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 №2

на тему

Расширенное использование оконного интерфейса Win32 и GDI.

Выполнил: студент группы 153503

Степанов Владислав Николаевич

Проверил: Гриценко Никита Юрьевич

Минск 2023

**СОДЕРЖАНИЕ**

[1 Постановка задачи 3](#_gjdgxs)

[2 Краткие теоретические сведения 4](#_30j0zll)

[3 Результаты выполнения лабораторной работы 6](#_1fob9te)

[Выводы 7](#_tyjcwt)

[Список использованных источников 8](#_3dy6vkm)

[Приложение А (обязательное) Листинг кода 9](#_1t3h5sf)

## 1 ПОСТАНОВКА ЗАДАЧИ

Целью выполнения второй лабораторной работы было расширение функциональности нашего текстового редактора, разработанного с использованием Win32 API. В рамках этой лабораторной работы были внедрены новые функциональные элементы, включая окно для поиска текста, доступное через меню и с использованием горячей клавиши. Также были добавлены кнопки для смены шрифта и цветовой темы в редакторе, обогащая возможности пользовательского интерфейса и делая редактор более удобным и настраиваемым для пользователей.

## 2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Win32 API (Windows API) является мощным инструментарием для разработки приложений под операционной системой Windows. Этот набор функций и интерфейсов предоставляет разработчикам доступ к различным функциональным возможностям Windows, позволяя создавать приложения, которые взаимодействуют с операционной системой, управляют окнами, обрабатывают сообщения, работают с файлами и реестром, а также выполняют множество других операций. Win32 API играет ключевую роль в разработке приложений для Windows, обеспечивая полный контроль над поведением приложений.

Одним из важных компонентов Win32 API является GDI (Graphics Device Interface). GDI отвечает за графический вывод и взаимодействие с графическим оборудованием. Он предоставляет функции для создания и рисования графических объектов, управления цветами и шрифтами, а также реализации различных графических эффектов. GDI позволяет разработчикам создавать интерфейсы пользовательских приложений, включая окна, кнопки, текст и другие графические элементы. Он обеспечивает высокое качество визуализации и интерактивность приложений, делая их более привлекательными и функциональными для пользователей.

Совместное использование Win32 API и GDI открывает широкие возможности для создания мощных и интерактивных графических приложений под Windows. Разработчики могут использовать функционал GDI для создания и редактирования изображений, а также для управления графическими элементами интерфейса.

Кроме того, в данной лабораторной работе были использованы винхуки (winhook). Винхуки представляют собой механизм перехвата сообщений и событий в операционной системе. Они позволяют приложению перехватывать и обрабатывать сообщения, отправляемые другим приложениям или операционной системой. В данной работе винхуки использовались для создания дополнительной функциональности, такой как обработка горячих клавиш и перехват сообщений о нажатии клавиш, что обогатило пользовательский опыт и расширило возможности редактора.

Таким образом, с использованием Win32 API, GDI и винхуков, данная лабораторная работа предоставляет пример создания приложения с продвинутой графикой, интерфейсом и обработкой событий, что делает его удовлетворяющим для пользователей и разработчиков.

## 3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

В ходе выполнения лабораторной работы в текстовый редактор было добавлено диалоговое окно для поиска в тексте подстрок (Рисунок 1).

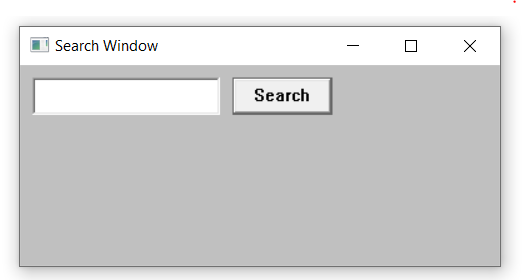


Рисунок 1 ─ Окно для поиска

При совершении поиска будет выделена часть текста, которая подходит под заданную строке. Если нажать повторно, будут произведен поиск по оставшейся строке. В конце концов выведется сообщение о том, что все совпадения закончились (Рисунок 2).

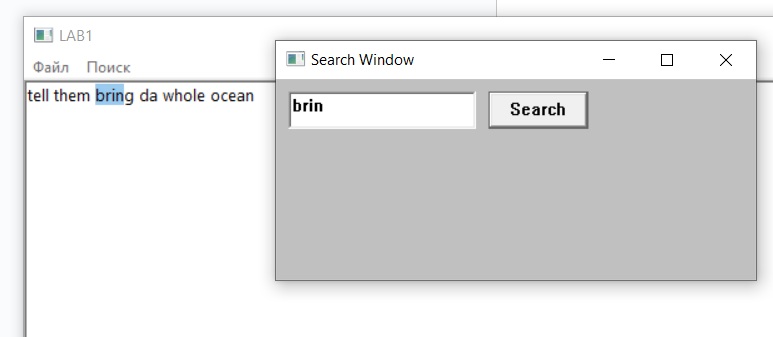


Рисунок 2 ─ Выделение найденного текста

Также был добавлена возможность изменения темы текстового редактора (Рисунок 3).

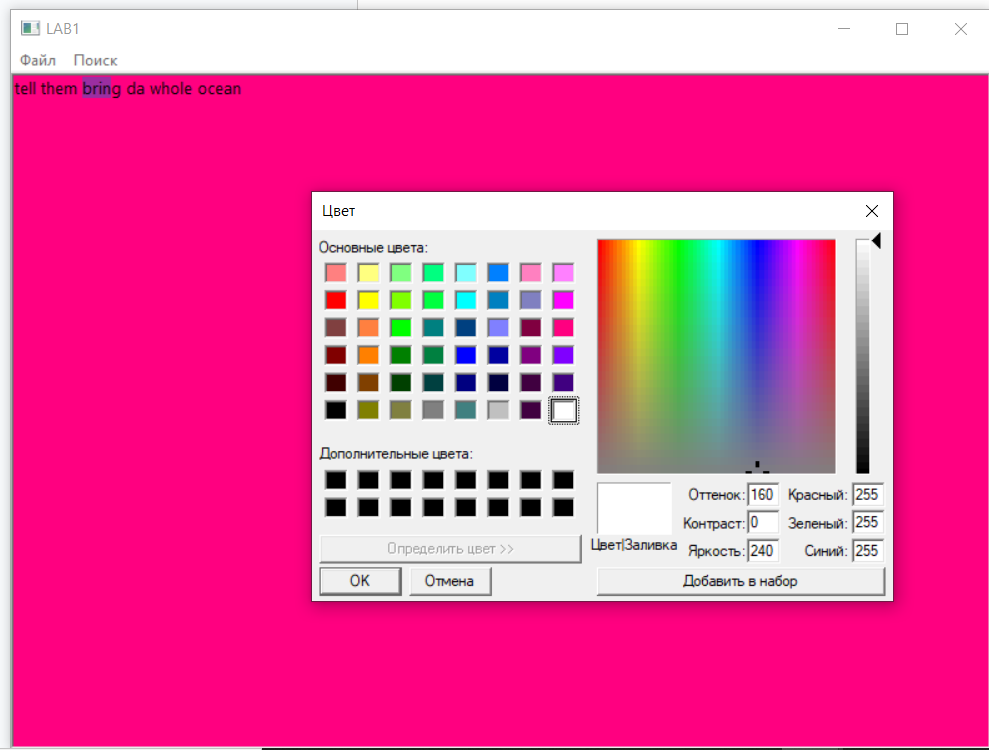


Рисунок 3 ─ Окно изменения текста

И была добавлена возможность изменения стиля текста (Рисунок 4).

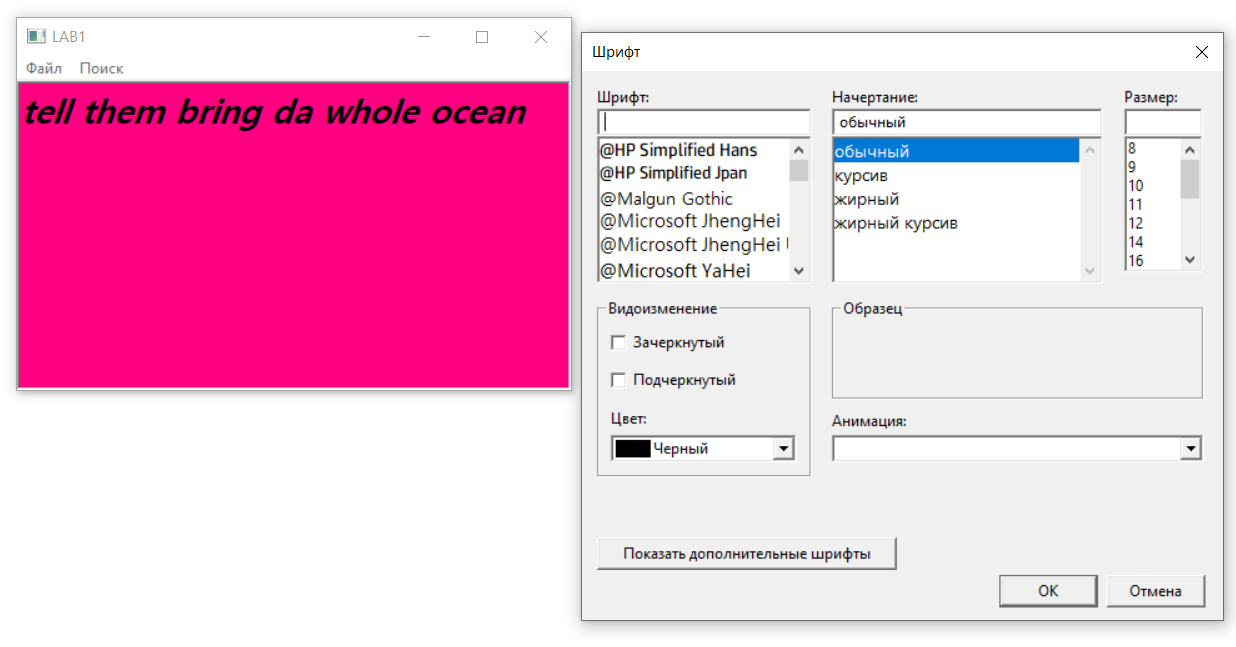


Рисунок 4 ─ Окно изменения стиля текста

## 

## ВЫВОДЫ

В результате нашей лабораторной работы был создан функциональный текстовый редактор, который позволяет редактировать текстовые документы. Мы сделали так, чтобы у него был расширенный функционал, включая возможность поиска текста, изменения шрифта и темы оформления.

Чтобы обеспечить красивый и плавный интерфейс, мы использовали специальный метод для обновления окна с частой перерисовки. Это позволяет нам плавно обновлять содержимое редактора и предотвращать мерцание или другие неприятные эффекты.

Кроме того, мы обеспечили правильную обработку изменений размеров окна, чтобы редактор мог адаптироваться к разным размерам экрана и обеспечивать удобное отображение текста.

Весь этот функционал реализован в коде WndProc, который управляет всеми действиями редактора, включая создание и работу с элементами управления, открытие и сохранение файлов, а также другие важные задачи. Таким образом, наш текстовый редактор становится полезным инструментом для редактирования текстовых документов с приятным и плавным интерфейсом.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Windows GDI [Электронный ресурс]. ─ Режим доступа: https://learn.microsoft.com/en-us/windows/win32/gdi/windows-gdi.

[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 Source.cpp

#include <windows.h>

#include <richedit.h>

#include <string>

#include "Richedit.h"

#include "commctrl.h"

#include "resource.h"

#define RICHEDIT\_CLASS L"RICHEDIT50W"

#define IDM\_OPEN 1001

#define IDM\_SAVE 1002

#define IDM\_EXIT 1003

#define IDM\_NEW 1004

#define IDC\_CLOSE\_BUTTON 1005

#define IDM\_SEARCH 1006

#define IDM\_SEARCH\_IN\_SEARCH\_WINDOW 1008

#define IDM\_CHANGE\_THEME 1009

#define IDM\_CHANGE\_FONT 1010

#define HOTKEY\_ID 1

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

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

static HWND hEdit = nullptr;

static HWND hSearchEdit = nullptr;

static int searchStartPosition = 0;

static HWND hSearchWindow = nullptr;

void OpenSearchWindow(HWND hWnd)

{

hSearchWindow = CreateWindow(

L"SearchWindowClass",

L"Search Window",

WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, CW\_USEDEFAULT, 400, 200,

nullptr,

nullptr,

GetModuleHandle(nullptr),

nullptr

);

if (hSearchWindow != nullptr)

{

hSearchEdit = CreateWindowEx(

WS\_EX\_CLIENTEDGE,

L"EDIT",

L"",

WS\_CHILD | WS\_VISIBLE | ES\_AUTOHSCROLL,

10, 10, 150, 30,

hSearchWindow,

nullptr,

GetModuleHandle(nullptr),

nullptr

);

HWND hSearchButton = CreateWindow(

L"BUTTON",

L"Search",

WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON,

170, 10, 80, 30,

hSearchWindow,

reinterpret\_cast<HMENU>(IDM\_SEARCH\_IN\_SEARCH\_WINDOW),

GetModuleHandle(nullptr),

nullptr

);

if (hSearchEdit == nullptr || hSearchButton == nullptr)

{

MessageBox(hWnd, L"Не удалось создать элементы для поиска.", L"Ошибка", MB\_OK | MB\_ICONERROR);

DestroyWindow(hSearchWindow);

}

ShowWindow(hSearchWindow, SW\_SHOWNORMAL);

}

else

{

MessageBox(hWnd, L"Не удалось создать окно поиска.", L"Ошибка", MB\_OK | MB\_ICONERROR);

}

}

void OnHotkey(HWND hWnd, WPARAM wParam, LPARAM lParam)

{

if (wParam == HOTKEY\_ID)

{

OpenSearchWindow(hWnd);

}

}

void OpenColorDialog()

{

CHOOSECOLOR cc = { sizeof(CHOOSECOLOR) };

static COLORREF custColors[16] = { 0 };

cc.rgbResult = RGB(255, 255, 255);

cc.lpCustColors = custColors;

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

if (ChooseColor(&cc))

{

SendMessage(hEdit, EM\_SETBKGNDCOLOR, FALSE, cc.rgbResult);

}

}

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

{

if (!RegisterHotKey(NULL, HOTKEY\_ID, MOD\_CONTROL, 'F'))

{

MessageBox(NULL, L"Не удалось зарегистрировать горячую клавишу!", L"Error", MB\_ICONERROR | MB\_OK);

return 1;

}

MSG msg{};

HWND hWnd{};

WNDCLASS wc{ sizeof(WNDCLASS) };

wc.lpfnWndProc = WndProc;

wc.hInstance = hInstance;

wc.cbClsExtra = 0;

wc.cbWndExtra = 0;

wc.hbrBackground = reinterpret\_cast<HBRUSH>(GetStockObject(LTGRAY\_BRUSH));

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

wc.hIcon = LoadIcon(nullptr, IDI\_APPLICATION);

wc.hInstance = hInstance;

wc.lpszClassName = L"LAB1!";

wc.lpszMenuName = nullptr;

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

WNDCLASS wcSearchWindow{ sizeof(WNDCLASS) };

wcSearchWindow.lpfnWndProc = SearchWindowProc;

wcSearchWindow.hInstance = hInstance;

wcSearchWindow.hbrBackground = reinterpret\_cast<HBRUSH>(GetStockObject(LTGRAY\_BRUSH));

wcSearchWindow.lpszClassName = L"SearchWindowClass";

if (!RegisterClass(&wcSearchWindow) || !RegisterClassW(&wc)) {

return EXIT\_FAILURE;

}

if (hWnd = CreateWindow(wc.lpszClassName, L"LAB1", WS\_OVERLAPPEDWINDOW, CW\_USEDEFAULT, CW\_USEDEFAULT, 800, 600, nullptr, nullptr, wc.hInstance, nullptr)) {

HMENU hMenu = CreateMenu();

HMENU hFileMenu = CreateMenu();

AppendMenu(hMenu, MF\_POPUP, (UINT\_PTR)hFileMenu, L"Файл");

AppendMenu(hFileMenu, MF\_STRING, IDM\_OPEN, L"Открыть");

AppendMenu(hFileMenu, MF\_STRING, IDM\_SAVE, L"Сохранить");

AppendMenu(hFileMenu, MF\_SEPARATOR, 0, NULL);

AppendMenu(hFileMenu, MF\_STRING, IDM\_EXIT, L"Выход");

AppendMenu(hFileMenu, MF\_SEPARATOR, 0, NULL);

AppendMenu(hFileMenu, MF\_STRING, IDM\_NEW, L"Новый");

AppendMenu(hFileMenu, MF\_SEPARATOR, 0, NULL);

AppendMenu(hFileMenu, MF\_STRING, IDM\_CHANGE\_THEME, L"Поменять тему");

AppendMenu(hFileMenu, MF\_STRING, IDM\_CHANGE\_FONT, L"Изменить шрифт");

AppendMenu(hMenu, MF\_STRING, IDM\_SEARCH, L"Поиск");

SetMenu(hWnd, hMenu);

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

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

TranslateMessage(&msg);

DispatchMessage(&msg);

if (msg.message == WM\_HOTKEY)

{

OnHotkey(msg.hwnd, msg.wParam, msg.lParam);

}

}

}

return EXIT\_SUCCESS;

}

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

{

switch (message)

{

case WM\_COMMAND:

switch (LOWORD(wParam))

{

case IDM\_SEARCH\_IN\_SEARCH\_WINDOW:

{

int searchQueryLength = GetWindowTextLength(hSearchEdit);

if (searchQueryLength == 0)

{

MessageBox(hWnd, L"Введите текст.", L"Error", MB\_OK | MB\_ICONERROR);

return 0;

}

wchar\_t\* searchQuery = new wchar\_t[searchQueryLength + 1];

GetWindowText(hSearchEdit, searchQuery, searchQueryLength + 1);

int mainEditLength = GetWindowTextLength(hEdit);

wchar\_t\* mainEditText = new wchar\_t[mainEditLength + 1];

GetWindowText(hEdit, mainEditText, mainEditLength + 1);

wchar\_t\* found = wcsstr(mainEditText + searchStartPosition, searchQuery);

if (found != nullptr)

{

int foundPos = found - mainEditText;

SendMessage(hEdit, EM\_SETSEL, foundPos, foundPos + searchQueryLength);

CHARFORMAT2 cf;

cf.cbSize = sizeof(cf);

cf.dwMask = CFM\_BACKCOLOR;

cf.crBackColor = RGB(255, 255, 0);

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

searchStartPosition = foundPos + searchQueryLength;

}

else

{

searchStartPosition = 0;

MessageBox(hWnd, L"No more matches found.", L"Search Complete", MB\_OK | MB\_ICONINFORMATION);

}

delete[] searchQuery;

delete[] mainEditText;

}

break;

case WM\_CLOSE:

if (hWnd == hSearchWindow)

DestroyWindow(hWnd);

break;

}

break;

case WM\_DESTROY:

if (hWnd == hSearchWindow)

hSearchWindow = nullptr;

break;

default:

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

}

return 0;

}

void ResizeEditControl(HWND hWnd, HWND hEdit)

{

RECT clientRect;

GetClientRect(hWnd, &clientRect);

SetWindowPos(hEdit, nullptr, 0, 0, clientRect.right, clientRect.bottom, SWP\_NOZORDER);

}

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

{

static wchar\_t currentFileName[MAX\_PATH] = L"";

switch (message)

{

case WM\_CREATE:

{

INITCOMMONCONTROLSEX icex;

icex.dwSize = sizeof(INITCOMMONCONTROLSEX);

icex.dwICC = ICC\_STANDARD\_CLASSES;

InitCommonControlsEx(&icex);

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

hEdit = CreateWindowEx(

WS\_EX\_CLIENTEDGE,

RICHEDIT\_CLASS,

L"",

WS\_CHILD | WS\_VISIBLE | WS\_VSCROLL | WS\_HSCROLL | ES\_MULTILINE | ES\_AUTOVSCROLL | ES\_AUTOHSCROLL | ES\_NOHIDESEL,

0, 0, 800, 600,

hWnd,

nullptr,

GetModuleHandle(nullptr),

nullptr

);

if (hEdit == nullptr)

{

MessageBox(hWnd, L"Не удалось создать элемент управления Edit.", L"Ошибка", MB\_OK | MB\_ICONERROR);

}

}

break;

case WM\_SIZE:

ResizeEditControl(hWnd, hEdit);

break;

case WM\_COMMAND:

switch (LOWORD(wParam))

{

case IDM\_CHANGE\_THEME:

OpenColorDialog();

break;

case IDM\_CHANGE\_FONT:

{

CHOOSEFONT cf = { sizeof(CHOOSEFONT) };

LOGFONT lf = { 0 };

cf.hwndOwner = hWnd;

cf.lpLogFont = &lf;

cf.Flags = CF\_SCREENFONTS | CF\_EFFECTS | CF\_INITTOLOGFONTSTRUCT;

if (ChooseFont(&cf))

{

HFONT hFont = CreateFontIndirect(&lf);

SendMessage(hEdit, WM\_SETFONT, (WPARAM)hFont, TRUE);

}

break;

}

case IDM\_OPEN:

{

OPENFILENAME ofn;

wchar\_t szFile[MAX\_PATH] = L"";

ZeroMemory(&ofn, sizeof(ofn));

ofn.lStructSize = sizeof(ofn);

ofn.hwndOwner = hWnd;

ofn.lpstrFilter = L"C++ файлы (\*.cpp;\*.txt)\0\*.cpp;\*.txt\0Все файлы (\*.\*)\0\*.\*\0";

ofn.lpstrFile = szFile;

ofn.nMaxFile = MAX\_PATH;

ofn.Flags = OFN\_FILEMUSTEXIST | OFN\_PATHMUSTEXIST;

if (GetOpenFileName(&ofn))

{

wcscpy\_s(currentFileName, MAX\_PATH, ofn.lpstrFile);

HANDLE hFile = CreateFile(ofn.lpstrFile, GENERIC\_READ, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, 0, NULL);

if (hFile != INVALID\_HANDLE\_VALUE)

{

DWORD dwFileSize = GetFileSize(hFile, NULL);

wchar\_t\* fileContent = (wchar\_t\*)malloc((dwFileSize + 1) \* sizeof(wchar\_t));

if (fileContent)

{

DWORD bytesRead;

ReadFile(hFile, fileContent, dwFileSize \* sizeof(wchar\_t), &bytesRead, NULL);

fileContent[dwFileSize / sizeof(wchar\_t)] = L'\0';

SetWindowText(hEdit, fileContent);

free(fileContent);

}

else

{

MessageBox(hWnd, L"Не удалось выделить память для содержимого файла.", L"Ошибка", MB\_OK | MB\_ICONERROR);

}

CloseHandle(hFile);

}

else

{

MessageBox(hWnd, L"Не удалось открыть выбранный файл.", L"Ошибка", MB\_OK | MB\_ICONERROR);

}

}

break;

}

case IDM\_SAVE:

{

HANDLE hFile = INVALID\_HANDLE\_VALUE;

if (currentFileName[0] != L'\0')

{

hFile = CreateFile(currentFileName, GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

}

if (hFile == INVALID\_HANDLE\_VALUE)

{

OPENFILENAME ofn;

wchar\_t szFile[MAX\_PATH] = L"";

ZeroMemory(&ofn, sizeof(ofn));

ofn.lStructSize = sizeof(ofn);

ofn.hwndOwner = hWnd;

ofn.lpstrFilter = L"C++ files (\*.cpp;\*.txt)\0\*.cpp;\*.txt\0All files (\*.\*)\0\*.\*\0";

ofn.lpstrFile = szFile;

ofn.nMaxFile = MAX\_PATH;

ofn.Flags = OFN\_OVERWRITEPROMPT;

if (GetSaveFileName(&ofn))

{

hFile = CreateFile(ofn.lpstrFile, GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (hFile != INVALID\_HANDLE\_VALUE)

{

wcscpy\_s(currentFileName, MAX\_PATH, ofn.lpstrFile);

}

else

{

MessageBox(hWnd, L"Не удалось создать выбранный файл.", L"Ошибка", MB\_OK | MB\_ICONERROR);

}

}

}

if (hFile != INVALID\_HANDLE\_VALUE)

{

int textLength = GetWindowTextLength(hEdit);

wchar\_t\* textBuffer = (wchar\_t\*)malloc((textLength + 1) \* sizeof(wchar\_t));

if (textBuffer)

{

GetWindowText(hEdit, textBuffer, textLength + 1);

DWORD bytesWritten;

WriteFile(hFile, textBuffer, textLength \* sizeof(wchar\_t), &bytesWritten, NULL);

free(textBuffer);

CloseHandle(hFile);

}

else

{

MessageBox(hWnd, L"Не удалось выделить память для текста.", L"Ошибка", MB\_OK | MB\_ICONERROR);

CloseHandle(hFile);

}

}

break;

}

case IDM\_NEW:

{

if (MessageBox(hWnd, L"Вы уверены, что хотите создать новый документ? Все несохраненные изменения будут потеряны.", L"Подтвердите действие", MB\_YESNO | MB\_ICONQUESTION) == IDYES)

{

SetWindowText(hEdit, L"");

currentFileName[0] = L'\0';

}

break;

}

case IDM\_SEARCH:

OpenSearchWindow(hWnd);

break;

case WM\_CLOSE:

CloseWindow(hWnd);

break;

case IDM\_EXIT:

PostQuitMessage(0);

break;

}

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

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

}

return 0;

}