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 №1

на тему

**ОСНОВЫ ПРОГРАММИРОВАНИЯ В WIN 32 API**

Студент В. А. Сидорко

Преподаватель Н. Ю. Гриценко

Минск 2023

СОДЕРЖАНИЕ

[1 Цель работы 3](#_Toc146752068)

[2 Теоретические сведения 4](#_Toc146752069)

[3 Результат выполнения 6](#_Toc146752070)

[Заключение 10](#_Toc146752071)

[Список использованных источников 11](#_Toc146752072)

[Приложение А (обязательное) Листинг кода 12](#_Toc146752073)

1. **ЦЕЛЬ РАБОТЫ**

Изучить основы программирования в Win32 API (обработка основных оконных сообщений) и применить полученные знания на практике при разработке оконного приложения с минимальной достаточной функциональностью. Для достижения цели будет создано оконное приложение для чтения и редактирования текстовых документов с возможностью выделения и копирования текста в буфер обмена.

1. **ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Win32 API – это исходная платформа для собственных *Windows* приложений на языке C/C++, которым требуется прямой доступ к *Windows* и оборудованию. Для этого Win32 API предоставляет множество 32-разрядных функций, структур данных и числовых констант, следующих соглашениям языка Си.

Центральным объектом в операционной системе *Windows* является окно – программная конструкция, которая может отображаться на экране, отвечать на события пользователя или операционной системы. Взаимодействие программы с окном происходит с помощью дескриптора, который используется программой для идентификации окна.

Главное окно приложения обычно имеет рамку с заголовком строки, кнопками управления окном приложения и другими стандартными элементами пользовательского интерфейса, обработка сообщений которых выполняется операционной системой, а также клиентскую область, управление которой выполняется программой. Элементы управления, такие как кнопки и поля редактирования, пользовательского интерфейса также являются окнами, однако в отличие от окна приложения они не существуют только как дочерние элементы окна приложения.

В основе взаимодействия программы с пользователем и операционной системы лежит концепция *сообщений.* С точки зрения приложения сообщение является сигналом о том, что произошло некоторое событие, которое может требовать или не требовать выполнения определенных действий. Сообщение представляет собой структуру данных, содержащую дескриптор окна, которому адресовано сообщение, код сообщения (32-битное целое число) и дополнительную информацию, зависящую от кода сообщения. Каждому коду сообщения сопоставлен уникальный символический идентификатор. Все системные идентификаторы Windows определены при помощи директивы *#define* в файле *winuser.h*. При необходимости разработчик может определить и использовать в приложении собственные коды сообщений. Сообщения, генерируемые приложением, хранятся в очереди сообщений приложения, и обрабатываются оконной процедурой.

*Оконная процедура* – это функция обратного вызова, предназначенная для обработки сообщений, адресованных любому окну класса, в котором содержится ссылка на данную процедуру. В теле функции обычно размещается оператор *switch*, внутри которого и происходит обработка нужных сообщений. Чаще всего приложение обрабатывает оконные сообщения, начинающиеся с префикса *WM\_*, например, *WM\_COMMAND*, *WM\_CREATE*, *WM\_DESTROY* и другие.

*Оконный класс*, или *класс окна* – это структура, определяющая основные характеристики окна. К ним относятся стиль окна и связанные с окном ресурсы, такие как курсор, меню и кисть для закрашивания фона. Кроме того, структура содержит адрес оконной процедуры данного класса. Ссылка на оконный класс передается в функцию *CreateWindow*, вызываемой для создания окна.

Непременным компонентом приложения является главная функция *WinMain*. Обычно она содержит вызовы функций для создания и инициализации окон, цикл обработки сообщений и код, необходимый для завершения работы приложения. В цикле обработки сообщений осуществляется извлечение сообщения из очереди с помощью функции *GetMessage*. Если очередное сообщение имеет код *WM\_QUIT*, то происходит выход из цикла, после чего приложение завершает свою работу. Если очередное сообщение не является сообщением *WM\_QUIT*, то оно передается функции *DispatchMessage*, которая возвращает сообщение обратно в *Windows*. Windows отправляет сообщение для его обработки соответствующей оконной процедуре — иными словами. После возврата из оконной процедуры *Windows* передает управление оператору, который расположен после *DispatchMessage*, и работа цикла продолжается.

1. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

В результате выполнения лабораторной работы было создано приложение для чтения и редактирования текстовых документов с возможностью выделения и копирования текста в буфер обмена (рисунок 1).

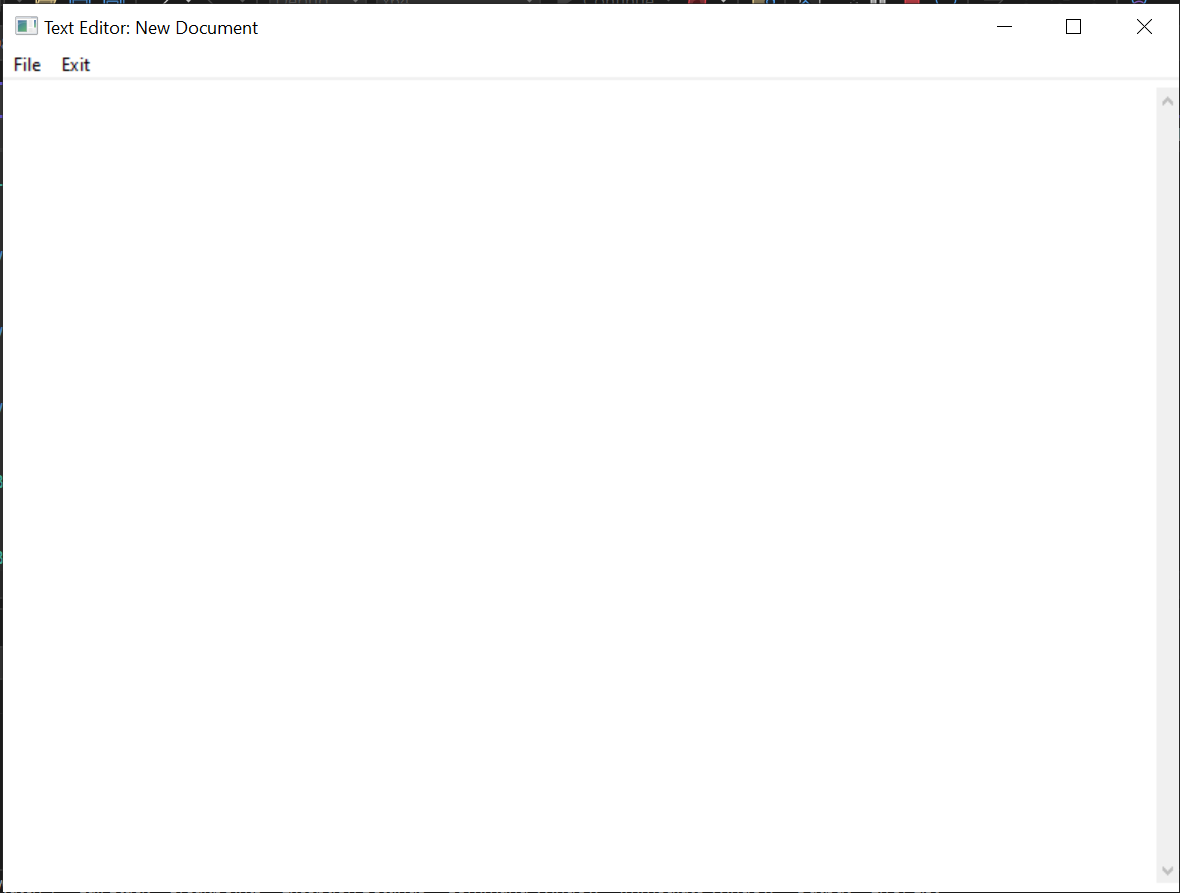


Рисунок 1 – Окно приложения

Меню окна содержит кнопки *«File»* и *«Exit»*. При нажатии *«Exit»* происходит завершение работы программы.

При нажатии на кнопку *«File»* пользователь видит выпадающее меню с кнопками *«Open»* и *«Save»* (рисунок 2).

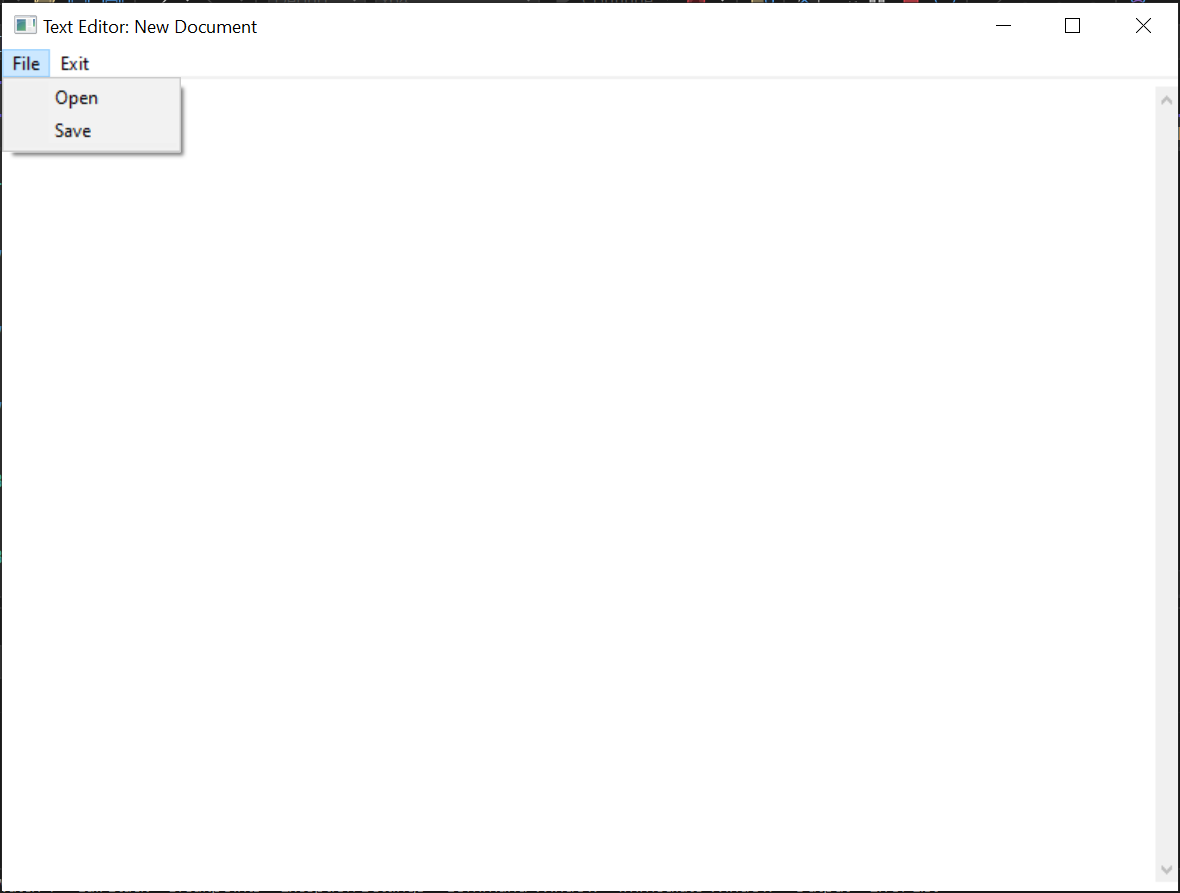


Рисунок 2– Меню *«File»*

*«Open»* дает возможность пользователю выбрать и открыть для чтения и редактирования существующий файл (рисунки 3, 4).

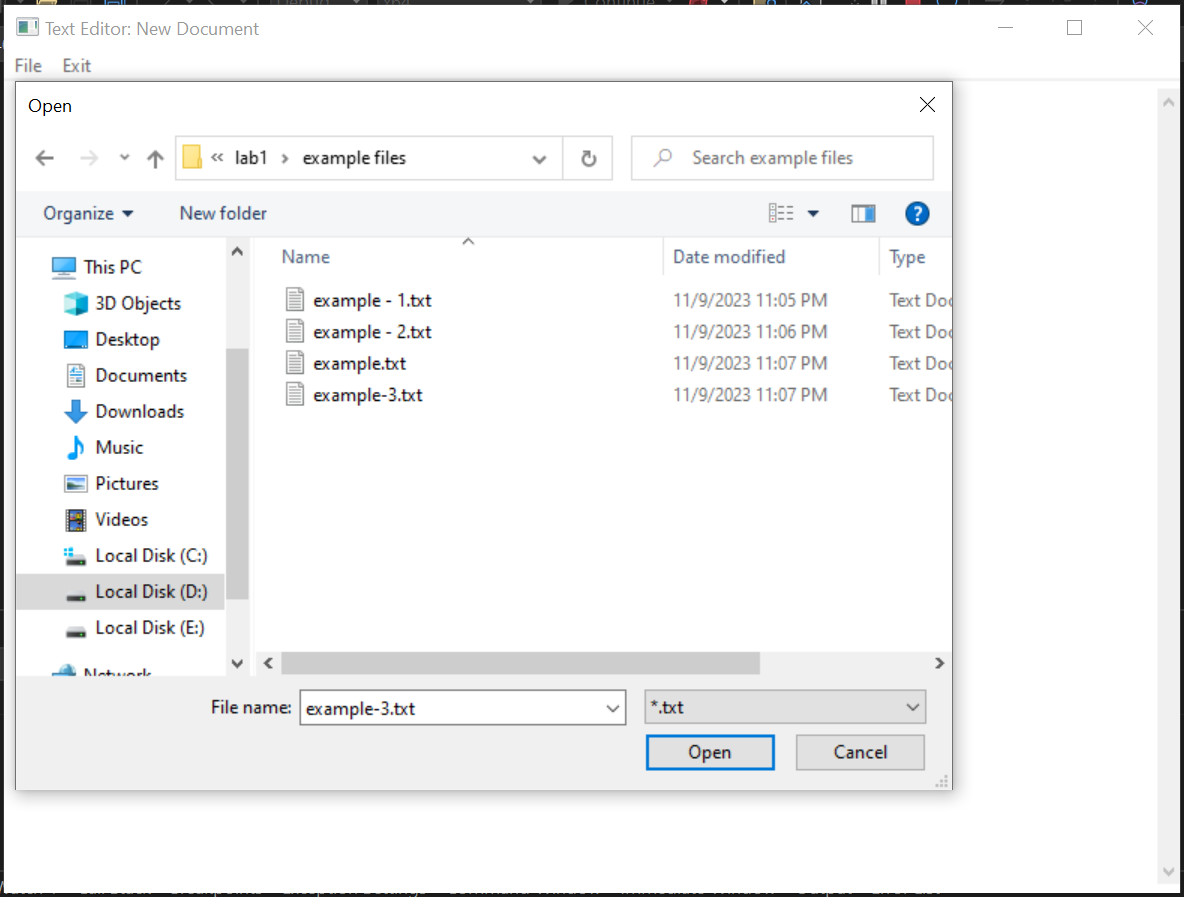


Рисунок 3– Выбор файла

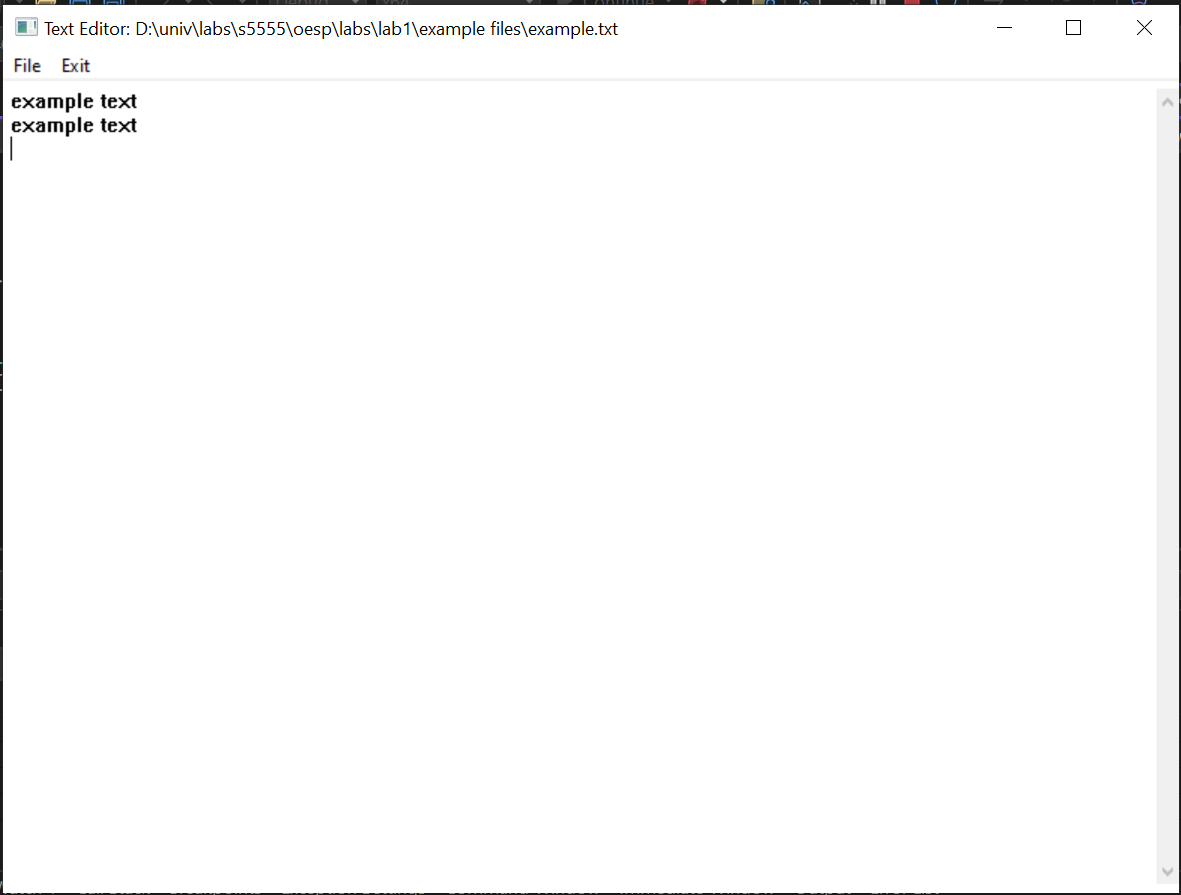
**

Рисунок 4– Чтение и редактирование выбранного файла

*«Save»* сохраняет отредактированный файл (рисунок 5).

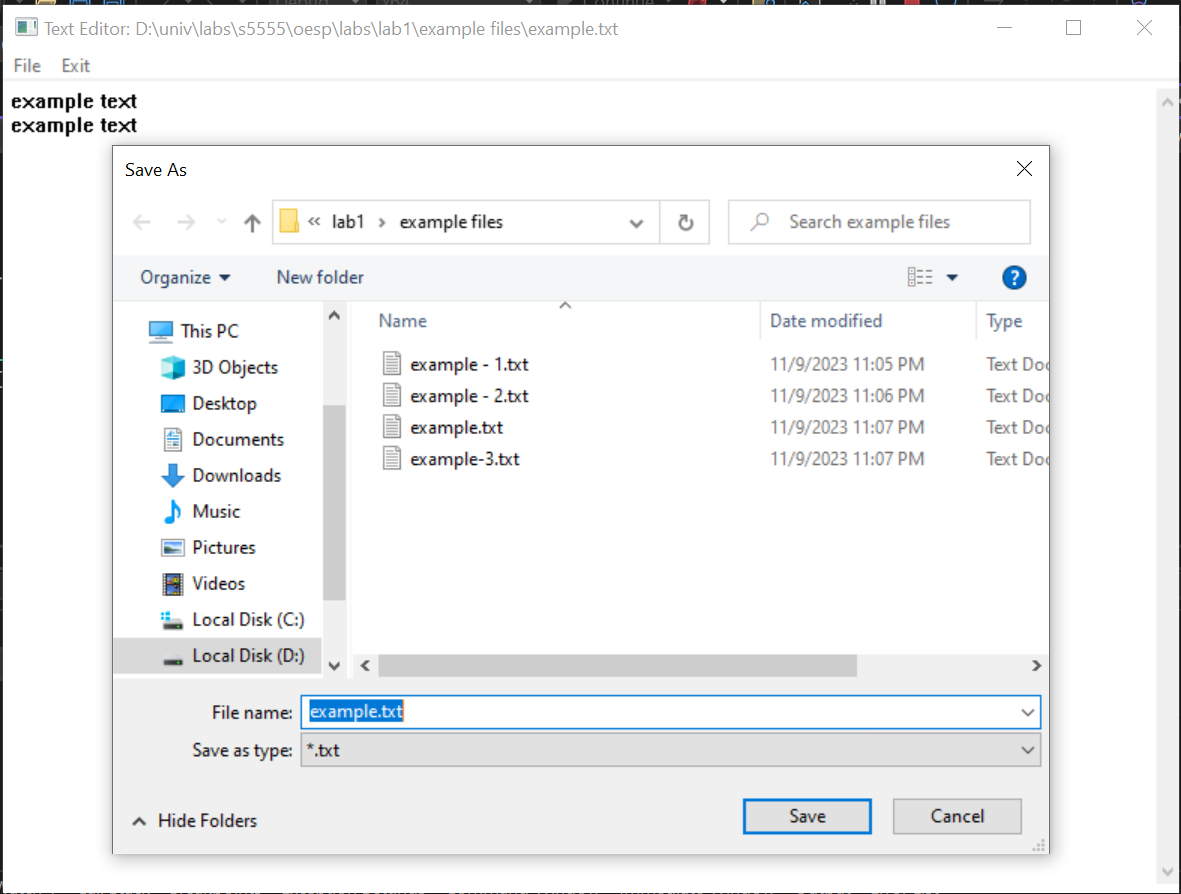


Рисунок 5– Сохранение файла

# ЗАКЛЮЧЕНИЕ

В ходе лабораторной работы были изучены основные принципы программирования в Win32 API: классы окон, регистрация классов, создание оконных процедур, обработка сообщений. В результате проделанной работы было создано оконное приложение для чтения, редактирования и сохранения текстовых документов с возможностью выделения и копирования текста в буфер обмена.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Справочник по программированию для API Win32 [Электронный ресурс]. – Режим доступа : <https://learn.microsoft.com/ru-ru/windows/win32/api/>.
2. Щупак, Ю.А. Win32 API. Разработка приложений для Windows /Ю.А. Щупак. – СПб : Питер, 2008. – 592 с.
3. Getting started with Win32 API [Электронный ресурс]. – Режим доступа : http://www.winprog.org/tutorial/start.html.

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

Листинг 1 – Файл main.cpp

#include <Windows.h>

#define OnMenuOpenClicked 1

#define OnMenuSaveClicked 2

#define OnMenuExitClicked 3

WNDCLASS NewWindowClass(HINSTANCE hInst, LPCWSTR name, WNDPROC procedure);

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

void AddMenus(HWND hWnd);

void AddWidgets(HWND hWnd);

void InitOpenParams(HWND hWnd);

BOOL LoadTextToEdit(HWND hEdit, LPCWSTR pszFileName);

BOOL SaveFileFromEdit(HWND hEdit, LPCTSTR pszFileName);

HWND hEditControl;

OPENFILENAME ofn;

char szFileName[MAX\_PATH];

int WINAPI WinMain(HINSTANCE hInst, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd) {

WNDCLASS MainClass = NewWindowClass(hInst, L"Main Window Class", WndProc);

if (!RegisterClass(&MainClass)) { return -1; }

MSG MainMessage = { 0 };

CreateWindow(L"Main Window Class", L"Text Editor: New Document",

WS\_OVERLAPPEDWINDOW | WS\_VISIBLE, 100, 50, 800, 600, NULL, NULL, hInst, NULL);

while (GetMessage(&MainMessage, NULL, NULL, NULL)) {

TranslateMessage(&MainMessage);

DispatchMessage(&MainMessage);

}

}

WNDCLASS NewWindowClass(HINSTANCE hInst, LPCWSTR name, WNDPROC procedure) {

WNDCLASS windowClass = {};

windowClass.hInstance = hInst;

windowClass.lpszClassName = name;

windowClass.lpfnWndProc = procedure;

return windowClass;

}

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

{

switch (uMsg)

{

case WM\_COMMAND:

switch (wParam)

{

case OnMenuOpenClicked:

if (GetOpenFileName(&ofn)) {

if (!LoadTextToEdit(hEditControl, ofn.lpstrFile)) {

MessageBox(hWnd, L"Failed to open file", L"Error", MB\_OK);

}

else {

WCHAR pszNewTitle[MAX\_PATH + 14] = { 0 };

wcscpy\_s(pszNewTitle, L"Text Editor: ");

SetWindowText(hWnd, lstrcat(pszNewTitle, ofn.lpstrFile));

}

}

break;

case OnMenuSaveClicked:

if (GetSaveFileName(&ofn)) {

if (!SaveFileFromEdit(hEditControl, (LPCTSTR)szFileName)) {

MessageBox(hWnd, L"Failed to save file", L"Error", MB\_OK);

}

else {

WCHAR pszNewTitle[MAX\_PATH + 14] = { 0 };

wcscpy\_s(pszNewTitle, L"Text Editor: ");

SetWindowText(hWnd, lstrcat(pszNewTitle, ofn.lpstrFile));

}

}

break;

case OnMenuExitClicked:

DestroyWindow(hWnd);

break;

default:

break;

}

break;

case WM\_CREATE:

AddMenus(hWnd);

AddWidgets(hWnd);

InitOpenParams(hWnd);

break;

case WM\_CLOSE:

DestroyWindow(hWnd);

break;

case WM\_DESTROY:

PostQuitMessage(0);

return 0;

}

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

}

void AddMenus(HWND hWnd) {

HMENU RootMenu = CreateMenu();

HMENU FileMenu = CreateMenu();

AppendMenu(RootMenu, MF\_POPUP, (UINT\_PTR)FileMenu, L"File");

AppendMenu(FileMenu, MF\_STRING, OnMenuOpenClicked, L"Open");

AppendMenu(FileMenu, MF\_STRING, OnMenuSaveClicked, L"Save");

AppendMenu(RootMenu, MF\_STRING, OnMenuExitClicked, L"Exit");

SetMenu(hWnd, RootMenu);

}

void AddWidgets(HWND hWnd) {

hEditControl = CreateWindowA("edit", NULL,

WS\_VISIBLE | WS\_CHILD | ES\_MULTILINE | WS\_VSCROLL,

5, 5, 780, 530, hWnd, NULL, NULL, NULL);

}

void InitOpenParams(HWND hWnd) {

ZeroMemory(&ofn, sizeof(ofn));

ofn.lStructSize = sizeof(ofn);

ofn.hwndOwner = hWnd;

ofn.lpstrFilter = L"\*.txt\0";

ofn.lpstrFile = (LPWSTR)szFileName;

ofn.nMaxFile = MAX\_PATH;

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

ofn.lpstrDefExt = L"txt";

}

BOOL LoadTextToEdit(HWND hEdit, LPCWSTR pszFileName) {

HANDLE hFile;

BOOL bSuccess = false;

hFile = CreateFile(pszFileName, GENERIC\_READ,

FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, 0, NULL);

if (hFile != INVALID\_HANDLE\_VALUE) {

DWORD dwFileSize = GetFileSize(hFile, NULL);

if (dwFileSize != 0xFFFFFFFF) {

LPSTR pszFileText = (LPSTR)GlobalAlloc(GPTR, dwFileSize + 1);

if (pszFileText != NULL) {

DWORD dwRead;

if (ReadFile(hFile, pszFileText, dwFileSize, &dwRead, NULL)) {

pszFileText[dwFileSize] = { 0 };

if (SetWindowTextA(hEdit, pszFileText)) {

bSuccess = true;

}

}

GlobalFree(pszFileText);

}

}

CloseHandle(hFile);

}

return bSuccess;

}

BOOL SaveFileFromEdit(HWND hEdit, LPCTSTR pszFileName) {

HANDLE hFile;

BOOL bSuccess = false;

hFile = CreateFile(pszFileName, GENERIC\_WRITE, 0,

NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (hFile != INVALID\_HANDLE\_VALUE) {

DWORD dwTextLength = GetWindowTextLengthA(hEdit);

DWORD dwBufferSize = dwTextLength + 1;

LPSTR pszText = (LPSTR)GlobalAlloc(GPTR, dwBufferSize);

if (pszText != NULL) {

if (GetWindowTextA(hEdit, pszText, dwBufferSize) == dwTextLength) {

DWORD dwWritten;

if (WriteFile(hFile, pszText, dwTextLength, &dwWritten, NULL)) {

bSuccess = true;

}

}

GlobalFree(pszText);

}

CloseHandle(hFile);

}

return bSuccess;

}