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 № 2

на тему «Расширенное использование оконного интерфейса Win 32 и GDI. Формирование сложных изображений, создание и использование элементов управления, обработка различных сообщений, механизм перехвата сообщений (winhook)»

Выполнил:

студент гр. 153504

Михалевич М.П.

Проверил:

Гриценко Н.Ю.

Минск 2023

СОДЕРЖАНИЕ

[1 Цели работы 3](#_Toc146655820)

[2 Краткие теоретические сведения 4](#_Toc146655821)

[3 Результаты выполнения лабораторной работы 5](#_Toc146655822)

[Вывод 6](#_Toc146655823)

[Приложение А 7](#_Toc146655824)

## 1 Цели работы

Целью данной работы является создание текстового редактора с использованием Win32 API, обеспечивающего базовую функциональность открытия, сохранения и редактирования текстовых файлов. Редактор должен также поддерживать операции копирования, вырезания и вставки текста, а также изменение цвета текста и фона.

## 2 Краткие теоретические сведения

Win32 API - это набор функций и процедур, предоставляемых операционной системой Windows для разработки приложений. Он служит связующим звеном между приложением и аппаратным обеспечением компьютера, обеспечивая возможность создания разнообразных оконных приложений для Windows. В рамках этого интерфейса разработчики получают доступ к широкому спектру возможностей, включая обработку событий, взаимодействие с системными ресурсами (например, файлами и памятью), и многие другие задачи.

Одной из ключевых концепций Win32 API является регистрация класса окна. Этот процесс определяет, как окно будет отображаться и какие характеристики оно будет иметь, такие как стиль, иконка и курсор.

Создание окна - еще один фундаментальный аспект приложения. С использованием функции CreateWindow, приложение создает окно с заданными параметрами, включая стиль, размер и положение на экране. Это окно становится основным элементом пользовательского интерфейса приложения.

Для обработки событий и взаимодействия с пользователем разработчики используют функцию WndProc, которая обрабатывает различные сообщения и события, поступающие от операционной системы. Это позволяет приложению реагировать на действия пользователя, такие как нажатия клавиш и движения мыши.

Важным элементом создания интерфейса приложения являются элементы управления, такие как кнопки и текстовые поля, которые также создаются с использованием функции CreateWindow и добавляются в окно. Эти элементы обеспечивают взаимодействие пользователя с приложением.

Особенностью данного текстового редактора является возможность настройки его внешнего оформления. Пользователи могут менять цвет текста, стиль текста и цвет фона редактора, что делает приложение более персонализированным и удовлетворяющим их предпочтениям. Это достигается через использование структуры CHOOSECOLOR, функции ChooseColor() для выбора цветов и обновления интерфейса с помощью InvalidateRect(). Также есть возможность настройки стиля текста и цвета фона через интерфейс приложения.

## 3 Результаты выполнения лабораторной работы

В ходе выполнения лабораторной работы был разработан текстовый редактор с поддержкой настраиваемых тем оформления (стили текста, цвета фона). При запуске программы пользователь может изменить цвет, цвет фона, размер и стиль текста. Работа программы показана рисунке 3.1.



Рисунок 3.1 – Результат работы программы

## ВЫВОД

По результатам выполнения данной лабораторной работы были изучены основы программирования в Win32 API, освоены ключевые понятия и компоненты этого интерфейса. Разработанное оконное приложение обладает минимальной, но достаточной функциональностью для чтения и редактирования текстовых документов, а также с поддержкой настраиваемых тем оформления (стили текста, цвета фона).

## ПРИЛОЖЕНИЕ А

**(обязательное)**

**Листинг кода**

#include "framework.h"

#include "helpers.h"

#include "lab1.h"

#include <commdlg.h>

#include <fstream>

#include <richedit.h>

#define MAX\_LOADSTRING 100

// Глобальные переменные:

HINSTANCE hInst; // текущий экземпляр

WCHAR szTitle[MAX\_LOADSTRING]; // Текст строки заголовка

WCHAR szWindowClass[MAX\_LOADSTRING]; // имя класса главного окна

HWND hEdit; //

// Отправить объявления функций, включенных в этот модуль кода:

ATOM MyRegisterClass(HINSTANCE hInstance);

BOOL InitInstance(HINSTANCE, int);

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

INT\_PTR CALLBACK About(HWND, UINT, WPARAM, LPARAM);

int APIENTRY wWinMain(\_In\_ HINSTANCE hInstance,

\_In\_opt\_ HINSTANCE hPrevInstance,

\_In\_ LPWSTR lpCmdLine,

\_In\_ int nCmdShow)

{

UNREFERENCED\_PARAMETER(hPrevInstance);

UNREFERENCED\_PARAMETER(lpCmdLine);

// TODO: Разместите код здесь.

// Инициализация глобальных строк

LoadStringW(hInstance, IDS\_APP\_TITLE, szTitle, MAX\_LOADSTRING);

LoadStringW(hInstance, IDC\_LAB1, szWindowClass, MAX\_LOADSTRING);

LoadLibrary(TEXT("riched20.dll"));

LoadLibrary(TEXT("Msftedit.dll"));

MyRegisterClass(hInstance);

// Выполнить инициализацию приложения:

if (!InitInstance (hInstance, nCmdShow))

{

return FALSE;

}

HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC\_LAB1));

MSG msg;

// Цикл основного сообщения:

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

{

if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

return (int) msg.wParam;

}

//

// ФУНКЦИЯ: MyRegisterClass()

//

// ЦЕЛЬ: Регистрирует класс окна.

//

ATOM MyRegisterClass(HINSTANCE hInstance)

{

WNDCLASSEXW 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(hInstance, MAKEINTRESOURCE(IDI\_LAB1));

wcex.hCursor = LoadCursor(nullptr, IDC\_ARROW);

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

wcex.lpszMenuName = MAKEINTRESOURCEW(IDC\_LAB1);

wcex.lpszClassName = szWindowClass;

wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI\_SMALL));

return RegisterClassExW(&wcex);

}

//

// ФУНКЦИЯ: InitInstance(HINSTANCE, int)

//

// ЦЕЛЬ: Сохраняет маркер экземпляра и создает главное окно

//

// КОММЕНТАРИИ:

//

// В этой функции маркер экземпляра сохраняется в глобальной переменной, а также

// создается и выводится главное окно программы.

//

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)

{

hInst = hInstance; // Сохранить маркер экземпляра в глобальной переменной

HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, 0, CW\_USEDEFAULT, 0, nullptr, nullptr, hInstance, nullptr);

if (!hWnd)

{

return FALSE;

}

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

return TRUE;

}

//

// ФУНКЦИЯ: WndProc(HWND, UINT, WPARAM, LPARAM)

//

// ЦЕЛЬ: Обрабатывает сообщения в главном окне.

//

// WM\_COMMAND - обработать меню приложения

// WM\_PAINT - Отрисовка главного окна

// WM\_DESTROY - отправить сообщение о выходе и вернуться

//

//

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

{

static WCHAR fileName[MAX\_PATH] = L"";

static COLORREF rgbCurrentText = RGB(0, 0, 0);

static COLORREF rgbCurrentBackground = RGB(255, 255, 255);

switch (message)

{

case WM\_CREATE: // Сообщение о создании окна

{

CreateWindow(L"BUTTON", L"Копировать", WS\_CHILD | WS\_VISIBLE, 10, 10, 100, 30, hWnd, (HMENU)COPY\_BUTTON, hInst, NULL);

CreateWindow(L"BUTTON", L"Вставить", WS\_CHILD | WS\_VISIBLE, 120, 10, 100, 30, hWnd, (HMENU)PASTE\_BUTTON, hInst, NULL);

CreateWindow(L"BUTTON", L"Вырезать", WS\_CHILD | WS\_VISIBLE, 230, 10, 100, 30, hWnd, (HMENU)CUT\_BUTTON, hInst, NULL);

CreateWindow(L"BUTTON", L"Очистить", WS\_CHILD | WS\_VISIBLE, 340, 10, 100, 30, hWnd, (HMENU)CLEAN\_BUTTON, hInst, NULL);

CreateWindow(L"BUTTON", L"Цвет текста", WS\_CHILD | WS\_VISIBLE, 500, 10, 100, 30, hWnd, (HMENU)CHANGE\_TEXT\_COLOR\_BUTTON, hInst, NULL);

CreateWindow(L"BUTTON", L"Цвет фона", WS\_CHILD | WS\_VISIBLE, 610, 10, 100, 30, hWnd, (HMENU)CHANGE\_BACKGROUND\_COLOR\_BUTTON, hInst, NULL);

CreateWindowEx(

WS\_EX\_CLIENTEDGE,

RICHEDIT\_CLASS,

L"",

WS\_CHILD | WS\_VISIBLE,

720,

10,

150,

30,

hWnd,

(HMENU)FONT\_SIZE\_EDIT,

hInst,

NULL

);

hEdit = CreateWindowEx(

WS\_EX\_CLIENTEDGE,

RICHEDIT\_CLASS,

L"",

WS\_CHILD | WS\_VISIBLE | WS\_VSCROLL | ES\_LEFT | ES\_MULTILINE | ES\_AUTOVSCROLL | ES\_WANTRETURN,

10,

100,

500,

400,

hWnd,

NULL,

hInst,

NULL

);

CreateWindow(L"BUTTON", L"Изменить размер", WS\_CHILD | WS\_VISIBLE, 720, 50, 150, 30, hWnd, (HMENU)CHANGE\_FONT\_SIZE\_BUTTON, hInst, NULL);

CreateWindow(L"BUTTON", L"Нормальный", WS\_CHILD | WS\_VISIBLE, 880, 10, 100, 25, hWnd, (HMENU)NORMAL\_TEXT\_BUTTON, hInst, NULL);

CreateWindow(L"BUTTON", L"Подчеркнутый", WS\_CHILD | WS\_VISIBLE, 880, 40, 100, 25, hWnd, (HMENU)UNDERLINE\_TEXT\_BUTTON, hInst, NULL);

CreateWindow(L"BUTTON", L"Наклонный", WS\_CHILD | WS\_VISIBLE, 880, 70, 100, 25, hWnd, (HMENU)ITALIC\_TEXT\_BUTTON, hInst, NULL);

CreateWindow(L"BUTTON", L"Жирный", WS\_CHILD | WS\_VISIBLE, 1000, 70, 70, 25, hWnd, (HMENU)BOLD\_TEXT\_BUTTON, hInst, NULL);

}

break;

case WM\_PAINT:

{

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hWnd, &ps);

EndPaint(hWnd, &ps);

}

break;

case WM\_COMMAND:

{

int wmId = LOWORD(wParam);

// Разобрать выбор в меню:

switch (wmId)

{

case EXIT:

DestroyWindow(hWnd);

break;

case OPEN\_BUTTON:

{

std::wstring openedFileName = HandleOpen(hWnd, hEdit);

wcscpy\_s(fileName, openedFileName.c\_str());

break;

}

case SAVE\_BUTTON:

{

if (wcslen(fileName) > 0)

{

WCHAR szNewContent[10000];

GetWindowText(hEdit, szNewContent, 10000);

std::wstring FilteredContent = FilterContent(szNewContent);

SaveToFile(hWnd, fileName, FilteredContent.c\_str());

}

else

{

HandleSaveAs(hWnd, hEdit);

}

}

break;

case SAVE\_AS\_BUTTON:

HandleSaveAs(hWnd, hEdit);

break;

case COPY\_BUTTON:

SendMessage(hEdit, WM\_COPY, 0, 0);

break;

case PASTE\_BUTTON:

SendMessage(hEdit, WM\_PASTE, 0, 0);

break;

case CUT\_BUTTON:

SendMessage(hEdit, WM\_CUT, 0, 0);

break;

case CLEAN\_BUTTON:

SetWindowText(hEdit, L"");

break;

case CHANGE\_TEXT\_COLOR\_BUTTON:

{

CHARFORMAT2 cf;

ZeroMemory(&cf, sizeof(cf));

cf.cbSize = sizeof(cf);

cf.dwMask = CFM\_COLOR | CFM\_BACKCOLOR;

SendMessage(hEdit, EM\_GETCHARFORMAT, SCF\_SELECTION, (LPARAM)&cf);

DWORD currentBackgroundColor = cf.crBackColor;

CHOOSECOLOR cc;

static COLORREF acrCustClr[16];

ZeroMemory(&cc, sizeof(cc));

cc.lStructSize = sizeof(cc);

cc.hwndOwner = hWnd;

cc.lpCustColors = (LPDWORD)acrCustClr;

cc.rgbResult = rgbCurrentText;

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

if (ChooseColor(&cc) == TRUE)

{

rgbCurrentText = cc.rgbResult;

CHARRANGE selRange;

SendMessage(hEdit, EM\_EXGETSEL, 0, (LPARAM)&selRange);

if (selRange.cpMin != selRange.cpMax)

{

CHARFORMAT2 cf;

ZeroMemory(&cf, sizeof(cf));

cf.cbSize = sizeof(cf);

cf.dwMask = CFM\_COLOR;

cf.crBackColor = currentBackgroundColor;

cf.crTextColor = rgbCurrentText;

SendMessage(hEdit, EM\_SETCHARFORMAT, SCF\_SELECTION, (LPARAM)&cf);

}

}

}

break;

case CHANGE\_BACKGROUND\_COLOR\_BUTTON:

{

CHARFORMAT2 cf;

ZeroMemory(&cf, sizeof(cf));

cf.cbSize = sizeof(cf);

cf.dwMask = CFM\_COLOR | CFM\_BACKCOLOR;

SendMessage(hEdit, EM\_GETCHARFORMAT, SCF\_SELECTION, (LPARAM)&cf);

DWORD currentTextColor = cf.crTextColor;

// Prepare and open the CHOOSECOLOR dialog for background color

CHOOSECOLOR cc;

static COLORREF acrCustClr[16]; // Array of custom colors

DWORD rgbCurrent;

ZeroMemory(&cc, sizeof(cc));

cc.lStructSize = sizeof(cc);

cc.hwndOwner = hWnd;

cc.lpCustColors = (LPDWORD)acrCustClr;

cc.rgbResult = cf.crBackColor; // use current background color

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

if (ChooseColor(&cc) == TRUE)

{

rgbCurrent = cc.rgbResult; // Get the selected background color

cf.dwMask = CFM\_COLOR | CFM\_BACKCOLOR;

cf.dwEffects &= ~CFE\_AUTOBACKCOLOR; // Clear the auto background color flag

cf.crTextColor = currentTextColor; // Preserve the original text color

cf.crBackColor = rgbCurrent; // Update background color

SendMessage(hEdit, EM\_SETCHARFORMAT, SCF\_SELECTION, (LPARAM)&cf);

}

}

break;

case CHANGE\_FONT\_SIZE\_BUTTON:

{

// Retrieve the font size from the Edit Control

wchar\_t buf[100];

GetDlgItemTextW(hWnd, FONT\_SIZE\_EDIT, buf, sizeof(buf));

int selectedSize = \_wtoi(buf);

// Check if the input is numeric and in a valid range

if (selectedSize <= 0 || !isdigit(buf[0]) || selectedSize < 5 || selectedSize > 72)

{

MessageBox(hWnd, L"Введите число между 5 и 72!", L"Ошибка", MB\_OK);

return 0;

}

// Convert size to twips

LONG newSizeInTwips = selectedSize \* 20;

// Prepare CHARFORMAT2 structure

CHARFORMAT2 cf;

ZeroMemory(&cf, sizeof(cf));

cf.cbSize = sizeof(cf);

cf.dwMask = CFM\_SIZE;

cf.yHeight = newSizeInTwips;

SendMessage(hEdit, EM\_SETCHARFORMAT, SCF\_SELECTION, (LPARAM)&cf);

}

break;

case NORMAL\_TEXT\_BUTTON:

{

CHARFORMAT2 cf;

ZeroMemory(&cf, sizeof(cf));

cf.cbSize = sizeof(cf);

cf.dwMask = CFM\_BOLD | CFM\_ITALIC | CFM\_UNDERLINE;

cf.dwEffects = 0;

SendMessage(hEdit, EM\_SETCHARFORMAT, SCF\_SELECTION, (LPARAM)&cf);

}

break;

case BOLD\_TEXT\_BUTTON:

{

CHARFORMAT2 cf;

ZeroMemory(&cf, sizeof(cf));

cf.cbSize = sizeof(cf);

cf.dwMask = CFM\_BOLD;

cf.dwEffects = CFE\_BOLD;

SendMessage(hEdit, EM\_SETCHARFORMAT, SCF\_SELECTION, (LPARAM)&cf);

}

break;

case ITALIC\_TEXT\_BUTTON:

{

CHARFORMAT2 cf;

ZeroMemory(&cf, sizeof(cf));

cf.cbSize = sizeof(cf);

cf.dwMask = CFM\_ITALIC;

cf.dwEffects = CFE\_ITALIC;

SendMessage(hEdit, EM\_SETCHARFORMAT, SCF\_SELECTION, (LPARAM)&cf);

}

break;

case UNDERLINE\_TEXT\_BUTTON:

{

CHARFORMAT2 cf;

ZeroMemory(&cf, sizeof(cf));

cf.cbSize = sizeof(cf);

cf.dwMask = CFM\_UNDERLINE;

cf.dwEffects = CFE\_UNDERLINE;

SendMessage(hEdit, EM\_SETCHARFORMAT, SCF\_SELECTION, (LPARAM)&cf);

}

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

}

break;

case WM\_KEYDOWN:

{

if (GetKeyState(VK\_CONTROL) & 0x8000)

{

switch (wParam)

{

case 'S': // Ctrl + S

PostMessage(hWnd, WM\_COMMAND, MAKEWPARAM(SAVE\_BUTTON, 0), 0);

return 0;

case 'C': // Ctrl + C

SendMessage(hEdit, WM\_COPY, 0, 0);

return 0;

case 'V': // Ctrl + V

SendMessage(hEdit, WM\_PASTE, 0, 0);

return 0;

case 'X': // Ctrl + X

SendMessage(hEdit, WM\_CUT, 0, 0);

return 0;

}

}

}

break;

case WM\_SIZE:

{

RECT rc;

GetClientRect(hWnd, &rc);

int windowWidth = rc.right - rc.left;

int windowHeight = rc.bottom - rc.top;

int editHeight = windowHeight - 120;

SetWindowPos(

hEdit,

NULL,

10,

100,

windowWidth - 20,

editHeight,

SWP\_NOZORDER

);

}

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

return 0;

}

#include "helpers.h"

#include <richedit.h>

void SaveToFile(HWND hWnd, const WCHAR\* fileName, const WCHAR\* content)

{

std::wofstream file(fileName);

file << content;

file.close();

MessageBox(hWnd, L"Файл успешно сохранен!", L"Уведомление", MB\_OK);

}

std::wstring FilterContent(const WCHAR\* content)

{

std::wstring FilteredContent;

for (int i = 0; content[i] != L'\0'; i++) {

if (content[i] != L'\n') {

FilteredContent += content[i];

}

}

return FilteredContent;

}

void HandleSaveAs(HWND hWnd, HWND hEdit)

{

WCHAR fileName[MAX\_PATH] = L"";

WCHAR szNewContent[10000];

GetWindowText(hEdit, szNewContent, sizeof(szNewContent) / sizeof(WCHAR));

std::wstring FilteredContent = FilterContent(szNewContent);

OPENFILENAME ofn;

ZeroMemory(&ofn, sizeof(ofn));

ofn.lStructSize = sizeof(ofn);

ofn.hwndOwner = hWnd;

ofn.lpstrFilter = L"Текстовые файлы (\*.txt)\0\*.txt\0";

ofn.lpstrFile = fileName;

ofn.nMaxFile = sizeof(fileName) / sizeof(WCHAR);

ofn.Flags = OFN\_EXPLORER | OFN\_PATHMUSTEXIST | OFN\_HIDEREADONLY;

if (GetSaveFileName(&ofn))

{

SaveToFile(hWnd, fileName, FilteredContent.c\_str());

}

}

std::wstring HandleOpen(HWND hWnd, HWND hEdit)

{

WCHAR fileName[MAX\_PATH] = L"";

WCHAR fileContent[10000] = L"";

OPENFILENAME ofn;

ZeroMemory(&ofn, sizeof(ofn));

ofn.lStructSize = sizeof(ofn);

ofn.hwndOwner = hWnd;

ofn.lpstrFilter = L"Текстовые файлы (\*.txt)\0\*.txt\0";

ofn.lpstrFile = fileName;

ofn.nMaxFile = sizeof(fileName) / sizeof(WCHAR);

ofn.Flags = OFN\_EXPLORER | OFN\_FILEMUSTEXIST | OFN\_HIDEREADONLY;

if (GetOpenFileName(&ofn))

{

std::wifstream file(fileName);

if (file.is\_open())

{

file.get(fileContent, sizeof(fileContent) / sizeof(WCHAR), L'\0');

file.close();

WCHAR modifiedContent[10000] = L"";

int j = 0;

for (int i = 0; fileContent[i] != L'\0'; i++) {

if (fileContent[i] == L'\n') {

modifiedContent[j++] = '\r';

}

modifiedContent[j++] = fileContent[i];

if (fileContent[i] == L'\r') {

modifiedContent[j++] = '\n';

}

}

SetWindowText(hEdit, modifiedContent);

return fileName;

}

else

{

MessageBox(hWnd, L"Не удалось открыть файл!", L"Ошибка", MB\_OK);

}

}

}

void ChangeTextColor(HWND hEdit, COLORREF color) {

// Get the current selection

CHARRANGE selRange;

SendMessage(hEdit, EM\_EXGETSEL, 0, (LPARAM)&selRange);

// Check if there's any selected text

if (selRange.cpMin != selRange.cpMax)

{

// Set the desired text color (For example, RGB(255, 0, 0) for red)

CHARFORMAT2 cf;

ZeroMemory(&cf, sizeof(cf));

cf.cbSize = sizeof(cf);

cf.dwMask = CFM\_COLOR;

cf.crTextColor = color;

// Set the new CHARFORMAT to the control

SendMessage(hEdit, EM\_SETCHARFORMAT, SCF\_SELECTION, (LPARAM)&cf);

}

}