numShapes; ++i) {

shapes.push\_back(\*(reinterpret\_cast<const Shape\*>(static\_cast<const char\*>(mapView) + i \* sizeof(Shape))));

}

// Закрываем memory-mapped файл и файл на диске

UnmapViewOfFile(mapView);

RePaint(false, false, true);

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

}

if (LOWORD(wParam) == 105)

{

// Создаем memory-mapped файл

HANDLE mapHandle = CreateFileMapping(INVALID\_HANDLE\_VALUE, NULL, PAGE\_READWRITE, 0, sizeof(Shape) \* shapes.size(), L"MyMappedFile");

if (mapHandle == NULL) {

MessageBox(NULL, L"Error 1. Не удалось создать memory-mapped файл!", L"Ошибка", MB\_ICONERROR | MB\_OK);

return 1;

}

// Отображаем файл в память

LPVOID mapView = MapViewOfFile(mapHandle, FILE\_MAP\_WRITE, 0, 0, 0);

if (mapView == NULL) {

CloseHandle(mapHandle);

MessageBox(NULL, L"Error 2. Не удалось отобразить файл в память!", L"Ошибка", MB\_ICONERROR | MB\_OK);

return 1;

}

// Закрываем memory-mapped файл

CloseHandle(mapHandle);

// Копируем данные из вектора в memory-mapped файл

memcpy(mapView, &shapes[0], sizeof(Shape) \* shapes.size());

// Открываем файл для записи на диск в двоичном режиме

HANDLE fileHandle = CreateFile(L"shapes.dat", GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (fileHandle == INVALID\_HANDLE\_VALUE) {

MessageBox(NULL, L"Error 3. Не удалось открыть файл для записи!", L"Ошибка", MB\_ICONERROR | MB\_OK);

return 1;

}

// Читаем данные из memory-mapped файла и записываем их на диск

DWORD bytesWritten;

if (!WriteFile(fileHandle, mapView, sizeof(Shape) \* shapes.size(), &bytesWritten, NULL)) {

MessageBox(NULL, L"Error 4. Не удалось записать данные на диск!", L"Ошибка", MB\_ICONERROR | MB\_OK);

return 1;

}

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

// Закрываем memory-mapped файл и файл на диске

UnmapViewOfFile(mapView);

CloseHandle(fileHandle);

}

if (LOWORD(wParam) == 106)

{

HDC screenDC = GetDC(hwndMain);

RECT windowRect;

GetWindowRect(hwndMain, &windowRect);

HBITMAP hBitmap = CreateCompatibleBitmap(screenDC, PW.Width - 16, PW.Height - 35);

HDC memDC = CreateCompatibleDC(screenDC);

SelectObject(memDC, hBitmap);

BitBlt(memDC, 0, 0, windowRect.right- windowRect.left+PW.x1, windowRect.bottom - windowRect.top, screenDC, PW.x1, PW.y1, SRCCOPY);

Gdiplus::Bitmap bitmap(hBitmap, NULL);

CLSID pngClsid;

GetEncoderClsid(L"image/png", &pngClsid);

bitmap.Save(L"paint.png", &pngClsid, NULL);

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

DeleteObject(hBitmap);

DeleteDC(memDC);

}

break;

}

case WM\_HSCROLL:

{

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

if (lParam == (LPARAM)hSlider)

{

n = SendMessage(hSlider, TBM\_GETPOS, 0, 0);selectedColorThickness = ccThickness.rgbResult;

if (selectedIndex != LB\_ERR)

{

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

RePaint(false, false);

}

}

if (lParam == (LPARAM)hSliderThickness)

{

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

if (selectedIndex != LB\_ERR)

{

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

RePaint(false, false);

}

}

if (lParam == (LPARAM)hSliderScale) {

if (selectedIndex != LB\_ERR)

{

shapes[shapes.size() - selectedIndex - 1].scale = SendMessage(hSliderScale, TBM\_GETPOS, 0, 0);

RePaint(false, false);

}

}

if (lParam == (LPARAM)hSliderRotation) {

if (selectedIndex != LB\_ERR)

{

shapes[shapes.size() - selectedIndex - 1].rotation = SendMessage(hSliderRotation, TBM\_GETPOS, 0, 0);

RePaint(false, false);

}

}

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, bool list)

{

if (list) {

SendMessage(hwndList, LB\_RESETCONTENT, 0, 0);

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

{

wchar\_t buffer[30];

swprintf(buffer, 30, L"(%d, %d), (%d, %d)", shapes[a].rect.left, shapes[a].rect.top, shapes[a].rect.right, shapes[a].rect.bottom);

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

}

}

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 bufSCT = selectedColorThickness;

COLORREF bufSCB = selectedColorBrush;

RECT bufCurrentShape = currentShape;

int bufSelectedShape = selectedShape;

int bufn = n;

std::vector<Gdiplus::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++) {

n = shapes[a].n;

selectedShape = shapes[a].selectedShape;

pen = shapes[a].pen;

selectedColorThickness = shapes[a].selectedColorThickness;

selectedColorBrush = shapes[a].selectedColorBrush;

currentShape = shapes[a].rect;

Gdiplus::Graphics graphics(hdc);

Gdiplus::Pen pen(Gdiplus::Color({ GetRValue(selectedColorThickness),GetGValue(selectedColorThickness),GetBValue(selectedColorThickness) }));

pen.SetWidth(shapes[a].Thickness);

Gdiplus::SolidBrush brush(Gdiplus::Color({ GetRValue(selectedColorBrush),GetGValue(selectedColorBrush),GetBValue(selectedColorBrush) }));

DrawShape(graphics, &pen, &brush, shapes[a].isCorrect, shapes[a].scale, shapes[a].rotation);

}

ReleaseDC(hwndMain, hdc);

selectedColorThickness = bufSCT;

selectedColorBrush = bufSCB;

currentShape = bufCurrentShape;

selectedShape = bufSelectedShape;

n = bufn;

pen = bufpen;

bufpen.clear();

}

}

void DrawShape(Gdiplus::Graphics& graphics, Gdiplus::Pen\* penPlus, Gdiplus::SolidBrush\* brush, bool isCorrect, int scale, int rotation)

{

long x1 = currentShape.left;

long y1 = currentShape.top;

long x2 = currentShape.right;

long y2 = currentShape.bottom;

float s = scale / 100.0;

graphics.TranslateTransform((x2 + x1) / 2, (y2 + y1) / 2);

graphics.ScaleTransform(s, s);

graphics.RotateTransform(rotation);

graphics.TranslateTransform(-(x2 + x1) / 2, -(y2 + y1) / 2);

switch (selectedShape) {

// Круг

case 1:

{

if (isCorrect) {

int centerX = (x1 + x2) / 2;

int centerY = (y1 + y2) / 2;

int radius = (x2 - x1) / 2;

x1 = centerX - radius;

x2 = centerX + radius;

y1 = centerY - radius;

y2 = centerY + radius;

Gdiplus::Rect ellipseRect(x1, y1, x2 - x1, y2 - y1);

graphics.FillEllipse(brush, ellipseRect);

graphics.DrawEllipse(penPlus, ellipseRect);

}

else {

Gdiplus::Rect ellipseRect(x1, y1, x2 - x1, y2 - y1);

graphics.FillEllipse(brush, ellipseRect);

graphics.DrawEllipse(penPlus, ellipseRect);

}

break;

}

// N-угольник

case 2: {

if (isCorrect)

{

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

int radius, x;

int y = y1;

std::vector<Gdiplus::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));

}

graphics.FillPolygon(brush, &vertices[0], vertices.size());

graphics.DrawPolygon(penPlus, &vertices[0], vertices.size());

}

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<Gdiplus::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 });

}

graphics.FillPolygon(brush, &vertices[0], vertices.size());

graphics.DrawPolygon(penPlus, &vertices[0], vertices.size());

}

break;

}

// Прямая

case 3: {

Gdiplus::Point startPoint(x1, y1);

Gdiplus::Point endPoint(x2, y2);

graphics.DrawLine(penPlus, startPoint, endPoint);

break;

}

// Карандаш

case 4:

{

graphics.DrawCurve(penPlus, &pen[0], pen.size());

break;

}

}

FillRectWindow();

}

int GetEncoderClsid(const WCHAR\* format, CLSID\* pClsid)

{

UINT num = 0; // number of image encoders

UINT size = 0; // size of the image encoder array in bytes

Gdiplus::ImageCodecInfo\* pImageCodecInfo = NULL;

Gdiplus::GetImageEncodersSize(&num, &size);

if (size == 0)

return -1; // Failure

pImageCodecInfo = (Gdiplus::ImageCodecInfo\*)(malloc(size));

if (pImageCodecInfo == NULL)

return -1; // Failure

GetImageEncoders(num, size, pImageCodecInfo);

for (UINT j = 0; j < num; ++j)

{

if (wcscmp(pImageCodecInfo[j].MimeType, format) == 0)

{

\*pClsid = pImageCodecInfo[j].Clsid;

free(pImageCodecInfo);

return j; // Success

}

}

free(pImageCodecInfo);

return -1; // Failure

}