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 Win32 и GDI. Формирование сложных изображений, создание и использование элементов управления, обработка различных сообщений, механизм перехвата сообщений (winhook).»

Выполнил:

студент гр. 153504

Скроба А.П.

Проверил:

Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_Toc146675523)

[2 Краткие теоретические сведения 4](#_Toc146675524)

[3 Результат выполнения программы 5](#_Toc146675525)

[Список использованных источников 9](#_Toc146675526)

[Приложение А 10](#_Toc146675527)

# 1 ЦЕЛЬ РАБОТЫ

Разработка текстового редактора с использованием Win32 API. Основное задание заключается в создании приложения, которое позволит пользователям читать и редактировать текстовые документы. Редактор должен обладать функциональностью настройки внешнего оформления, включая стили текста и цвета фона.

**2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Интерфейс графического устройства (GDI) является частью операционной системы Windows. Он предоставляет приложениям средства отправки графической информации на такие устройства, как видеодисплей и принтер.

При использовании GDI графическое устройство представляется как контекст устройства (DC). Все вызовы рисования выполняются через объект контекста устройства, который инкапсулирует API-интерфейсы Windows для рисования линий, фигур и текста. Контексты устройств позволяют рисовать в Windows независимо от устройства. Объекты GDI, такие как растровые изображения, кисти, палитры и перья, выбираются в контексте устройства, прежде чем их можно будет использовать для отображения графической информации. Шрифты также можно выбирать в контексте устройства для управления рендерингом текста.

Основой работы с графикой и текстом является цветовая палитра. Цветовая палитра — это массив, содержащий значения цветов, определяющие цвета, которые в настоящее время могут отображаться или рисоваться на устройстве вывода. Цветовые палитры используются устройствами, которые способны генерировать много цветов, но могут отображать или рисовать их подмножество в любой момент времени. Для таких устройств система поддерживает системную палитру для отслеживания текущих цветов устройства и управления ими. Приложения не имеют прямого доступа к системной палитре. Вместо этого система связывает палитру по умолчанию с каждым контекстом устройства. Приложения могут использовать цвета в палитре по умолчанию или определять собственные цвета, создавая логические палитры и связывая их с контекстами отдельных устройств.

Приложения могут использовать цвета в палитре по умолчанию или определять собственные цвета, создавая логические палитры и связывая их с контекстами отдельных устройств.

# 3 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ ПРОГРАММЫ

Результатом выполнения лабораторной работы является улучшенная версия программы предыдущей лабораторной. В меню добавилось четыре кнопки для редактирования текста и фона: «Bold», «Italic», «Text color» и «Bg color» (рисунок 3.1).

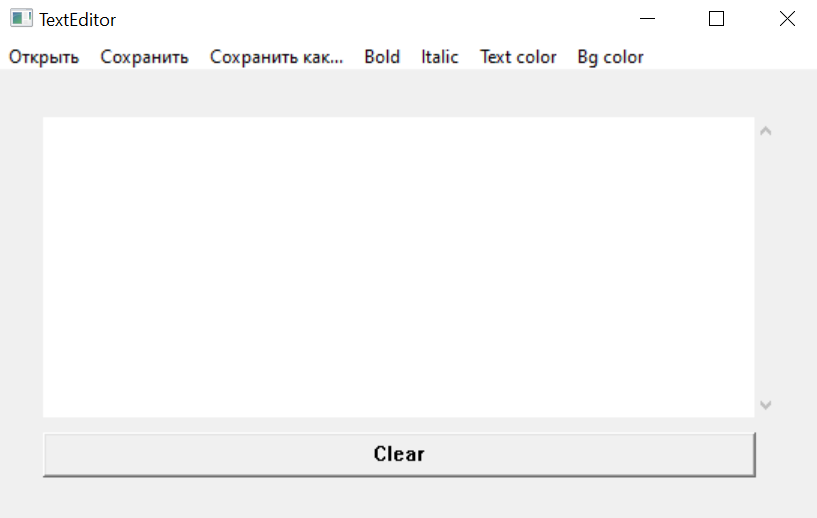
****

Рисунок 3.1 – Основной интерфейс приложения

При нажатии на кнопку «Bold» ко всему тексту в редакторе будет применено полужирное начертание. Соответственно при повторном нажатии на кнопку, оно будет отменено. (рисунок 3.2, рисунок 3.3).

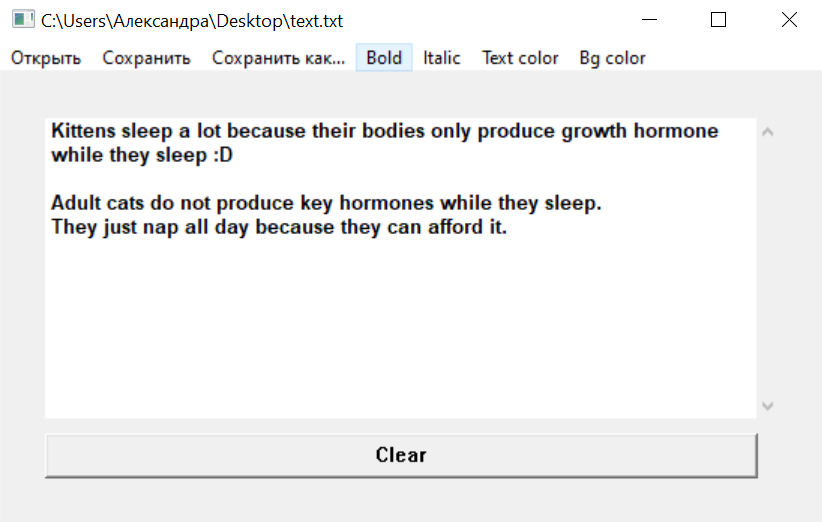


Рисунок 3.2 – Применения полужирного начертания к тексту

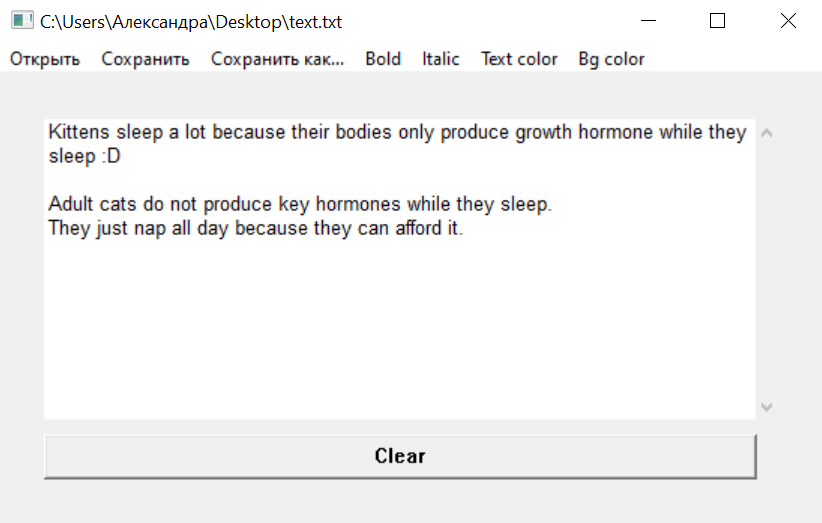


Рисунок 3.3 – Возвращение тексту его первоначального стиля

Так же при нажатии на кнопку «Italic» весь текст в редакторе применит курсивное начертание, а при повторном нажатии на кнопку текст примет свой первоначальный стиль. В программе реализована возможность объединять стили текста (рисунок 3.4).

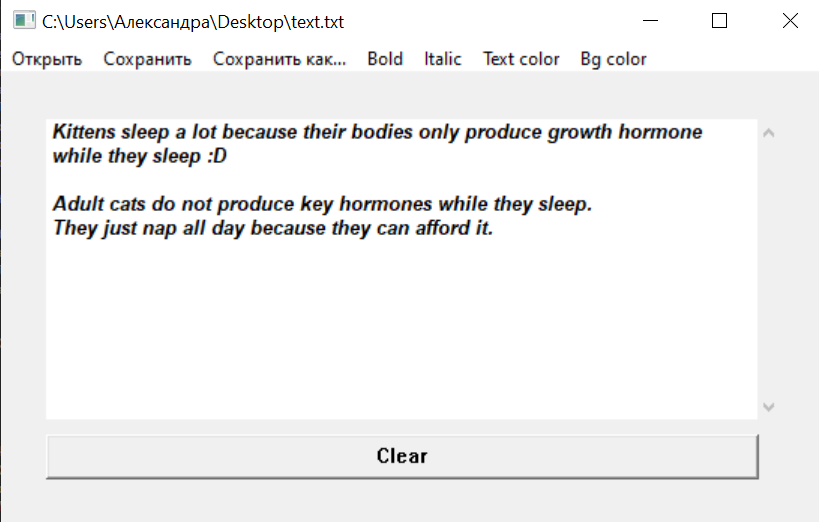
****

Рисунок 3.4 – Применение полужирного и курсивного начертания к тексту

Кнопка «Text color» позволяет выбирать цвет текста редактора. После нажатия на данную кнопку откроется диалоговая панель, с помощью которой пользователь может выбрать цвет из основного набора цветов или дополнительного (рисунок 3.5).

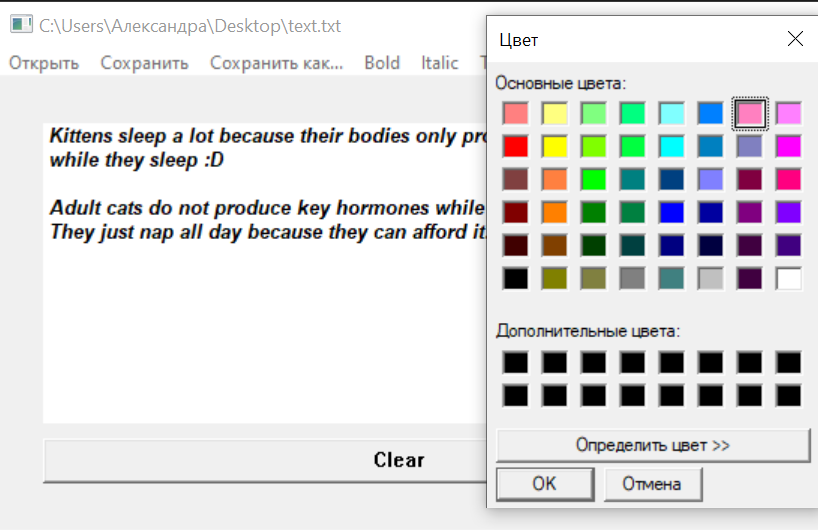


Рисунок 3.5 – Диалоговое окно с цветовой палитрой

Если нажать на кнопку «Определить цвет» внешний вид диалоговой панели изменится и пользователь сможет добавить новый цвет в набор дополнительных цветов и затем выбрать его из этого набора (рисунок 3.6).

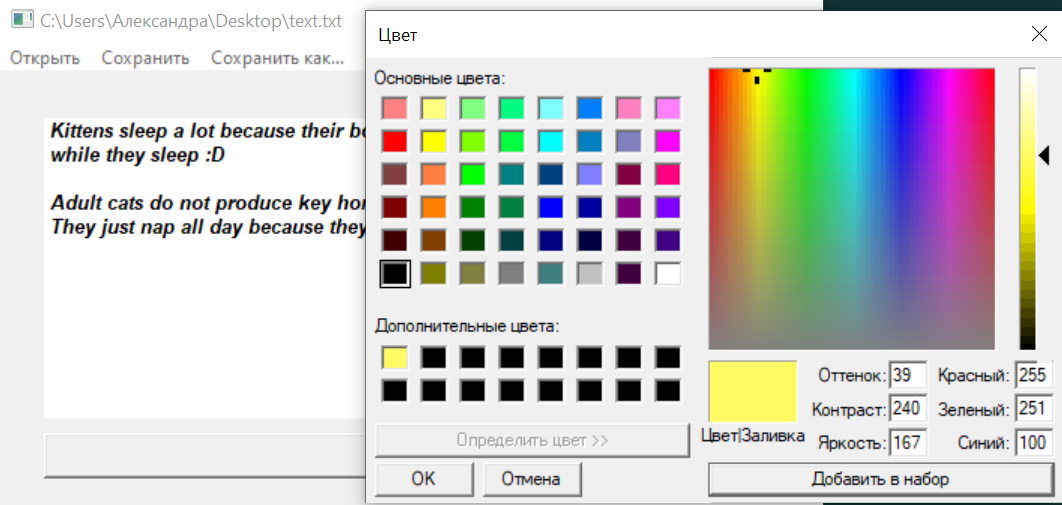


Рисунок 3.6 – Добавление пользовательского цвета в палитру

После выбора цвета и нажатия кнопки «ОК» цвет текста в редакторе изменится на выбранный пользователем (рисунок 3.7).

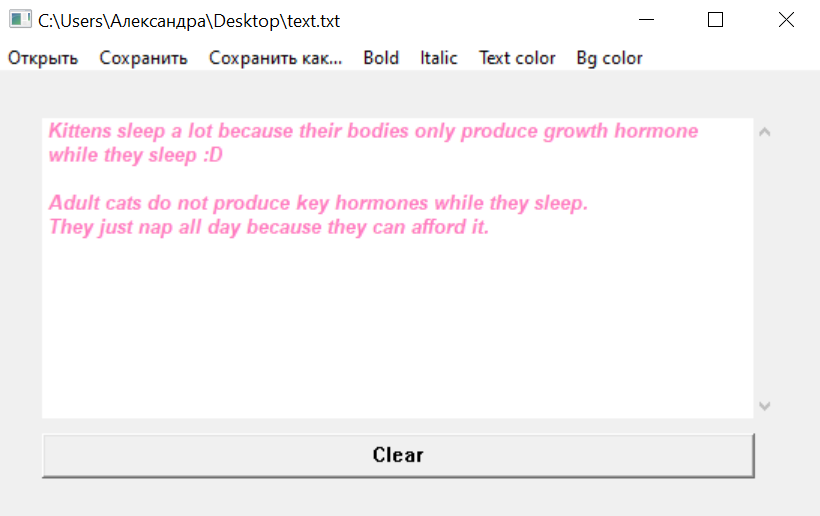


Рисунок 3.7 – Изменение цвета текста в редакторе

При нажатии на кнопку «Bg color», так же как и при «Text color» откроется диалоговая панель с палитрой цветов и возможностью добавления пользовательских. После нажатия кнопки «ОК» фон редактора изменит свой цвет на выбранный пользователем (рисунок 3.8).

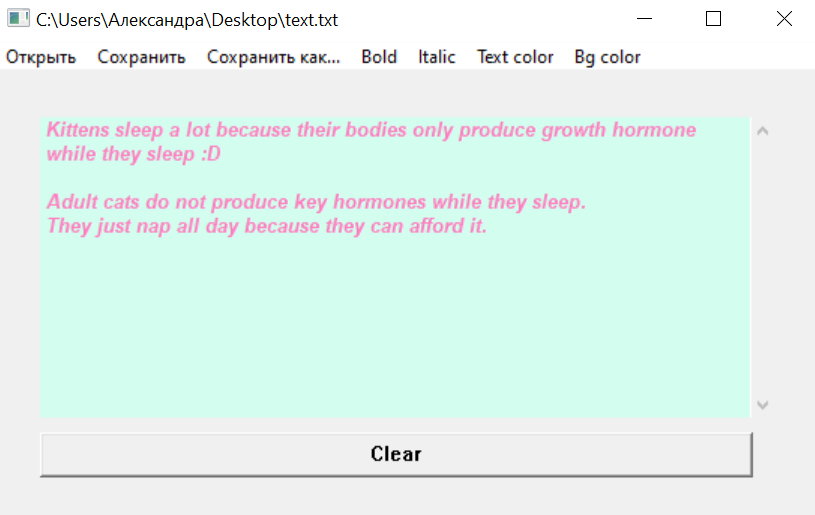
****

Рисунок 3.8 – Возможность выделения и копирования текста

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Марапулец Ю. В. Системное программирование в WIN API. Учебное пособие, 2021. – 256 с

[2] Графический интерфейс GDI в Microsoft Windows [Электронный ресурс]. – Режим доступа: https://www.frolov-lib.ru/books/bsp/v14/ch3\_2.htm.

[3] Интерфейс графического устройства (GDI) [Электронный ресурс]. – Режим доступа: https://documentation.help/Win32/GDI.htm.

[4] Системная палитра [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/gdi/system-palette>.

[4] Использование перехватчиков [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/winmsg/using-hooks.

# ПРИЛОЖЕНИЕ А

**(обязательное)**

**Исходный код программы**

Листинг 1 – Файл Resource.h:

#pragma once

#define OnOpenMenuClicked 1

#define OnSaveAsMenuClicked 2

#define OnButtonClicked 3

#define OnSaveMenuClicked 4

#define OnBoldClicked 5

#define OnItalicClicked 6

#define OnTextColorClicked 7

#define OnBgColorClicked 8

#define TextBufferSize 256

char Buffer[TextBufferSize];

char currentFileName[260];

const DWORD ItalicFont = 0x00000002; // Битовый флаг для стиля шрифта курсив

const DWORD UnderlineFont = 0x00000004; // Битовый флаг для стиля шрифта подчеркивание

CHOOSECOLOR cc; // Структура для диалога выбора цвета

COLORREF customColors[16]; // Массив для пользовательских цветов

HBRUSH hBrush;

HDC dc;

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

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

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

WNDCLASS WindowClass(HBRUSH BGColor, HCURSOR Cursor, HINSTANCE hInst, LPCWSTR Name, WNDPROC wc);

HFONT CreateMyFont(int fontSize, bool isBold, bool isItalic, bool isUnderline);

void MainWndAddMenu(HWND hwnd);

void MainWndAddWidget(HWND hwnd);

void SetOpenFileParams(HWND hwnd);

void SaveData(LPCSTR path);

void LoadData(LPCSTR path);

HWND hEditConstrol;

HFONT hFont;

char fileName[260];

OPENFILENAMEA ofn = { 0 };

bool isTextChanged = false;

int length = 0;

int loadFileLength;

bool isBold = false;

bool isItalic = false;

Листинг 2 **–** Файл Source.cpp:

#include <Windows.h>

#include <string.h>

#include <shlwapi.h>

#pragma comment(lib, "shlwapi.lib")

#include "Resource.h"

#include <commdlg.h>

int WINAPI WinMain(\_In\_ HINSTANCE hInstance, \_In\_opt\_ HINSTANCE hPrevInstance, \_In\_ LPSTR lpCmdLine, \_In\_ int nShowCmd) {

WNDCLASS MainClass = WindowClass((HBRUSH)COLOR\_WINDOW, LoadCursor(NULL, IDC\_ARROW), hInstance, L"MainWndClass", WndProc);

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

MSG MainMsg = { };

CreateWindow(L"MainWndClass", L"TextEditor", WS\_OVERLAPPEDWINDOW | WS\_VISIBLE, 200, 100, 565, 360, NULL, NULL, NULL, NULL);

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

TranslateMessage(&MainMsg);

DispatchMessage(&MainMsg);

}

return 0;

}

WNDCLASS WindowClass(HBRUSH BGColor, HCURSOR Cursor, HINSTANCE hInst, LPCWSTR Name, WNDPROC wc) {

WNDCLASS nwc = { };

nwc.hbrBackground = BGColor;

nwc.hCursor = Cursor;

nwc.hInstance = hInst;

nwc.lpszClassName = Name;

nwc.lpfnWndProc = wc;

return nwc;

}

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

switch (uMsg) {

case WM\_COMMAND:

switch (wParam)

{

case OnOpenMenuClicked:

if (GetOpenFileNameA(&ofn))

{

LoadData(fileName);

lstrcpyA(currentFileName, fileName); // Сохраняем имя файла в переменной currentFileName

SetWindowTextA(hwnd, currentFileName);

loadFileLength = GetWindowTextLength(hEditConstrol);

isTextChanged = true;

}

break;

case OnSaveAsMenuClicked:

while (true)

{

if (GetSaveFileNameA(&ofn)) {

if (PathFileExistsA(fileName)) {

int result = MessageBoxW(hwnd, L"Файл с таким именем уже существует. Хотите перезаписать?", L"Подтвердить сохранение", MB\_YESNO | MB\_ICONQUESTION);

if (result == IDYES) {

SaveData(fileName);

MessageBoxW(hwnd, L"Файл сохранен!", L"Сохранение", MB\_OK);

isTextChanged = false;

break;

}

}

else {

SaveData(fileName);

isTextChanged = false;

break;

}

}

else {

break;

}

}

break;

case OnButtonClicked:

SetWindowTextW(hEditConstrol, L"");

selectedColor = RGB(0, 0, 0);

selectedBgColor = RGB(255, 255, 255);

isBold = false;

isItalic = false;

isTextChanged = true;

break;

case OnSaveMenuClicked:

if (PathFileExistsA(fileName)) {

SaveData(fileName);

MessageBoxW(hwnd, L"Файл сохранен!", L"Сохранение", MB\_OK);

break;

}

else {

if (GetSaveFileNameA(&ofn)) {

SaveData(fileName);

MessageBoxW(hwnd, L"Файл сохранен!", L"Save", MB\_OK);

}

}

break;

case OnBoldClicked:

isBold = !isBold;

hFont = CreateMyFont(16, isBold, isItalic, false);

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

break;

case OnItalicClicked:

isItalic = !isItalic;

hFont = CreateMyFont(16, isBold, isItalic, false);

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

break;

case OnTextColorClicked:

ZeroMemory(&cc, sizeof(cc));

cc.lStructSize = sizeof(cc);

cc.hwndOwner = hwnd;

cc.lpCustColors = customColors;

if (ChooseColor(&cc))

{

selectedColor = cc.rgbResult;

InvalidateRect(hEditConstrol, NULL, TRUE);

}

break;

case OnBgColorClicked:

ZeroMemory(&cc, sizeof(cc));

cc.lStructSize = sizeof(cc);

cc.hwndOwner = hwnd;

cc.lpCustColors = customColors;

if (ChooseColor(&cc))

{

selectedBgColor = cc.rgbResult;

InvalidateRect(hEditConstrol, NULL, TRUE);

}

break;

break;

default:

break;

}

break;

case WM\_CTLCOLOREDIT:

dc = (HDC)wParam;

SetTextColor(dc, selectedColor);

hBrush = CreateSolidBrush(selectedBgColor);

SetBkColor(dc, selectedBgColor); // Установка цвета фона

return (LRESULT)hBrush;

break;

case WM\_CREATE:

MainWndAddMenu(hwnd);

MainWndAddWidget(hwnd);

SetOpenFileParams(hwnd);

break;

case WM\_DESTROY:

length = GetWindowTextLength(hEditConstrol);

if (length == loadFileLength) {

PostQuitMessage(0);

break;

}

if (isTextChanged || (isTextChanged == false && length > 1))

{

int result = MessageBox(hwnd, L"Хотите сохранить изменения?", L"Подтверждение", MB\_YESNO);

if (result == IDYES)

{

if (PathFileExistsA(currentFileName))

{

SaveData(currentFileName);

}

else

{

if (GetSaveFileNameA(&ofn))

{

SaveData(fileName);

}

}

}

}

PostQuitMessage(0);

break;

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

}

}

void MainWndAddMenu(HWND hwnd) {

HMENU RootMenu = CreateMenu();

AppendMenu(RootMenu, MF\_STRING, OnOpenMenuClicked, L"Открыть");

AppendMenu(RootMenu, MF\_STRING, OnSaveMenuClicked, L"Сохранить");

AppendMenu(RootMenu, MF\_STRING, OnSaveAsMenuClicked, L"Сохранить как...");

AppendMenu(RootMenu, MF\_STRING, OnBoldClicked, L"Bold");

AppendMenu(RootMenu, MF\_STRING, OnItalicClicked, L"Italic");

AppendMenu(RootMenu, MF\_STRING, OnTextColorClicked, L"Text color");

AppendMenu(RootMenu, MF\_STRING, OnBgColorClicked, L"Bg color");

SetMenu(hwnd, RootMenu);

}

void MainWndAddWidget(HWND hwnd) {

hEditConstrol = CreateWindow(L"edit", L" ", WS\_VISIBLE | WS\_CHILD | ES\_MULTILINE | WS\_VSCROLL, 30, 30, 490, 200, hwnd, NULL, NULL, NULL);

CreateWindowA("button", "Clear", WS\_VISIBLE | WS\_CHILD, 30, 240, 475, 30, hwnd, (HMENU)OnButtonClicked, NULL, NULL);

hFont = CreateMyFont(16, isBold, isItalic, false);

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

}

void SaveData(LPCSTR path) {

HANDLE FileToSave = CreateFileA(

path,

GENERIC\_WRITE,

0,

NULL,

CREATE\_ALWAYS,

FILE\_ATTRIBUTE\_NORMAL,

NULL);

int saveLenth = GetWindowTextLength(hEditConstrol) + 1;

char\* data = new char[saveLenth];

saveLenth = GetWindowTextA(hEditConstrol, data, saveLenth);

DWORD bytesIterated;

WriteFile(FileToSave, data, saveLenth, &bytesIterated, NULL);

CloseHandle(FileToSave);

delete[] data;

}

void LoadData(LPCSTR path) {

HANDLE FileToLoad = CreateFileA(

path,

GENERIC\_READ,

0,

NULL,

OPEN\_EXISTING,

FILE\_ATTRIBUTE\_NORMAL,

NULL);

DWORD saveSize = GetFileSize(FileToLoad, NULL) + 1;

char\* data = new char[saveSize];

DWORD bytesIterated;

ReadFile(FileToLoad, data, saveSize, &bytesIterated, NULL);

data[bytesIterated] = '\0';

SetWindowTextA(hEditConstrol, data);

CloseHandle(FileToLoad);

delete[] data;

}

void SetOpenFileParams(HWND hwnd) {

ZeroMemory(&ofn, sizeof(ofn));

ofn.lStructSize = sizeof(ofn);

ofn.hwndOwner = hwnd;

ofn.lpstrFile = fileName;

ofn.nMaxFile = sizeof(fileName);

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

ofn.lpstrFileTitle = NULL;

ofn.nMaxFileTitle = 0;

ofn.lpstrInitialDir = "D:\\С++\\WinApi\\lab1";

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

}

HFONT CreateMyFont(int fontSize, bool isBold, bool isItalic, bool isUnderline) {

int fontWeight = isBold ? FW\_BOLD : FW\_NORMAL;

DWORD fontStyle = isItalic ? (DWORD)ItalicFont : 0;

fontStyle |= isUnderline ? (DWORD)UnderlineFont : 0;

HFONT hFont = CreateFont(

fontSize, // Размер шрифта

0, // Ширина символов (0 для автоматического)

0, // Угол наклона (0 для нормального)

0, // Угол наклона базовой линии (0 для нормального)

fontWeight, // Толщина шрифта

fontStyle, // Стиль шрифта (курсив, подчеркивание)

0, // Подчеркивание

0, // Зачеркивание

DEFAULT\_CHARSET, // Набор символов

OUT\_OUTLINE\_PRECIS, // Точность вывода

CLIP\_DEFAULT\_PRECIS, // Точность обрезки

CLEARTYPE\_QUALITY, // Качество рендеринга

DEFAULT\_PITCH | FF\_DONTCARE, // Расстояние между символами и семейство шрифта

L"Arial" // Имя шрифта

);

return hFont;

}