Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные системы и системное программирование

Отчёт

к лабораторной работе

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)

Студент: гр.153502

Кирзнер А. П.

Проверил: Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

1 Цель работы 3

2 Теоретические сведения 4

3 Результат выполнения программы 5

Список использованных источников 7

Приложение А 8

# 1 ЦЕЛЬ РАБОТЫ

Расширенное использование оконного интерфейса *Win* 32 и *GDI*. Формирование сложных изображений, создание и использование элементов управления, обработка различных сообщений, механизм перехвата сообщений (*winhook*).

Разработать текстовый редактор с поддержкой настраиваемых тем оформления (стили текста, цвета фона).

# 2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Интерфейс графических устройств *Microsoft* *Windows* (*GDI*) позволяет приложениям использовать графику и отформатированный текст как на видеодисплее, так и на принтере. Приложения на основе *Windows* не обращаются к графическому оборудованию напрямую. Вместо этого *GDI* взаимодействует с драйверами устройств от имени приложений.

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.

Функция *ChooseColor* позволяет создать диалоговое окно, позволяющее пользователю выбрать цвет. Если пользователь нажимает кнопку «ОК» в диалоговом окне, возвращаемое значение не равно нулю. Член *rgbResult* структуры *CHOOSECOLOR* содержит значение цвета *RGB* для цвета, выбранного пользователем. Если пользователь отменяет или закрывает диалоговое окно или возникает ошибка, возвращаемое значение равно нулю.

# 3 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ ПРОГРАММЫ

В результате выполнения лабораторной работы получилась улучшенная версия предыдущей лабораторной: добавились элементы в меню для стилизации текста и фона во время редактирования (рисунок 1).

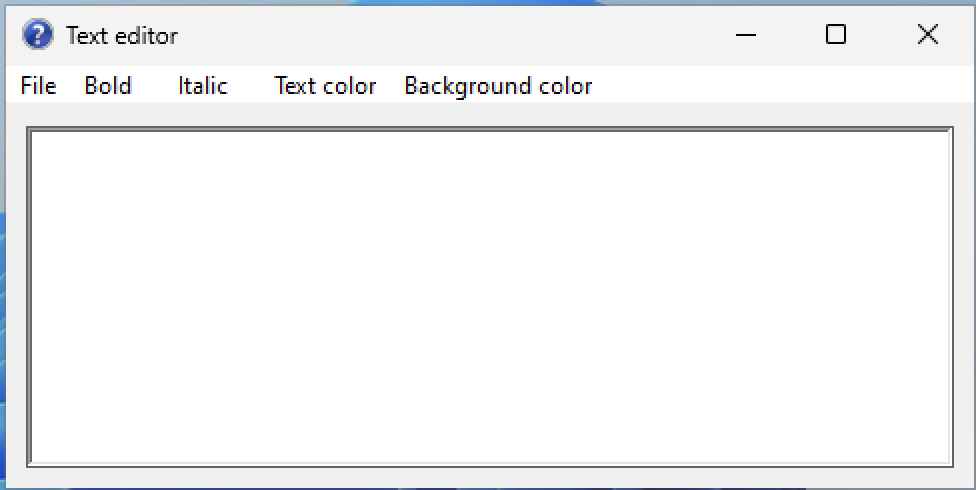


Рисунок 1 – Графический интерфейс программы

В соответствии с рисунком 2 при нажатии на «*Bold*» и «*Italic*» будет применён стиль к отображаемому тексту соответственно.

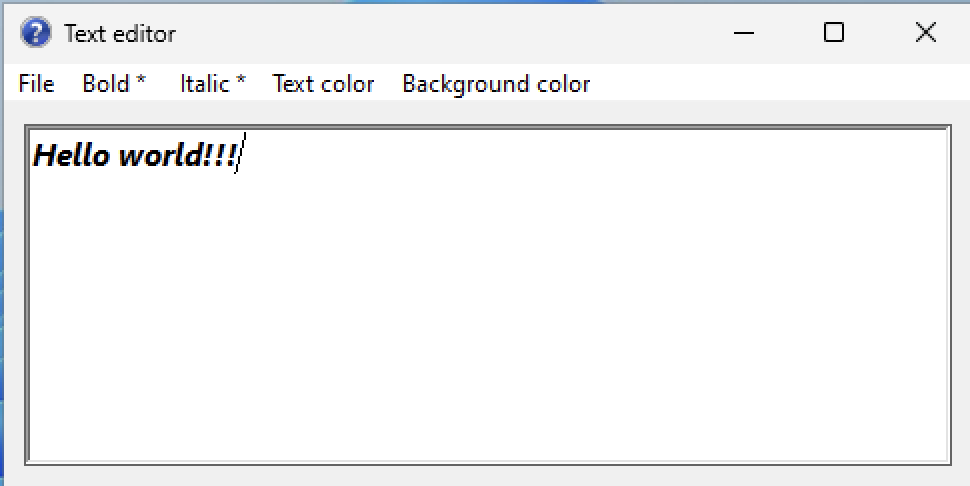


Рисунок 2 – Применение стиля к тексту

При выборе «*Text color*» или «*Background color*» появляется меню для выбора цвета для текста и фона соответственно. Палитра содержит 48 начальных цветов, также есть возможность задать свой собственный цвет, отличный от предложенных (рисунок 3).

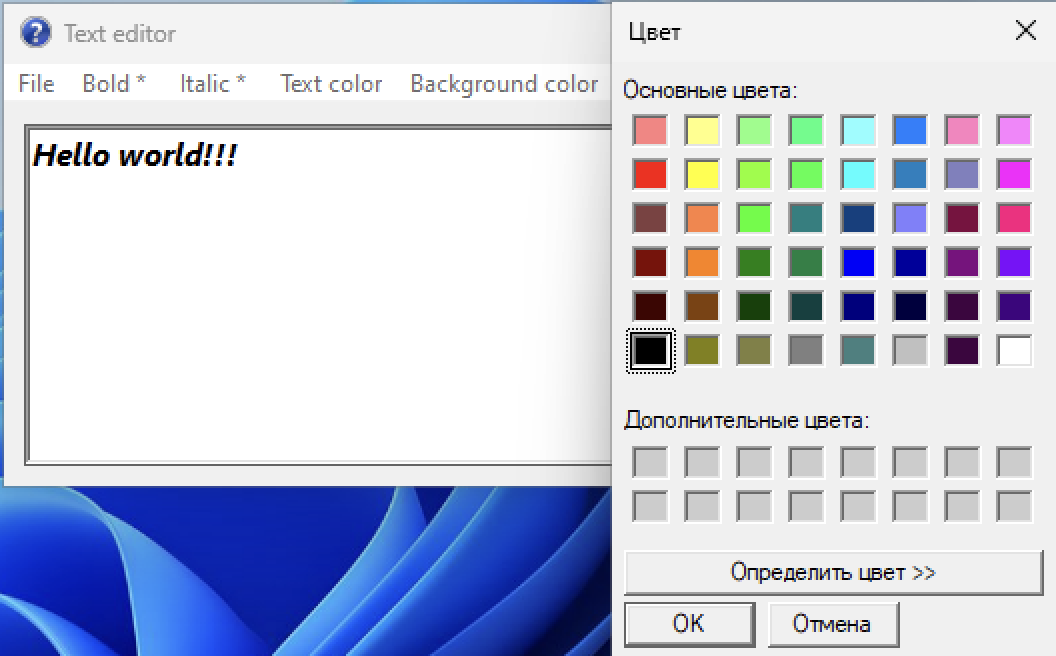


Рисунок 3 – Выбор цвета

После выбора цветов можно заметить изменение внешнего вида текстового редактора (рисунок 4).

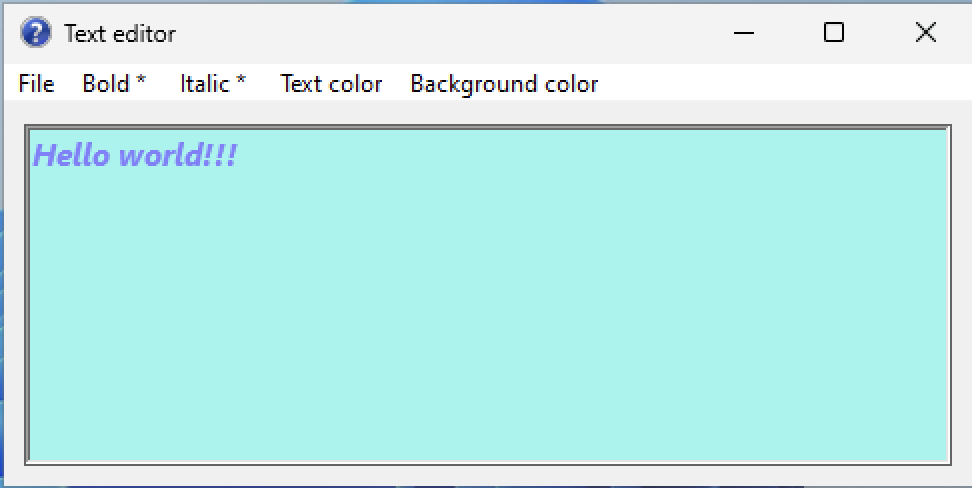


Рисунок 4 – Отображение содержимого выбранного файла

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Windows GDI [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/gdi/windows-gdi>.

[2] Цветовые палитры (Windows GDI) [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/gdi/color-palettes>.

[3] 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**

**Файл Lab2.h**

#pragma once

#define OnSaveFile 1

#define OnSaveFileAs 2

#define OnLoadFile 3

#define OnBold 4

#define OnItalic 5

#define OnTextColor 6

#define OnBgColor 7

#define TextBufferSize 256

char Buffer[TextBufferSize];

bool isFileLoaded = false;

bool isBold = false;

bool isItalic = false;

HWND hEditControl;

char filename[260];

OPENFILENAMEA ofn;

CHARFORMAT cf{ };

COLORREF textColor = RGB(0, 0, 0);

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

LRESULT CALLBACK SoftwareMainProcedure(HWND hWnd, UINT msg, WPARAM wp, LPARAM lp);

WNDCLASS NewWindowClass(HBRUSH BGColor, HCURSOR Cursor, HINSTANCE hInst, HICON Icon, LPCWSTR Name, WNDPROC Procedure);

void MainWndMenus(HWND hWnd);

void MainWndAddWidgets(HWND hWnd);

void SaveData(LPCSTR path);

void LoadData(LPCSTR path);

void SetOpenFileParams(HWND hWnd);

LPCSTR GetFileName(LPCSTR filepath);

bool FileExists(LPCSTR path);

bool HasTextChanged(HWND hEditControl, LPCSTR filepath);

void LoadFile(HWND hWnd);

void ChangeWindowTitle(HWND hWnd);

**Файл Lab2.cpp**

#include <windows.h>

#include <richedit.h>

#include <commdlg.h>

#include "Lab2.h"

int WINAPI WinMain(HINSTANCE hInst, HINSTANCE hPrevInst, LPSTR args, int ncmdshows) {

WNDCLASS SoftwareMainClass = NewWindowClass((HBRUSH)COLOR\_WINDOW, LoadCursor(NULL, IDC\_ARROW), hInst, LoadIcon(NULL, IDI\_QUESTION), L"MainWndClass", SoftwareMainProcedure);

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

MSG SoftwareMainMessage = { 0 };

CreateWindow(L"MainWndClass", L"Text editor", WS\_OVERLAPPEDWINDOW | WS\_VISIBLE, 100, 100, 500, 250, NULL, NULL, NULL, NULL);

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

TranslateMessage(&SoftwareMainMessage);

DispatchMessage(&SoftwareMainMessage);

}

return 0;

}

WNDCLASS NewWindowClass(HBRUSH BGColor, HCURSOR Cursor, HINSTANCE hInst, HICON Icon, LPCWSTR Name, WNDPROC Procedure) {

WNDCLASS NWC = { 0 };

NWC.hCursor = Cursor;

NWC.hIcon = Icon;

NWC.hInstance = hInst;

NWC.lpszClassName = Name;

NWC.hbrBackground = BGColor;

NWC.lpfnWndProc = Procedure;

return NWC;

}

LRESULT CALLBACK SoftwareMainProcedure(HWND hWnd, UINT msg, WPARAM wp, LPARAM lp) {

switch (msg) {

case WM\_COMMAND:

switch (wp) {

case OnLoadFile:

if (HasTextChanged(hEditControl, filename)) {

int result = MessageBoxA(hWnd, "Do you want to save changes?", "Confirmation", MB\_YESNOCANCEL | MB\_APPLMODAL);

if (result == IDYES) {

SendMessage(hWnd, WM\_COMMAND, isFileLoaded ? OnSaveFile : OnSaveFileAs, 0);

LoadFile(hWnd);

}

else if (result == IDNO) {

LoadFile(hWnd);

}

}

else {

LoadFile(hWnd);

}

break;

case OnSaveFile:

if (FileExists(filename)) {

SaveData(filename);

}

break;

case OnSaveFileAs:

while (true) {

if (GetSaveFileNameA(&ofn)) {

if (FileExists(filename)) {

int result = MessageBoxA(hWnd, "File already exists. Do you want to overwrite it?", "Confirmation", MB\_YESNO | MB\_ICONQUESTION);

if (result == IDYES) {

SaveData(filename);

ChangeWindowTitle(hWnd);

break;

}

}

else {

SaveData(filename);

ChangeWindowTitle(hWnd);

break;

}

}

else {

break;

}

}

break;

case OnBold: {

MENUITEMINFO menuItemInfo = { 0 };

menuItemInfo.cbSize = sizeof(MENUITEMINFO);

menuItemInfo.fMask = MIIM\_STRING;

UINT menuItemID = OnBold;

const char\* narrowString;

if (isBold) {

cf.dwEffects &= ~CFE\_BOLD;

narrowString = "Bold ";

}

else {

cf.dwEffects |= CFE\_BOLD;

narrowString = "Bold \* ";

}

isBold = !isBold;

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

int bufferSize = MultiByteToWideChar(CP\_UTF8, 0, narrowString, -1, NULL, 0);

wchar\_t\* wideString = new wchar\_t[bufferSize];

MultiByteToWideChar(CP\_UTF8, 0, narrowString, -1, wideString, bufferSize);

menuItemInfo.dwTypeData = wideString;

SetMenuItemInfo(GetMenu(hWnd), menuItemID, FALSE, &menuItemInfo);

break;

}

case OnItalic: {

MENUITEMINFO menuItemInfo = { 0 };

menuItemInfo.cbSize = sizeof(MENUITEMINFO);

menuItemInfo.fMask = MIIM\_STRING;

UINT menuItemID = OnItalic;

const char\* narrowString;

if (isItalic) {

cf.dwEffects &= ~CFE\_ITALIC;

narrowString = "Italic ";

}

else {

cf.dwEffects |= CFE\_ITALIC;

narrowString = "Italic \* ";

}

isItalic = !isItalic;

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

int bufferSize = MultiByteToWideChar(CP\_UTF8, 0, narrowString, -1, NULL, 0);

wchar\_t\* wideString = new wchar\_t[bufferSize];

MultiByteToWideChar(CP\_UTF8, 0, narrowString, -1, wideString, bufferSize);

menuItemInfo.dwTypeData = wideString;

SetMenuItemInfo(GetMenu(hWnd), menuItemID, FALSE, &menuItemInfo);

delete[] wideString;

break;

}

case OnTextColor: {

CHOOSECOLOR cc;

COLORREF acrCustClr[6];

ZeroMemory(&cc, sizeof(cc));

cc.lStructSize = sizeof(cc);

cc.hwndOwner = hWnd;

cc.lpCustColors = (LPDWORD)acrCustClr;

cc.rgbResult = textColor;

if (ChooseColor(&cc) == TRUE) {

COLORREF chosenColor = cc.rgbResult;

cf.crTextColor = chosenColor;

textColor = chosenColor;

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

}

break;

}

case OnBgColor: {

CHOOSECOLOR bgcc;

COLORREF acrCustClr[6];

ZeroMemory(&bgcc, sizeof(bgcc));

bgcc.lStructSize = sizeof(bgcc);

bgcc.hwndOwner = hWnd;

bgcc.lpCustColors = (LPDWORD)acrCustClr;

bgcc.rgbResult = bgColor;

if (ChooseColor(&bgcc) == TRUE) {

COLORREF chosenColor = bgcc.rgbResult;

bgColor = chosenColor;

SendMessage(hEditControl, EM\_SETBKGNDCOLOR, 0, chosenColor);

}

break;

}

}

break;

case WM\_CREATE:

MainWndMenus(hWnd);

MainWndAddWidgets(hWnd);

SetOpenFileParams(hWnd);

break;

case WM\_SIZE:

RECT clientRect;

GetClientRect(hWnd, &clientRect);

MoveWindow(hEditControl, 10, 10, clientRect.right - 20, clientRect.bottom - 20, TRUE);

break;

case WM\_DESTROY:

if (HasTextChanged(hEditControl, filename)) {

int result = MessageBoxA(hWnd, "Do you want to save changes?", "Confirmation", MB\_YESNO | MB\_APPLMODAL);

if (result == IDYES) {

SendMessage(hWnd, WM\_COMMAND, isFileLoaded ? OnSaveFile : OnSaveFileAs, 0);

}

PostQuitMessage(0);

}

else {

PostQuitMessage(0);

}

break;

default: return DefWindowProc(hWnd, msg, wp, lp);

}

}

void MainWndMenus(HWND hWnd)

{

HMENU RootMenu = CreateMenu();

HMENU SubMenu = CreateMenu();

AppendMenu(SubMenu, MF\_STRING, OnLoadFile, L"Load");

AppendMenu(SubMenu, MF\_STRING | MF\_GRAYED, OnSaveFile, L"Save");

AppendMenu(SubMenu, MF\_STRING, OnSaveFileAs, L"Save as");

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

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

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

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

AppendMenu(RootMenu, MF\_STRING, OnBgColor, L"Background color");

SetMenu(hWnd, RootMenu);

}

void MainWndAddWidgets(HWND hWnd) {

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

hEditControl = CreateWindowEx(0, MSFTEDIT\_CLASS, TEXT(""),

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

0, 0, 200, 200,

hWnd, NULL, NULL, NULL);

cf.cbSize = sizeof(CHARFORMAT);

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

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

cf.yHeight = (12 \* 1440) / 72;

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

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

}

void SaveData(LPCSTR path) {

HANDLE FileToSave = CreateFileA(

path,

GENERIC\_WRITE,

0,

NULL,

CREATE\_ALWAYS,

FILE\_ATTRIBUTE\_NORMAL,

NULL);

int saveLength = GetWindowTextLength(hEditControl) + 1;

char\* data = new char[saveLength];

saveLength = GetWindowTextA(hEditControl, data, saveLength);

DWORD bytesIterated;

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

CloseHandle(FileToSave);

delete[] data;

}

void LoadData(LPCSTR path) {

SetWindowTextA(hEditControl, "");

HANDLE FileToLoad = CreateFileA(

path,

GENERIC\_READ,

0,

NULL,

OPEN\_EXISTING,

FILE\_ATTRIBUTE\_NORMAL,

NULL);

DWORD fileSize = GetFileSize(FileToLoad, NULL);

char\* data = new char[fileSize + 1];

DWORD bytesRead = 0;

ReadFile(FileToLoad, data, fileSize, &bytesRead, NULL);

data[bytesRead] = '\0';

CloseHandle(FileToLoad);

SetWindowTextA(hEditControl, data);

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 = ".txt\0\*.txt\0";

ofn.lpstrFileTitle = NULL;

ofn.nMaxFileTitle = 0;

ofn.lpstrInitialDir = NULL;

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

}

LPCSTR GetFileName(LPCSTR filepath) {

LPCSTR filename = strrchr(filepath, '\\');

if (filename) {

filename++;

}

else {

filename = filepath;

}

return filename;

}

bool FileExists(LPCSTR path) {

DWORD attributes = GetFileAttributesA(path);

return (attributes != INVALID\_FILE\_ATTRIBUTES && !(attributes & FILE\_ATTRIBUTE\_DIRECTORY));

}

bool HasTextChanged(HWND hEditControl, LPCSTR filepath) {

bool result;

int editLength = GetWindowTextLength(hEditControl) + 1;

char\* editText = new char[editLength];

GetWindowTextA(hEditControl, editText, editLength);

if (FileExists(filepath)) {

HANDLE FileToCheck = CreateFileA(

filepath,

GENERIC\_READ,

FILE\_SHARE\_READ,

NULL,

OPEN\_EXISTING,

FILE\_ATTRIBUTE\_NORMAL,

NULL);

DWORD fileSize = GetFileSize(FileToCheck, NULL);

char\* fileData = new char[fileSize + 1];

DWORD bytesRead = 0;

ReadFile(FileToCheck, fileData, fileSize, &bytesRead, NULL);

fileData[bytesRead] = '\0';

CloseHandle(FileToCheck);

int editLength = GetWindowTextLength(hEditControl) + 1;

char\* editText = new char[editLength];

GetWindowTextA(hEditControl, editText, editLength);

result = strcmp(fileData, editText) != 0;

delete[] fileData;

delete[] editText;

return result;

}

else {

result = strcmp("", editText) != 0;

}

delete[] editText;

return result;

}

void ChangeWindowTitle(HWND hWnd) {

char newTitle[300];

strcpy\_s(newTitle, GetFileName(filename));

strcat\_s(newTitle, " - Text editor");

SetWindowTextA(hWnd, newTitle);

}

void LoadFile(HWND hWnd) {

if (GetOpenFileNameA(&ofn)) {

LoadData(filename);

ChangeWindowTitle(hWnd);

EnableMenuItem(GetMenu(hWnd), OnSaveFile, MF\_ENABLED);

isFileLoaded = true;

}

}