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 Толстой Д.В.

Проверил ассистент кафедры информатики Гриценко Н.Ю.

Минск 2023

СОДЕРЖАНИЕ

[1 Формулировка задачи 3](#_Toc147247188)

[2 Теоритические сведения 4](#_Toc147247189)

[3 Описание функций программы 5](#_Toc147247190)

[3.1 Изменение шрифта 5](#_Toc147247191)

[3.2 Изменение темы текстового редактора 5](#_Toc147247192)

[Список использованных источников 7](#_Toc147247193)

[Приложение а 8](#_Toc147247194)

# **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 оконных сообщений.

В качестве задачи необходимо создать текстовый редактор с поддержкой множества открытых документов. Реализовать функциональность сохранения и загрузки файлов. Также требуется реализовать возмонжность смены тема текстового редактора, смены шрифта и его различных параметров.

# **2 ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ**

Цвета в WinAPI представлены с использованием структуры COLORREF, которая представляет собой 32-битное значение, состоящее из красной, зеленой и синей компонент (RGB).

Компоненты цвета (красный, зеленый, синий) могут быть представлены в диапазоне от 0 до 255, где 0 означает полное отсутствие цвета, а 255 - максимальную интенсивность цвета.

Мы можем использовать функцию RGB(r, g, b) для создания цветовой структуры, например: COLORREF textColor = RGB(255, 0, 0); для создания красного цвета.

Для работы с текстом и шрифтами в WinAPI используется структура LOGFONT. Она содержит информацию о шрифте, такую как размер, имя шрифта, стиль и другие параметры.

Чтобы создать и настроить шрифт, заполните экземпляр структуры LOGFONT и используйте функцию CreateFontIndirect(&logFont), чтобы получить дескриптор шрифта. Управление элементами интерфейса:

Для управления цветами и шрифтами элементов интерфейса, таких как кнопки, текстовые поля и т. д., используются сообщения и функции управления окнами, такие как WM\_CTLCOLORBTN, WM\_CTLCOLOREDIT, WM\_CTLCOLORSTATIC, SetTextColor(), SetBkColor() и другие.

# **3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ**

Согласно формулировке задачи, были спроектированы следующие функции программы:

– Изменение параметров шрифта текстового редактора;

– Изменение темы текстового редактора на темную и обратно.

## **3.1 Изменение шрифта**

Для изменения шрифта, выберите в меню View, далее Change Font. Выберите шрифт, размер текста и его стиль (Рисунок 1).

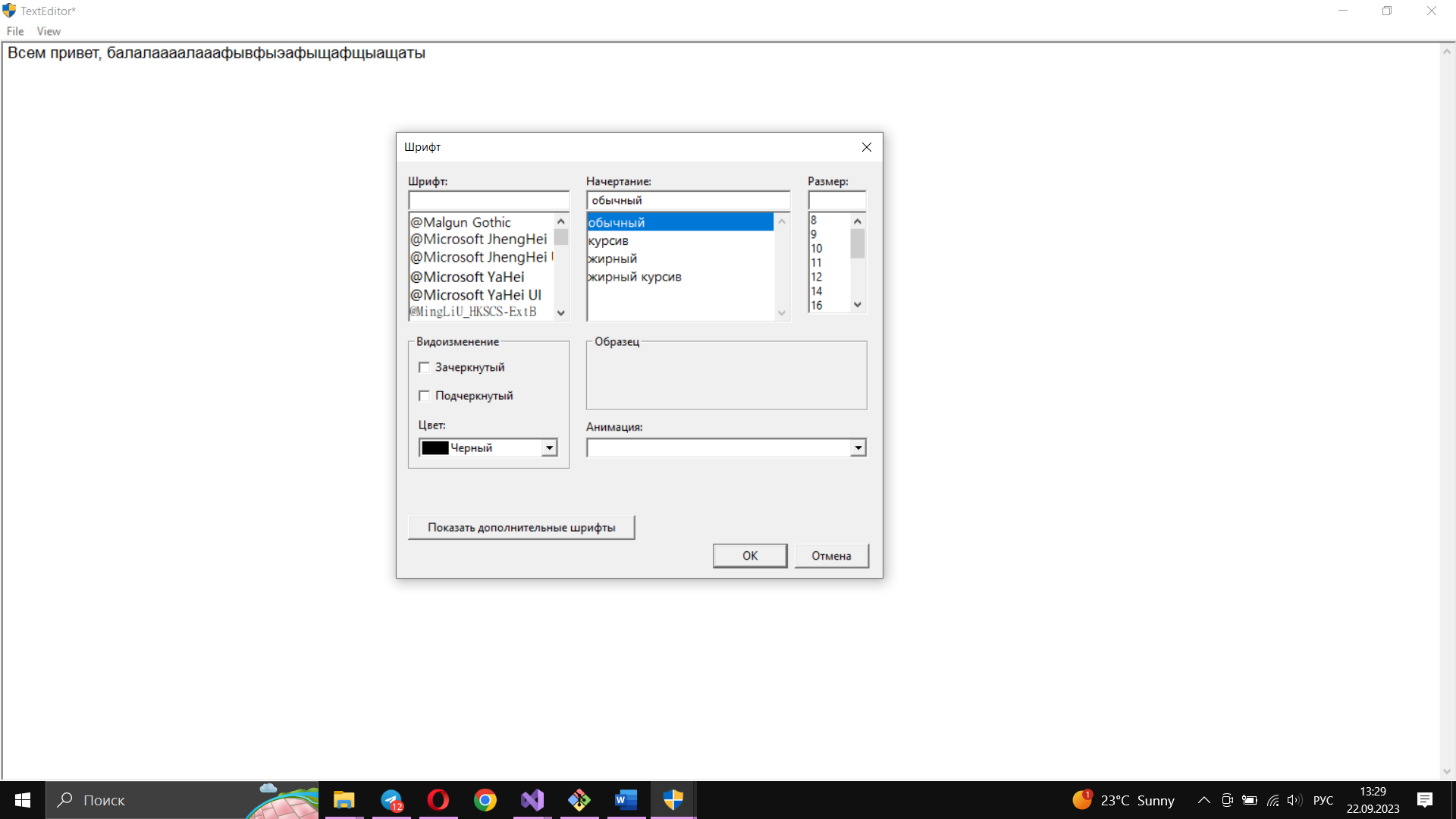


Рисунок 1 – Изменение шрифта и стиля текста

## **3.2 Изменение темы текстового редактора**

Для изменения темы, выберите в меню View, далее Change Theme. В зависимости от текущей темы, будет выбрана противоположная(Рисунок 2-3).

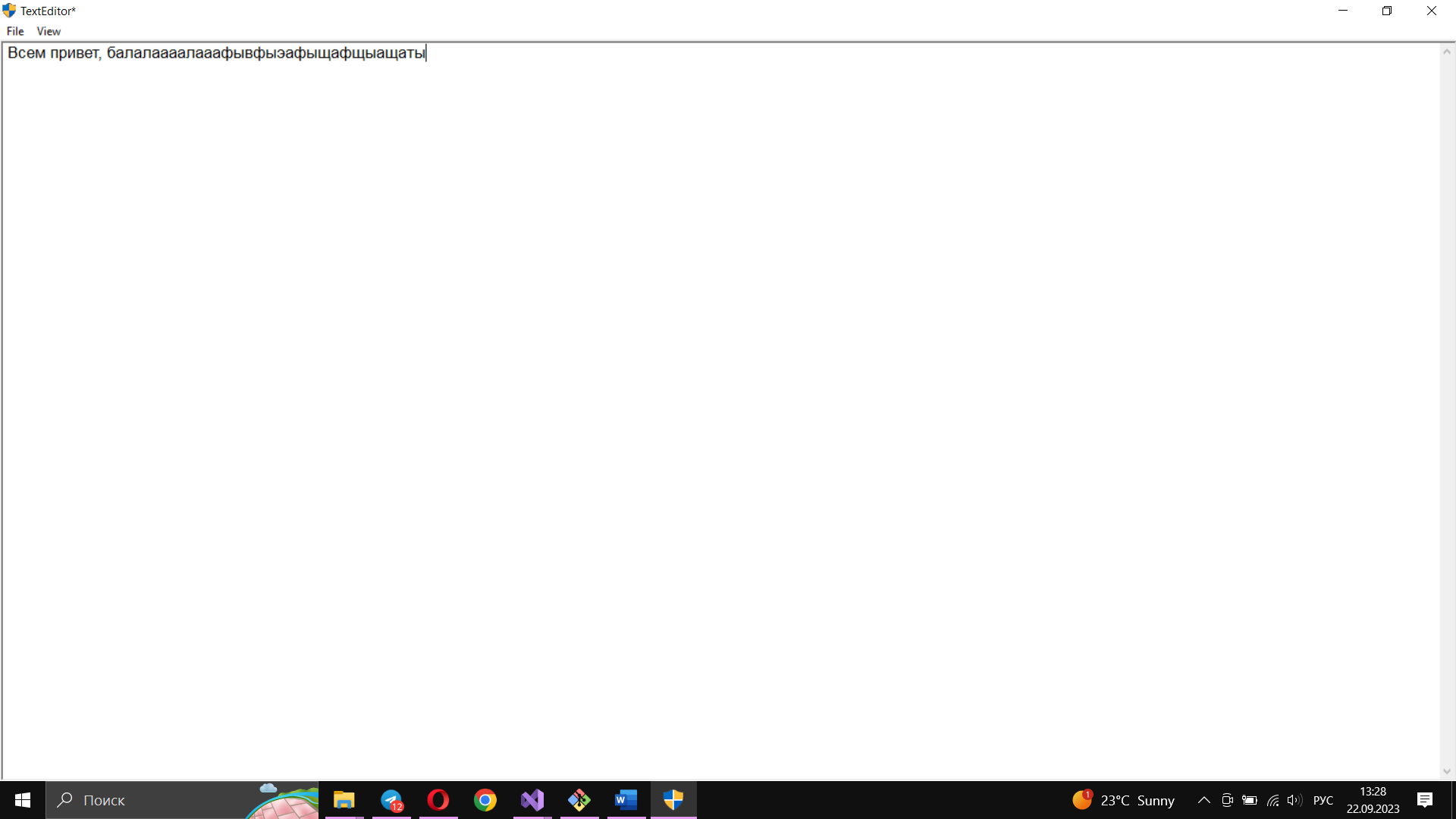


Рисунок 2 – светлая тема

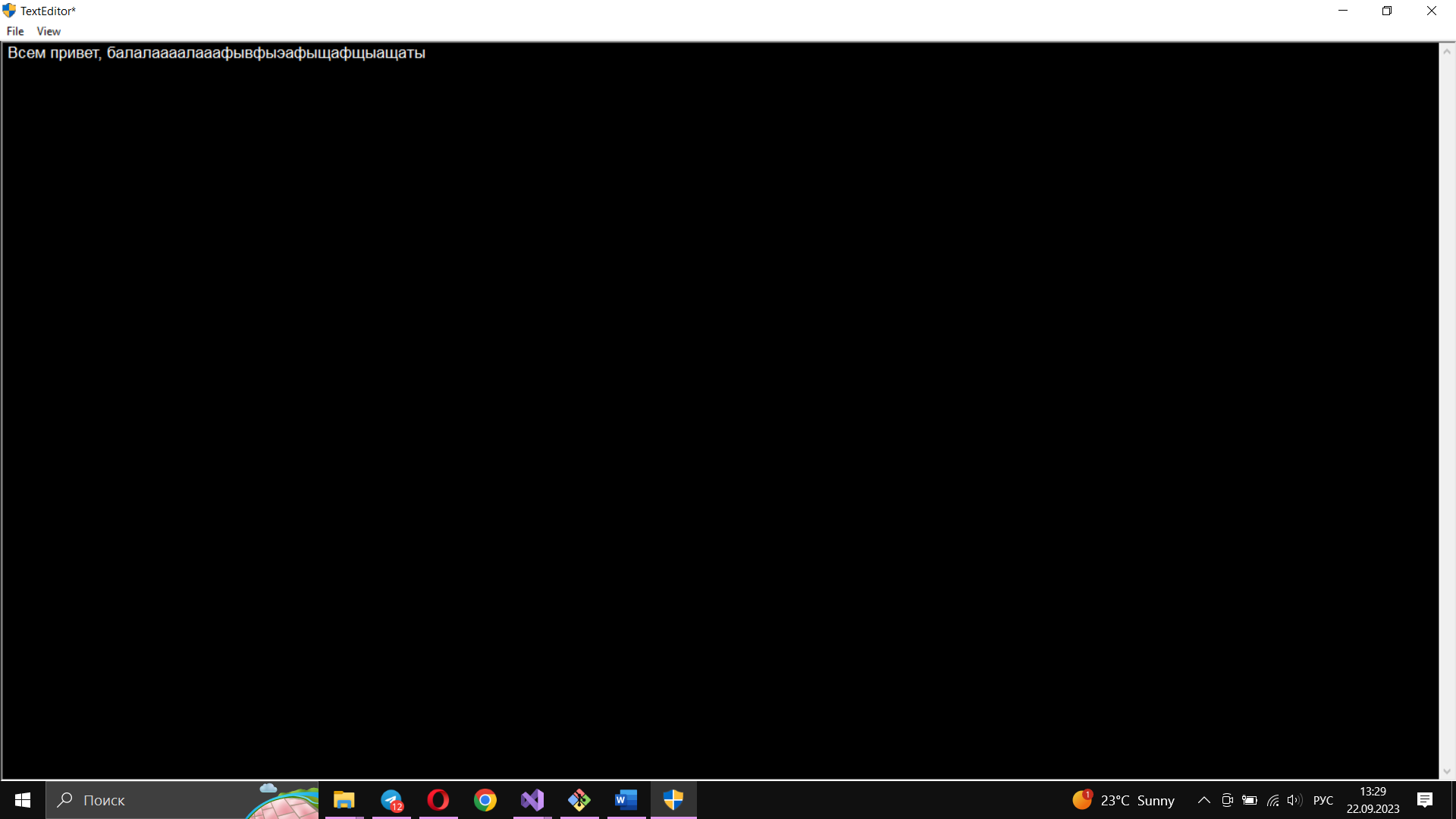


Рисунок 3 – темная тема

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Build desktop Windows apps using the Win32 API [Электронный ресурс]. – Электронные данные. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32

[2] C++ WINDOWS API – Электронные данные. – Режим доступа: https://www.youtube.com/@egordeveloper3794

# **ПРИЛОЖЕНИЕ А**

Листинг кода

Файл SowtwareDefinitions.h

#pragma once

#include <string>

#include <windows.h>

#define OnMenuClicked 1

#define OnOpenFileClicked 2

#define OnSaveFileClicked 3

#define OnSaveAsFileClicked 4

#define OnCloseEditorClicked 5

#define IDC\_EDIT 6

#define HOTKEY\_CTRL\_S 7

#define HOTKEY\_CTRL\_O 8

#define OnChangeFontClicked 9

#define OnChangeThemeClicked 10

#define OnCopyTextClicked 11

#define OnCutTextClicked 12

#define OnInsertTextClicked 13

HWND hEditControl;

std::wstring currFilePath;

std::wstring THEME(L"Light");

bool SAVED = true;

HFONT g\_hFont; // Дескриптор шрифта

COLORREF bgColor;

COLORREF txtColor;

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam);

int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, PWSTR pCmdLine, int nCmdShow);

void MainWindAddMenus(HWND hwnd);

void MainWindAddWidgets(HWND hwnd);

std::wstring GetFilenameFromPath(std::wstring path);

void SaveTextToFile(const HWND& hwnd);

bool AskForExit();

void DeleteStarFromName(const HWND& hwnd);

void ChangeFont();

void ChangeTheme();

Файл TextEditor.cpp

#ifndef UNICODE

#define UNICODE

#endif

#include <windows.h>

#include <wchar.h>

#include <fstream>

#include "SoftwareDefinitions.h"

#include <string>

#include <atlstr.h>

#include <vector>

#include <iostream>

int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, PWSTR pCmdLine, int nCmdShow)

{

bgColor = RGB(255, 255, 255);

txtColor = RGB(0, 0, 0);

// Register the window class.

const wchar\_t CLASS\_NAME[] = L"Sample Window Class";

WNDCLASS wc = { };

wc.hIcon = LoadIcon(NULL, IDI\_SHIELD);

wc.lpfnWndProc = WindowProc;

wc.hInstance = hInstance;

wc.lpszClassName = CLASS\_NAME;

wc.hbrBackground = CreateSolidBrush(bgColor);

RegisterClass(&wc);

// Create the window

HWND hwnd = CreateWindowExW(0, CLASS\_NAME, L"TextEditor", WS\_OVERLAPPEDWINDOW, CW\_USEDEFAULT, CW\_USEDEFAULT, CW\_USEDEFAULT, CW\_USEDEFAULT, NULL, NULL, hInstance, NULL);

if (hwnd == NULL)

{

return 0;

}

ShowWindow(hwnd, nCmdShow);

g\_hFont = CreateFont(20, 0, 0, 0, FW\_NORMAL, FALSE, FALSE, FALSE, DEFAULT\_CHARSET, OUT\_DEFAULT\_PRECIS, CLIP\_DEFAULT\_PRECIS, DEFAULT\_QUALITY, DEFAULT\_PITCH | FF\_DONTCARE, L"Arial");

SendMessage(hEditControl, WM\_SETFONT, (WPARAM)g\_hFont, TRUE);

// Run the message loop.

MSG msg = { };

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

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

if (g\_hFont)

{

DeleteObject(g\_hFont);

}

return static\_cast<int>(msg.wParam);

}

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)

{

wchar\_t msg[32];

switch (uMsg)

{

case WM\_COMMAND:

{

int wmId = LOWORD(wParam);

if (wmId == IDC\_EDIT)

{

if (HIWORD(wParam) == EN\_CHANGE)

{

if (SAVED)

{

SAVED = false;

wchar\_t oldName[100];

GetWindowTextW(hwnd, oldName, 100);

std::wstring newName(oldName);

SetWindowTextW(hwnd, newName.append(L"\*").c\_str());

}

}

}

switch (wParam)

{

case OnCopyTextClicked:

{

// Отправляем сообщение WM\_COPY для копирования выделенного текста в буфер обмена

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

break;

}

case OnCutTextClicked:

{

// Отправляем сообщение WM\_CUT для вырезания выделенного текста в буфер обмена

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

SAVED = false;

break;

}

case OnInsertTextClicked:

{

// Отправляем сообщение WM\_PASTE для вставки текста из буфера обмена

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

SAVED = false;

break;

}

case OnChangeFontClicked:

{

ChangeFont();

break;

}

case OnChangeThemeClicked:

{

ChangeTheme();

InvalidateRect(hwnd, nullptr, TRUE);

break;

}

case OnOpenFileClicked:

{

wchar\_t buff[100];

OPENFILENAME ofn = { 0 };

ofn.lStructSize = sizeof(ofn);

ofn.hwndOwner = hwnd;

ofn.lpstrFile = buff;

ofn.nMaxFile = sizeof(buff);

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

ofn.nFilterIndex = 1;

ofn.lpstrDefExt = L"txt";

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

if (GetOpenFileName(&ofn))

{

wchar\_t textBuffer[1024];

currFilePath = buff;

auto name = GetFilenameFromPath(currFilePath);

auto newNameOfWnd = std::wstring(L"Text Editor - ").append(name);

std::wifstream outFile((LPWSTR)currFilePath.c\_str());

if (outFile.is\_open()) {

outFile >> textBuffer;

outFile.close();

SetWindowTextW(hEditControl, textBuffer);

SetWindowTextW(hwnd, newNameOfWnd.c\_str());

SAVED = true;

}

else {

MessageBox(hwnd, L"Failed to open file!", L"Error", MB\_OK | MB\_ICONERROR);

}

}

break;

}

case OnSaveAsFileClicked:

{

SaveTextToFile(hwnd);

SAVED = true;

DeleteStarFromName(hwnd);

break;

}

case OnSaveFileClicked:

{

if (currFilePath.empty())

{

SaveTextToFile(hwnd);

}

else

{

wchar\_t textBuffer[1024];

GetWindowTextW(hEditControl, textBuffer, sizeof(textBuffer) / sizeof(textBuffer[0]));

std::wofstream outFile(currFilePath);

if (outFile.is\_open()) {

outFile << textBuffer;

outFile.close();

MessageBox(hwnd, L"File saved successfully!", L"Success", MB\_OK | MB\_ICONINFORMATION);

SAVED = true;

DeleteStarFromName(hwnd);

}

else {

MessageBox(hwnd, L"Failed to save file!", L"Error", MB\_OK | MB\_ICONERROR);

}

}

break;

}

case OnCloseEditorClicked:

{

if (AskForExit())

{

return 0;

}

break;

}

}

break;

}

case WM\_CTLCOLOREDIT:

{

HDC hdcEdit = (HDC)wParam;

SetBkColor(hdcEdit, bgColor);

SetTextColor(hdcEdit, txtColor);

return (LRESULT)CreateSolidBrush(bgColor);

break;

}

case WM\_SIZE:

{

int newWidth = LOWORD(lParam);

int newHeight = HIWORD(lParam);

// Установите новый размер окна

SetWindowPos(hEditControl, NULL, 1, 0, newWidth, newHeight, SWP\_NOMOVE | SWP\_NOZORDER);

break;

}

case WM\_CREATE:

{

MainWindAddMenus(hwnd);

MainWindAddWidgets(hwnd);

break;

}

case WM\_PAINT:

{

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hwnd, &ps);

HBRUSH hBrush = CreateSolidBrush(bgColor);

FillRect(hdc, &ps.rcPaint, hBrush);

DeleteObject(hBrush);

EndPaint(hwnd, &ps);

break;

}

case WM\_CLOSE:

{

if (AskForExit())

{

return 0;

}

break;

}

case WM\_HOTKEY:

{

int id = wParam;

// Проверяем, была ли нажата клавиша Ctrl

switch (id)

{

// Проверяем, была ли нажата клавиша "S"

case HOTKEY\_CTRL\_S:

{

// Обработка нажатия Ctrl + S

if (currFilePath.empty())

{

SaveTextToFile(hwnd);

}

else

{

wchar\_t textBuffer[1024];

GetWindowTextW(hEditControl, textBuffer, sizeof(textBuffer) / sizeof(textBuffer[0]));

std::wofstream outFile(currFilePath);

if (outFile.is\_open()) {

outFile << textBuffer;

outFile.close();

MessageBox(hwnd, L"File saved successfully!", L"Success", MB\_OK | MB\_ICONINFORMATION);

SAVED = true;

DeleteStarFromName(hwnd);

}

else {

MessageBox(hwnd, L"Failed to save file!", L"Error", MB\_OK | MB\_ICONERROR);

}

}

break;

}

case HOTKEY\_CTRL\_O:

{

// Обработка нажатия Ctrl + O

wchar\_t buff[100];

OPENFILENAME ofn = { 0 };

ofn.lStructSize = sizeof(ofn);

ofn.hwndOwner = hwnd;

ofn.lpstrFile = buff;

ofn.nMaxFile = sizeof(buff);

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

ofn.nFilterIndex = 1;

ofn.lpstrDefExt = L"txt";

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

if (GetOpenFileName(&ofn))

{

wchar\_t textBuffer[1024];

currFilePath = buff;

auto name = GetFilenameFromPath(currFilePath);

auto newNameOfWnd = std::wstring(L"Text Editor - ").append(name);

std::wifstream outFile((LPWSTR)currFilePath.c\_str());

if (outFile.is\_open()) {

outFile >> textBuffer;

outFile.close();

SetWindowTextW(hEditControl, textBuffer);

SetWindowTextW(hwnd, newNameOfWnd.c\_str());

SAVED = true;

}

else {

MessageBox(hwnd, L"Failed to open file!", L"Error", MB\_OK | MB\_ICONERROR);

}

}

break;

}

}

break;

}

default:

{

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

}

}

}

void MainWindAddMenus(HWND hwnd)

{

HMENU RootMenu = CreateMenu();

HMENU FileMenu = CreateMenu();

HMENU ViewMenu = CreateMenu();

HMENU SubViewMenu = CreateMenu();

HMENU SubMenu = CreateMenu();

AppendMenuW(SubMenu, MF\_STRING, OnOpenFileClicked, L"Open");

AppendMenuW(SubMenu, MF\_STRING, OnSaveAsFileClicked, L"Save As");

AppendMenuW(SubMenu, MF\_STRING, OnSaveFileClicked, L"Save");

AppendMenuW(SubMenu, MF\_SEPARATOR, NULL, NULL);

AppendMenuW(SubMenu, MF\_STRING, OnCopyTextClicked, L"Copy Text");

AppendMenuW(SubMenu, MF\_STRING, OnCutTextClicked, L"Cut Text");

AppendMenuW(SubMenu, MF\_STRING, OnInsertTextClicked, L"Insert Text");

AppendMenuW(SubMenu, MF\_SEPARATOR, NULL, NULL);

AppendMenuW(SubMenu, MF\_STRING, OnCloseEditorClicked, L"Close Editor");

AppendMenuW(SubViewMenu, MF\_STRING, OnChangeFontClicked, L"Change font");

AppendMenuW(SubViewMenu, MF\_STRING, OnChangeThemeClicked, L"Change theme");

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

AppendMenu(ViewMenu, MF\_POPUP, (UINT\_PTR)SubViewMenu, L"View");

AppendMenuW(RootMenu, MF\_POPUP, (UINT\_PTR)SubMenu, L"File");

AppendMenuW(RootMenu, MF\_POPUP, (UINT\_PTR)SubViewMenu, L"View");

RegisterHotKey(hwnd, HOTKEY\_CTRL\_S, MOD\_CONTROL, 'S');

RegisterHotKey(hwnd, HOTKEY\_CTRL\_O, MOD\_CONTROL, 'O');

SetMenu(hwnd, RootMenu);

}

void MainWindAddWidgets(HWND hwnd)

{

// Получите размер и позицию окна в экранных координатах

RECT windowRect;

GetWindowRect(hwnd, &windowRect);

int windowWidth = windowRect.right - windowRect.left;

int windowHeight = windowRect.bottom - windowRect.top;

// windowWidth и windowHeight содержат ширину и высоту окна

// windowRect.left и windowRect.top содержат координаты верхнего левого угла

hEditControl = CreateWindowEx(WS\_EX\_CLIENTEDGE, L"Edit", L"", WS\_CHILD | WS\_VISIBLE | ES\_LEFT |

WS\_VSCROLL | ES\_MULTILINE | ES\_AUTOVSCROLL,

1, 0, windowWidth - 20, windowHeight - 20, hwnd, (HMENU)IDC\_EDIT, NULL, NULL);

}

std::wstring GetFilenameFromPath(std::wstring path)

{

// Найти последний символ разделителя пути

size\_t found = path.find\_last\_of(L"\\/");

if (found != std::wstring::npos) {

// Извлечь имя файла после последнего разделителя пути

std::wstring fileName = path.substr(found + 1);

// вернуть имя файла

return fileName;

}

else {

// Если разделитель не найден, это может быть имя файла без пути

// Вывести всю строку как имя файла

return path;

}

}

void SaveTextToFile(const HWND& hwnd)

{

currFilePath.clear();

wchar\_t buff[100];

OPENFILENAME ofn = { 0 };

ofn.lStructSize = sizeof(ofn);

ofn.hwndOwner = hwnd;

ofn.lpstrFile = buff;

ofn.nMaxFile = sizeof(buff);

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

ofn.nFilterIndex = 1;

ofn.lpstrDefExt = L"txt";

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

if (GetSaveFileName(&ofn))

{

wchar\_t textBuffer[1024];

currFilePath = buff;

auto name = GetFilenameFromPath(currFilePath);

auto newNameOfWnd = std::wstring(L"Text Editor - ").append(name);

GetWindowTextW(hEditControl, textBuffer, sizeof(textBuffer) / sizeof(textBuffer[0]));

std::wofstream outFile(buff);

if (outFile.is\_open()) {

outFile << textBuffer;

outFile.close();

MessageBox(hwnd, L"File saved successfully!", L"Success", MB\_OK | MB\_ICONINFORMATION);

SetWindowTextW(hwnd, newNameOfWnd.c\_str());

SAVED = true;

DeleteStarFromName(hwnd);

}

else {

MessageBox(hwnd, L"Failed to save file!", L"Error", MB\_OK | MB\_ICONERROR);

}

}

}

bool AskForExit()

{

if (!SAVED)

{

int msgboxID = MessageBox(

NULL,

L"Do you really want to exit? You have unsaved changes",

L"Exit",

MB\_ICONEXCLAMATION | MB\_YESNO);

if (msgboxID == IDYES)

{

PostQuitMessage(0);

return true;

}

return false;

}

else

{

PostQuitMessage(0);

return true;

}

}

void DeleteStarFromName(const HWND& hwnd)

{

wchar\_t oldName[100];

GetWindowTextW(hwnd, oldName, 100);

std::wstring newName(oldName);

// Поиск и удаление символа "\*"

size\_t pos = newName.find(L"\*");

while (pos != std::wstring::npos) {

newName.erase(pos, 1); // Удаление 1 символа, начиная с позиции pos

pos = newName.find(L"\*", pos); // Поиск следующего символа "\*"

}

SetWindowTextW(hwnd, newName.c\_str());

}

void ChangeFont()

{

CHOOSEFONT cf = { sizeof(CHOOSEFONT) };

LOGFONT lf = { 0 };

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

cf.lpLogFont = &lf;

if (ChooseFont(&cf))

{

if (g\_hFont)

{

DeleteObject(g\_hFont);

}

g\_hFont = CreateFontIndirect(&lf);

SendMessage(hEditControl, WM\_SETFONT, (WPARAM)g\_hFont, TRUE);

}

}

void ChangeTheme()

{

if (THEME == L"Light")

{

bgColor = RGB(0, 0, 0);

txtColor = RGB(255, 255, 255);

THEME = L"Dark";

}

else

{

bgColor = RGB(255, 255, 255);

txtColor = RGB(0, 0, 0);

THEME = L"Light";

}

}