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 WIN 32 И GDI

ФОРМИРОВАНИЕ СЛОЖНЫХ ИЗОБРАЖЕНИЙ, СОЗДАНИЕ И ИСПОЛЬЗОВАНИЕ ЭЛЕМЕНТОВ

УПРАВЛЕНИЯ, ОБРАБОТКА РАЗЛИЧНЫХ СООБЩЕНИЙ, МЕХАНИЗМ ПЕРЕХВАТА СООБЩЕНИЙ

(WINHOOK)

Выполнил студент гр.153502 Богомолов М.А.

Проверил ассистент кафедры информатики Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Формулировка задачи 3](#_Toc147500218)

[2 Теоретические сведения 4](#_Toc147500219)

[3 Описание функций программы 5](#_Toc147500220)

[3.1 Изменение шрифта и других параметров текста 5](#_Toc147500221)

[3.2 Изменение цвета фона, переключение общей темы оформления 6](#_Toc147500222)

[Список использованных источников 9](#_Toc147500223)

[Приложение А (обязательное) Листинг кода 10](#_Toc147500224)

# **1 ФОРМУЛИРОВКА ЗАДАЧИ**

Целью выполнения лабораторной работы является создание оконного приложения на Win32 API, обладающее минимальным функционалом, позволяющим отработать базовые навыки написания программы на Win32 API, таких как обработка различных сообщений и использование механизма перехвата сообщений (*WinHook*).

В качестве задачи необходимо реализовать возможность смены фона элемента редактирования текста, смены семейства шрифта и его различных параметров, таких как цвет, размер и др.

# **2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

*Rich Edit* (*RichEdit*) – это текстовый компонент в библиотеке *Windows API*, предоставляющий богатую функциональность для редактирования текста. *Rich Edit* позволяет создавать и редактировать текстовые документы с различными форматированиями, включая шрифты, цвета, выравнивание и многое другое. Этот компонент используется в различных текстовых редакторах и программах для создания и редактирования документов.

О *richedit* контроле можно думать, как о функционально-расширенном средстве редактирования. Он обеспечивает множество полезных особенностей, которых нет в простых средствах редактирования, например, возможность использовать множество видов и размеров шрифта, глубокий уровень отмены/восстановления, операцией поиска по тексту, встроенные *OLE*-объекты, поддержка редактирования методом перетаскивания (*drag-and-drop*), и т.д. Так как *richedit* контрол имеет так много особенностей, он сохранен в отдельной *DLL*-библиотеке. Это также означает что, чтобы его использовать, недостаточно просто вызывать *InitCommonControls*, как в других *Common*-контролах. Нужно вызвать *LoadLibrary*, чтобы загрузить *richedit DLL* [1].

В *Windows API* (*WinAPI*), глобальные клавиатурные и мышиные хуки (*WinHooks*) представляют собой механизм перехвата и обработки событий клавиатуры и мыши на уровне операционной системы. Это мощный инструмент, который позволяет программам следить за действиями пользователя на компьютере, перехватывать и модифицировать ввод и действия с клавиатурой и мышью.

С помощью *WinHooks* можно реализовать горячие клавиши для запуска приложений, выполнять команды или события в вашей программе.

Чтобы установить процедуру перехватчика, можно вызвать функцию *SetWindowsHookEx* и указать тип обработчика, вызывающий процедуру, а также указать, должна ли процедура быть связана со всеми потоками на одном рабочем столе с вызывающим потоком или с определенным потоком, а также указатель на точку входа процедуры [2].

# **3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ**

Согласно формулировке задачи, были спроектированы следующие функции программы:

– изменение различных параметров шрифта текстового редактора (размер, стиль, цвет, семейство шрифтов);

– изменение заднего фона элемента *RichEdit*;

– изменение общей темы оформления элемента *RichEdit* с помощью *WinHook*.

## **3.1** **Изменение шрифта и других параметров текста**

Для изменения шрифта и других параметров текста необходимо в меню нажать *Styles*-*Change Text Style* и в открывшемся окне установить необходимые настройки для текста в окне редактирования (Рисунок 1).

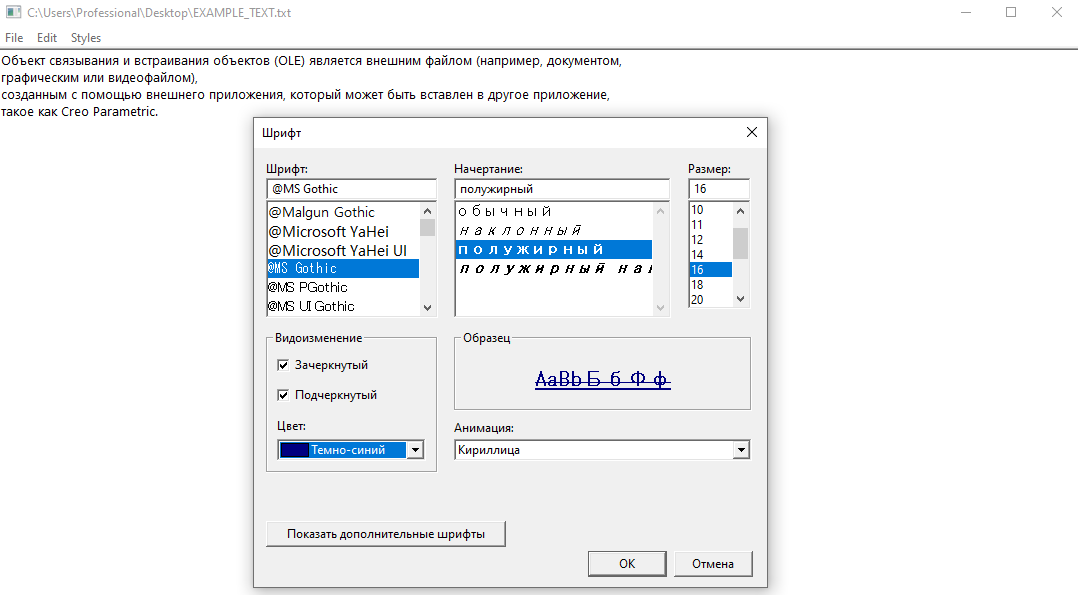


Рисунок 1 – Изменение параметров редактируемого текста

Результат применения различных параметров редактируемого текста (Рисунок 2).



Рисунок 2 – Результат применения параметров

## **3.2** **Изменение цвета фона, переключение общей темы оформления**

Для изменения цвета фона требуется в меню нажать *Styles*-*Change Background Color* и в открывшемся окне выбрать необходимый цвет из предложенных или воспользоваться элементом *ColorPicker* (Рисунок 3).

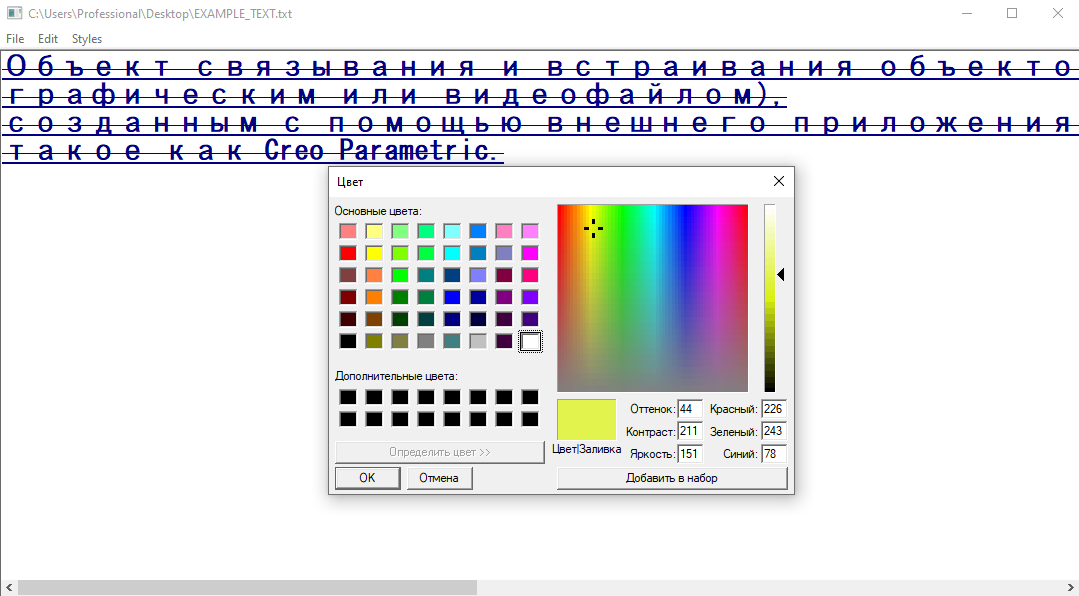


Рисунок 3 – Изменение цвета заднего фона

Результат изменения цвета заднего фона области редактирования (Рисунок 4).

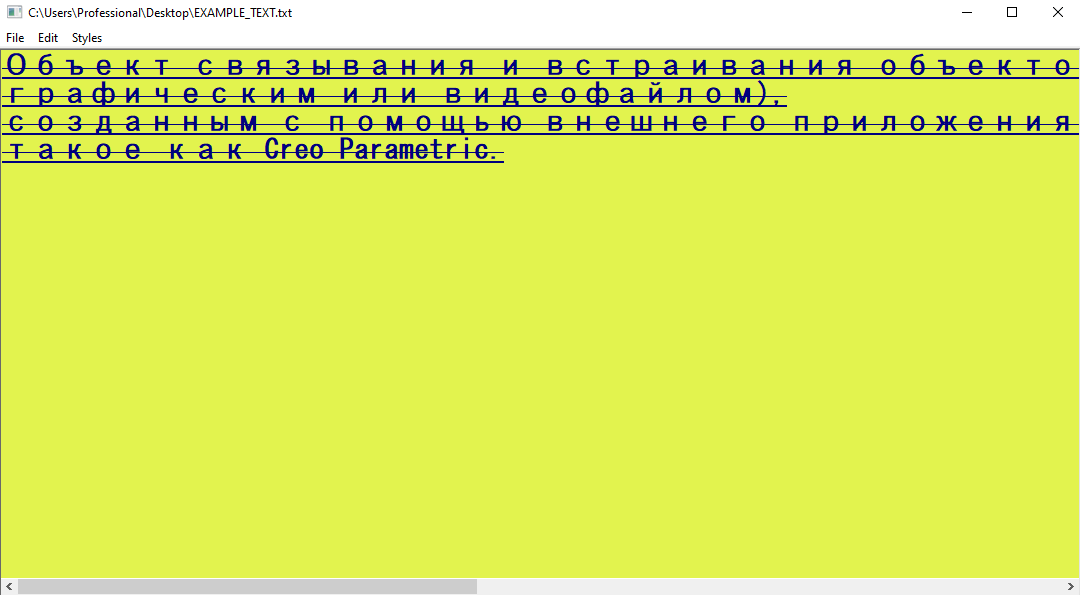


Рисунок 4 – Результат изменения цвета заднего фона

Чтобы переключить общую тему оформления (светлая тема или тёмная тема), необходимо воспользоваться сочетанием клавиш *CTRL* + *B* для использования тёмной темы и *CTRL* + *W* для использования белой темы соответственно. Стоит отметить, что при включении какой-либо из тем оформления среди параметров текста меняется только цвет, т.е. все остальные параметры текста остаются неизменными. Результат переключения общей темы оформления на тёмную тему (Рисунок 5).

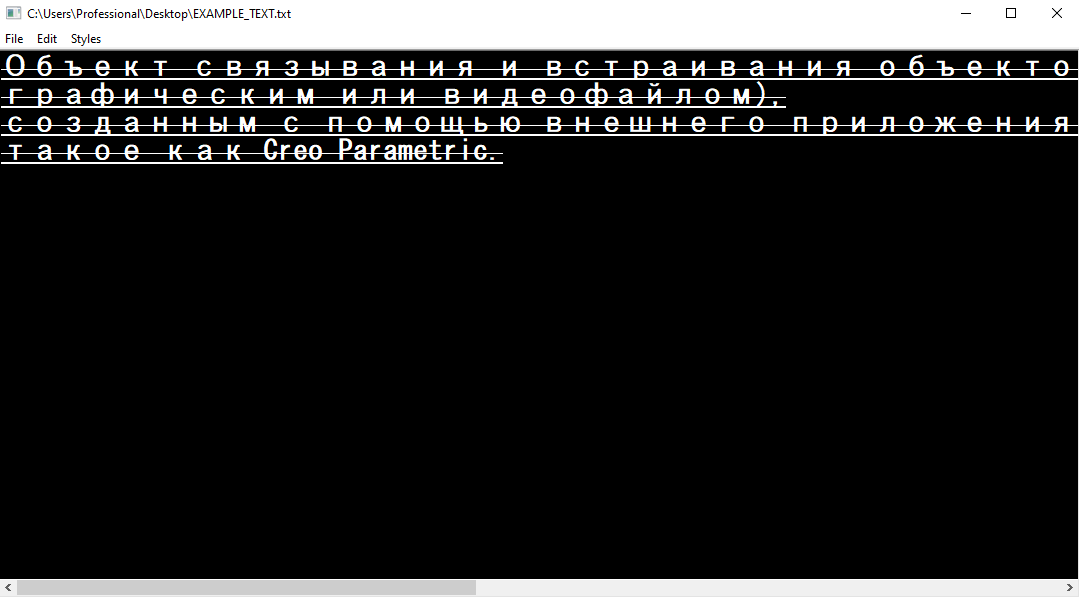


Рисунок 5 – Результат переключения общей темы оформления

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Win32 API. Урок 33. RichEdit Control: основы [Электронный ресурс]. – Режим доступа: <https://wasm.in/blogs/win32-api-urok-33-richedit-control-osnovy.47/> – Дата доступа: 15.10.2023.

[2] Использование перехватчиков [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/winmsg/using-hooks> – Дата доступа: 16.10.2023.

ПРИЛОЖЕНИЕ А  
(обязательное)   
Листинг кода

Листинг 1 – Файл OSISP\_1\_2.cpp

#include "framework.h"

#include "OSISP\_1\_2.h"

#include <windows.h>

#include <commdlg.h>

#include <Richedit.h>

#include "resource.h"

#define MAX\_CONTENT\_SIZE 4096

HINSTANCE hInst;

HWND hwnd;

HWND hwndEdit;

TCHAR currentFileName[MAX\_PATH] = \_T("");

HFONT hFont = NULL;

#define ID\_FILE\_OPEN 1

#define ID\_FILE\_SAVE 2

#define ID\_FILE\_CREATE 3

#define ID\_FILE\_SAVEAS 4

#define ID\_EDIT\_COPY 5

#define ID\_EDIT\_PASTE 6

#define ID\_STYLES\_CHANGE\_BACKGROUND\_COLOR 7

#define ID\_STYLES\_CHANGE\_TEXT\_STYLE 8

COLORREF currentBackgroundColor = RGB(255, 255, 255);

bool isEditingEnabled = false;

HHOOK g\_hHook = NULL;

void OpenFile(HWND hwnd);

void SaveFile(HWND hwnd);

void CreateFile(HWND hwnd);

void SaveFileAs(HWND hwnd);

bool IsTextModified(HWND hwndEdit);

void UpdateWindowTitle(HWND hwnd);

void ChangeTextStyle(HWND hwndEdit);

void ChangeBackgroundColor(HWND hwnd);

void SwitchToLightTheme(HWND hwndEdit);

void SwitchToDarkTheme(HWND hwndEdit);

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

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

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

{

hInst = hInstance;

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

RegisterClassEx(&wc);

hwnd = CreateWindow(\_T("Sample"), \_T("Текстовый редактор"), WS\_OVERLAPPEDWINDOW, CW\_USEDEFAULT, CW\_USEDEFAULT, 800, 600, NULL, NULL, hInstance, NULL);

if (!hwnd) return 0;

HMENU hMenu = CreateMenu();

HMENU hSubMenu = CreatePopupMenu();

AppendMenu(hSubMenu, MF\_STRING, ID\_FILE\_OPEN, \_T("Open"));

AppendMenu(hSubMenu, MF\_STRING | MF\_GRAYED, ID\_FILE\_SAVE, \_T("Save"));

AppendMenu(hSubMenu, MF\_STRING | MF\_GRAYED, ID\_FILE\_SAVEAS, \_T("Save As"));

AppendMenu(hSubMenu, MF\_STRING, ID\_FILE\_CREATE, \_T("Create"));

AppendMenu(hMenu, MF\_POPUP, (UINT\_PTR)hSubMenu, \_T("File"));

HMENU hEditMenu = CreateMenu();

AppendMenu(hEditMenu, MF\_STRING | MF\_GRAYED, ID\_EDIT\_PASTE, \_T("Paste"));

AppendMenu(hEditMenu, MF\_STRING | MF\_GRAYED, ID\_EDIT\_COPY, \_T("Copy"));

AppendMenu(hMenu, MF\_POPUP, (UINT\_PTR)hEditMenu, \_T("Edit"));

HMENU hStylesMenu = CreateMenu();

AppendMenu(hStylesMenu, MF\_STRING | MF\_GRAYED, ID\_STYLES\_CHANGE\_BACKGROUND\_COLOR, \_T("Change Background Color"));

AppendMenu(hStylesMenu, MF\_STRING | MF\_GRAYED, ID\_STYLES\_CHANGE\_TEXT\_STYLE, \_T("Change Text Style"));

AppendMenu(hMenu, MF\_POPUP, (UINT\_PTR)hStylesMenu, \_T("Styles"));

SetMenu(hwnd, hMenu);

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

hwndEdit = CreateWindowEx(WS\_EX\_CLIENTEDGE,MSFTEDIT\_CLASS,NULL,WS\_CHILD | WS\_VISIBLE | WS\_VSCROLL | WS\_HSCROLL | ES\_MULTILINE | ES\_AUTOVSCROLL | ES\_AUTOHSCROLL,0, 0, 0, 0,hwnd,(HMENU)IDC\_TEXT\_EDIT,hInst, NULL);

EnableWindow(hwndEdit, FALSE);

ShowWindow(hwnd, nCmdShow);

UpdateWindow(hwnd);

g\_hHook = SetWindowsHookEx(WH\_KEYBOARD\_LL, KeyboardProc, GetModuleHandle(NULL), 0);

MSG msg;

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

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return (int)msg.wParam;

}

LRESULT CALLBACK KeyboardProc(int nCode, WPARAM wParam, LPARAM lParam) {

if (nCode == HC\_ACTION) {

if (wParam == WM\_KEYDOWN || wParam == WM\_SYSKEYDOWN) {

KBDLLHOOKSTRUCT\* kbdStruct = (KBDLLHOOKSTRUCT\*)lParam;

if (kbdStruct->vkCode == 'B' && GetAsyncKeyState(VK\_CONTROL) < 0) {

SwitchToDarkTheme(hwndEdit);

}

else if (kbdStruct->vkCode == 'W' && GetAsyncKeyState(VK\_CONTROL) < 0) {

SwitchToLightTheme(hwndEdit);

}

}

}

return CallNextHookEx(g\_hHook, nCode, wParam, lParam);

}

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

{

switch (message)

{

case WM\_SIZE:

RECT rcClient;

GetClientRect(hwnd, &rcClient);

MoveWindow(hwndEdit, rcClient.left, rcClient.top, rcClient.right, rcClient.bottom, TRUE);

break;

case WM\_COMMAND:

switch (LOWORD(wParam)) {

case ID\_FILE\_OPEN:

if (IsTextModified(hwndEdit)) {

int result = MessageBox(hwnd, \_T("Сохранить изменения перед открытием нового файла?"), \_T("Подтверждение"), MB\_YESNOCANCEL | MB\_ICONQUESTION);

if (result == IDYES) {

SaveFile(hwnd);

}

else if (result == IDCANCEL) {

return 0;

}

}

OpenFile(hwnd);

SetFocus(hwndEdit);

break;

case ID\_FILE\_SAVE:

SaveFile(hwnd);

break;

case ID\_EDIT\_COPY:

if (OpenClipboard(hwnd)) {

EmptyClipboard();

CloseClipboard();

}

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

break;

case ID\_EDIT\_PASTE:

if (OpenClipboard(hwnd)) {

if (IsClipboardFormatAvailable(CF\_TEXT)) {

HANDLE hClipboardData = GetClipboardData(CF\_TEXT);

if (hClipboardData) {

char\* clipboardText = static\_cast<char\*>(GlobalLock(hClipboardData));

if (clipboardText) {

int textLength = MultiByteToWideChar(CP\_ACP, 0, clipboardText, -1, NULL, 0);

if (textLength > 0) {

wchar\_t\* utf8Text = new wchar\_t[textLength];

MultiByteToWideChar(CP\_ACP, 0, clipboardText, -1, utf8Text, textLength);

int cursorPos = SendMessage(hwndEdit, EM\_GETSEL, 0, 0);

SendMessage(hwndEdit, EM\_REPLACESEL, TRUE, (LPARAM)utf8Text);

delete[] utf8Text;

}

}

GlobalUnlock(hClipboardData);

}

}

CloseClipboard();

}

break;

case ID\_FILE\_SAVEAS:

SaveFileAs(hwnd);

break;

case ID\_FILE\_CREATE:

if (IsTextModified(hwndEdit)) {

int result = MessageBox(hwnd, \_T("Сохранить изменения перед созданием нового файла?"), \_T("Подтверждение"), MB\_YESNOCANCEL | MB\_ICONQUESTION);

if (result == IDYES) {

SaveFile(hwnd);

}

else if (result == IDCANCEL) {

return 0;

}

}

CreateFile(hwnd);

SetFocus(hwndEdit);

break;

case ID\_STYLES\_CHANGE\_TEXT\_STYLE:

if (IsWindowEnabled(hwndEdit)) {

ChangeTextStyle(hwndEdit);

}

break;

case ID\_STYLES\_CHANGE\_BACKGROUND\_COLOR:

ChangeBackgroundColor(hwndEdit);

break;

}

EnableMenuItem(GetMenu(hwnd), ID\_FILE\_SAVE, IsWindowEnabled(hwndEdit) ? MF\_ENABLED : MF\_GRAYED);

EnableMenuItem(GetMenu(hwnd), ID\_FILE\_SAVEAS, IsWindowEnabled(hwndEdit) ? MF\_ENABLED : MF\_GRAYED);

EnableMenuItem(GetMenu(hwnd), ID\_EDIT\_COPY, IsWindowEnabled(hwndEdit) ? MF\_ENABLED : MF\_GRAYED);

EnableMenuItem(GetMenu(hwnd), ID\_EDIT\_PASTE, IsWindowEnabled(hwndEdit) ? MF\_ENABLED : MF\_GRAYED);

EnableMenuItem(GetMenu(hwnd), ID\_STYLES\_CHANGE\_BACKGROUND\_COLOR, IsWindowEnabled(hwndEdit) ? MF\_ENABLED : MF\_GRAYED);

EnableMenuItem(GetMenu(hwnd), ID\_STYLES\_CHANGE\_TEXT\_STYLE, IsWindowEnabled(hwndEdit) ? MF\_ENABLED : MF\_GRAYED);

DrawMenuBar(hwnd);

break;

case WM\_CLOSE:

if (IsTextModified(hwndEdit)) {

int result = MessageBox(hwnd, \_T("Сохранить изменения перед закрытием?"), \_T("Сохранение"), MB\_YESNOCANCEL | MB\_ICONQUESTION);

if (result == IDYES) {

SaveFile(hwnd);

}

else if (result == IDCANCEL) {

return 0;

}

}

DestroyWindow(hwnd);

break;

case WM\_DROPFILES:

return 0;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

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

}

return 0;

}

void OpenFile(HWND hwnd)

{

OPENFILENAME ofn;

TCHAR szFileName[MAX\_PATH] = \_T("");

ZeroMemory(&ofn, sizeof(OPENFILENAME));

ofn.lStructSize = sizeof(OPENFILENAME);

ofn.hwndOwner = hwnd;

ofn.lpstrFilter = \_T("Text Files (\*.txt)\0\*.txt\0");

ofn.lpstrFile = szFileName;

ofn.nMaxFile = MAX\_PATH;

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

if (GetOpenFileName(&ofn)) {

HANDLE hFile = CreateFile(szFileName, GENERIC\_READ, 0, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

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

DWORD dwFileSize = GetFileSize(hFile, NULL);

if (dwFileSize != INVALID\_FILE\_SIZE) {

char\* buffer = new char[dwFileSize + 1];

DWORD dwRead;

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

buffer[dwRead] = '\0';

SendMessage(hwndEdit, WM\_SETTEXT, 0, (LPARAM)\_T(""));

SetWindowTextA(hwndEdit, buffer);

}

delete[] buffer;

}

CloseHandle(hFile);

lstrcpy(currentFileName, szFileName);

EnableWindow(hwndEdit, TRUE);

isEditingEnabled = true;

UpdateWindowTitle(hwnd);

}

}

}

void SaveFile(HWND hwnd)

{

if (lstrlen(currentFileName) == 0) {

OPENFILENAME ofn;

TCHAR szFileName[MAX\_PATH] = \_T("");

ZeroMemory(&ofn, sizeof(OPENFILENAME));

ofn.lStructSize = sizeof(OPENFILENAME);

ofn.hwndOwner = hwnd;

ofn.lpstrFilter = \_T("Text Files (\*.txt)\0\*.txt\0");

ofn.lpstrFile = szFileName;

ofn.nMaxFile = MAX\_PATH;

ofn.Flags = OFN\_OVERWRITEPROMPT;

ofn.lpstrDefExt = \_T("txt");

if (!GetSaveFileName(&ofn)) {

return;

}

lstrcpy(currentFileName, szFileName);

UpdateWindowTitle(hwnd);

}

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

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

int textLength = GetWindowTextLength(hwndEdit);

if (textLength > 0) {

char\* buffer = new char[textLength + 1];

GetWindowTextA(hwndEdit, buffer, textLength + 1);

DWORD dwWritten;

WriteFile(hFile, buffer, textLength, &dwWritten, NULL);

delete[] buffer;

}

CloseHandle(hFile);

}

}

void CreateFile(HWND hwnd)

{

OPENFILENAME ofn;

TCHAR szFileName[MAX\_PATH] = \_T("");

ZeroMemory(&ofn, sizeof(OPENFILENAME));

ofn.lStructSize = sizeof(OPENFILENAME);

ofn.hwndOwner = hwnd;

ofn.lpstrFilter = \_T("Text Files (\*.txt)\0\*.txt\0");

ofn.lpstrFile = szFileName;

ofn.nMaxFile = MAX\_PATH;

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

ofn.lpstrDefExt = \_T("txt");

ofn.lpstrTitle = \_T("Создание нового файла");

if (GetSaveFileName(&ofn)) {

lstrcpy(currentFileName, szFileName);

UpdateWindowTitle(hwnd);

SetWindowText(hwndEdit, \_T(""));

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

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

CloseHandle(hFile);

EnableWindow(hwndEdit, TRUE);

isEditingEnabled = true;

}

}

}

void SaveFileAs(HWND hwnd) {

OPENFILENAME ofn;

TCHAR szFileName[MAX\_PATH] = \_T("");

ZeroMemory(&ofn, sizeof(OPENFILENAME));

ofn.lStructSize = sizeof(OPENFILENAME);

ofn.hwndOwner = hwnd;

ofn.lpstrFilter = \_T("Text Files (\*.txt)\0\*.txt\0");

ofn.lpstrFile = szFileName;

ofn.nMaxFile = MAX\_PATH;

ofn.Flags = OFN\_OVERWRITEPROMPT;

ofn.lpstrDefExt = \_T("txt");

if (GetSaveFileName(&ofn)) {

lstrcpy(currentFileName, szFileName);

UpdateWindowTitle(hwnd);

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

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

int textLength = GetWindowTextLength(hwndEdit);

if (textLength > 0) {

char\* buffer = new char[textLength + 1];

GetWindowTextA(hwndEdit, buffer, textLength + 1);

DWORD dwWritten;

WriteFile(hFile, buffer, textLength, &dwWritten, NULL);

delete[] buffer;

}

CloseHandle(hFile);

}

}

}

bool IsTextModified(HWND hwndEdit) {

return IsWindowEnabled(hwndEdit);

}

void UpdateWindowTitle(HWND hwnd) {

if (lstrlen(currentFileName) > 0) {

SetWindowText(hwnd, currentFileName);

}

else {

SetWindowText(hwnd, \_T("Текстовый редактор"));

}

}

void ChangeTextStyle(HWND hwndEdit) {

LOGFONT lfont;

CHOOSEFONT cFont;

ZeroMemory(&cFont, sizeof(CHOOSEFONT));

cFont.lStructSize = sizeof(CHOOSEFONT);

cFont.hwndOwner = hwndEdit;

cFont.lpLogFont = &lfont;

cFont.Flags = CF\_SCREENFONTS | CF\_EFFECTS | CF\_SELECTSCRIPT;

cFont.nFontType = RUSSIAN\_CHARSET;

if (ChooseFont(&cFont)) {

HFONT hfont = CreateFontIndirect(cFont.lpLogFont);

CHARFORMAT cf;

memset(&cf, 0, sizeof(CHARFORMAT));

cf.cbSize = sizeof(CHARFORMAT);

cf.dwMask = CFM\_COLOR | CFM\_FACE | CFM\_SIZE | CFM\_BOLD | CFM\_ITALIC | CFM\_UNDERLINE | CFM\_STRIKEOUT;

cf.crTextColor = cFont.rgbColors;

wcscpy\_s(cf.szFaceName, LF\_FACESIZE, cFont.lpLogFont->lfFaceName);

cf.yHeight = cFont.lpLogFont->lfHeight \* 20;

cf.dwEffects = 0;

if (cFont.lpLogFont->lfWeight == FW\_BOLD) {

cf.dwEffects |= CFE\_BOLD;

}

if (cFont.lpLogFont->lfItalic) {

cf.dwEffects |= CFE\_ITALIC;

}

if (cFont.lpLogFont->lfUnderline) {

cf.dwEffects |= CFE\_UNDERLINE;

}

if (cFont.lpLogFont->lfStrikeOut) {

cf.dwEffects |= CFE\_STRIKEOUT;

}

SendMessage(hwndEdit, EM\_SETCHARFORMAT, SCF\_ALL, (LPARAM)&cf);

}

}

void ChangeBackgroundColor(HWND hwnd)

{

CHOOSECOLOR cc;

COLORREF acrCustClr[16] = { NULL };

ZeroMemory(&cc, sizeof(cc));

cc.lStructSize = sizeof(cc);

cc.hwndOwner = hwnd;

cc.lpCustColors = (LPDWORD)acrCustClr;

cc.rgbResult = currentBackgroundColor;

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

if (ChooseColor(&cc))

{

currentBackgroundColor = cc.rgbResult;

SendMessage(hwndEdit, EM\_SETBKGNDCOLOR, 0, currentBackgroundColor);

}

InvalidateRect(hwnd, NULL, TRUE);

}

void SwitchToLightTheme(HWND hwndEdit) {

CHARFORMAT cf;

memset(&cf, 0, sizeof(CHARFORMAT));

cf.cbSize = sizeof(CHARFORMAT);

cf.dwMask = CFM\_COLOR;

cf.crTextColor = RGB(0, 0, 0);

currentBackgroundColor = RGB(0, 0, 0);

SendMessage(hwndEdit, EM\_SETCHARFORMAT, SCF\_ALL, (LPARAM)&cf);

SendMessage(hwndEdit, EM\_SETBKGNDCOLOR, 0, RGB(255, 255, 255));

}

void SwitchToDarkTheme(HWND hwndEdit) {

CHARFORMAT cf;

memset(&cf, 0, sizeof(CHARFORMAT));

cf.cbSize = sizeof(CHARFORMAT);

cf.dwMask = CFM\_COLOR;

cf.crTextColor = RGB(255, 255, 255);

currentBackgroundColor = RGB(255, 255, 255);

SendMessage(hwndEdit, EM\_SETCHARFORMAT, SCF\_ALL, (LPARAM)&cf);

SendMessage(hwndEdit, EM\_SETBKGNDCOLOR, 0, RGB(0, 0, 0));

}