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 Миненков Я.А.

Проверил ассистент кафедры информатики Гриценко Н.Ю.

Минск 2023

СОДЕРЖАНИЕ

1 Формулировка задачи 3
2 Теоритические сведения 4
3 Описание функций программы 5
3.1 Изменение шрифта 5
3.2 Изменение фона 6
Список использованных источников 8
Приложение А (обязательное) Листинг кода 9

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 реализовать возможность смены фона элемента редактирования текста, смены семейства шрифта и его различных параметров.

2 ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ

Настройка цвета фона окна редактирования в *Windows* осуществляется с помощью диалогового окна выбора цветов, предоставляемого операционной системой. Этот процесс начинается с вызова функции *ChooseColor()*, которая открывает диалоговое окно выбора цвета. Когда пользователь выбирает цвет и закрывает диалоговое окно, *ChooseColor()* возвращает *TRUE*, и выбранный цвет сохраняется в структуре *CHOOSECOLOR*. Затем новый цвет используется для изменения цвета фона окна редактирования с помощью сообщения *EM_SETBKGNDCOLOR*.

Для изменения шрифта текста используется диалоговое окно выбора шрифтов, предоставляемое операционной системой. Это достигается с помощью функции *ChooseFont()*, которая открывает диалоговое окно выбора шрифтов. Когда пользователь выбирает шрифт и закрывает диалог, *ChooseFont()* возвращает *TRUE*, и информация о выбранном шрифте сохраняется в структуре *LOGFONT*. Эта информация используется для создания нового шрифта с помощью *CreateFontIndirect()*. Наконец, мы применяем этот новый шрифт к окну редактирования с помощью сообщения *EM_SETCHARFORMAT*. Этот новый шрифт к тексту в окне редактирования, отправляя сообщение *EM_SETCHARFORMAT*. В *WinAPI*, большинство действий пользователя с интерфейсом генерируют сообщения. Эти сообщения затем отправляются в цикл обработки сообщений, где они обрабатываются.

Функция *WndProc* используется для обработки сообщений, отправленных окну приложения. К примеру, в данной лабораторной были добавлены такие сообщения, как *IDM_CHANGE_BG и IDM_SETFONT*.

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*.

3.1 Изменение шрифта

Для изменения шрифта требуется выбрать пункт в меню - *Styles*, далее *Font styles*. Выберите требуемые изменения шрифта (Рисунок 1).

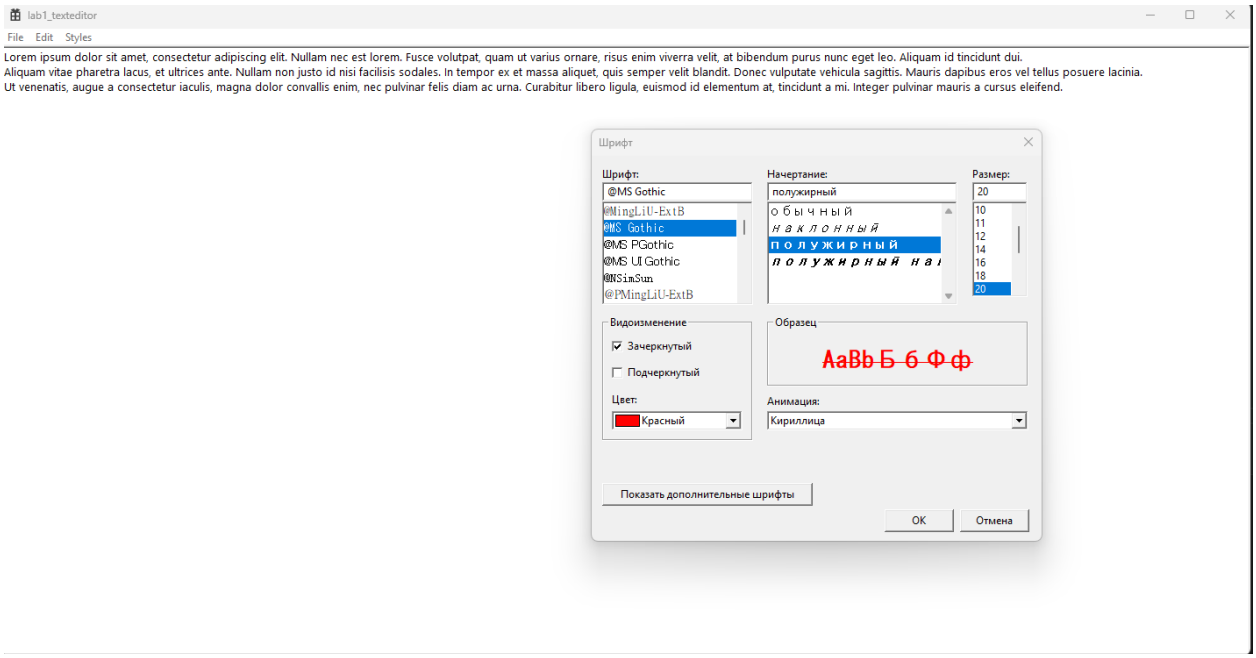

Рисунок 1 – Окно изменения параметров шрифта

Результат применения различных параметров шрифта (Рисунок 2).

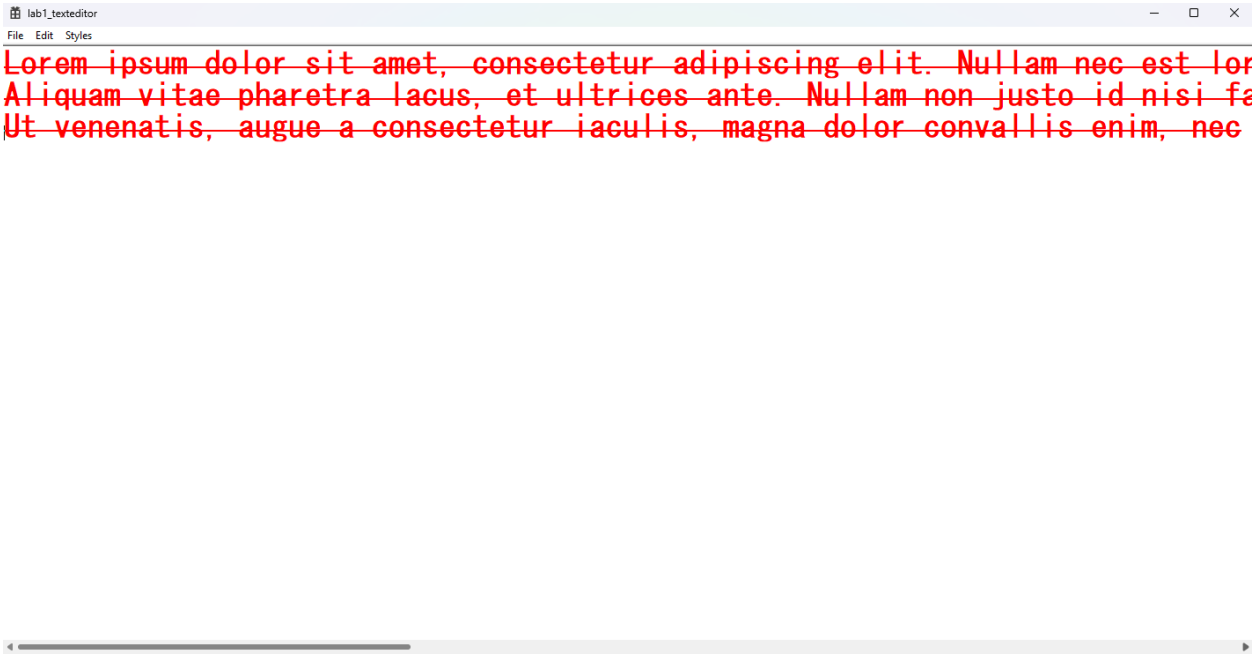

Рисунок 2 – Результат применения параметров

3.2 Изменение фона

Для изменения заднего фона требуется выбрать пункт в меню - *Styles*, далее *Background Color*. Выберите цвет фона (Рисунок 1).

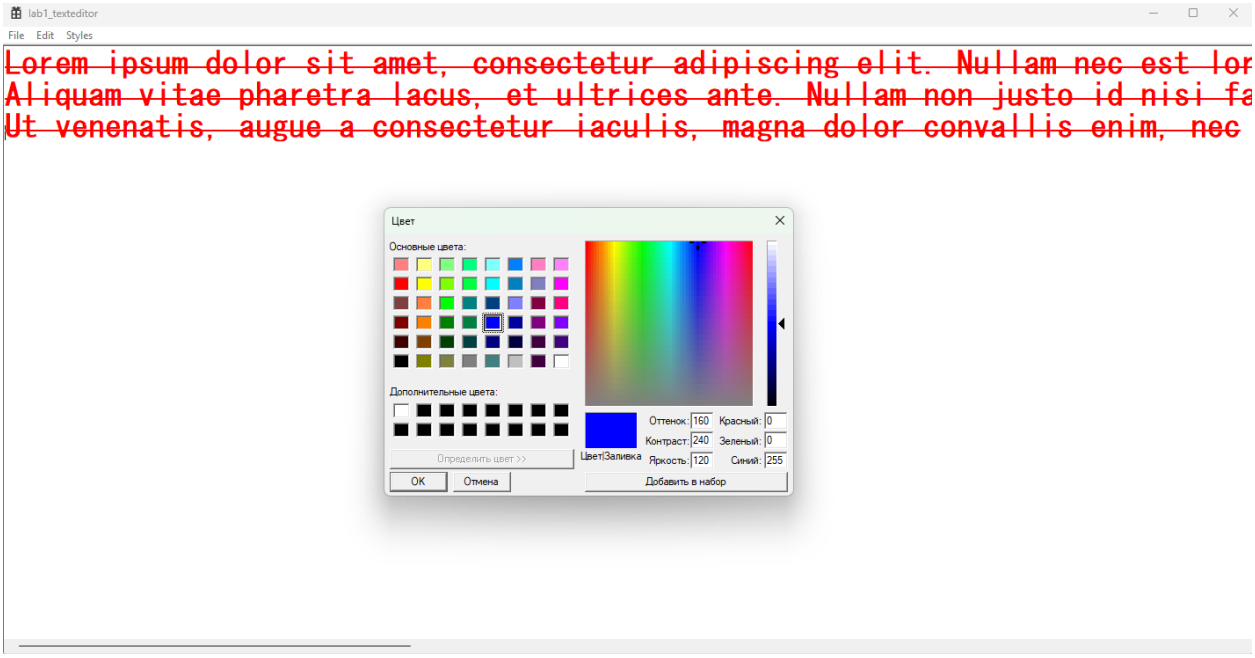

Рисунок 3 – Окно изменения фона

Результат изменения фона (Рисунок 4).

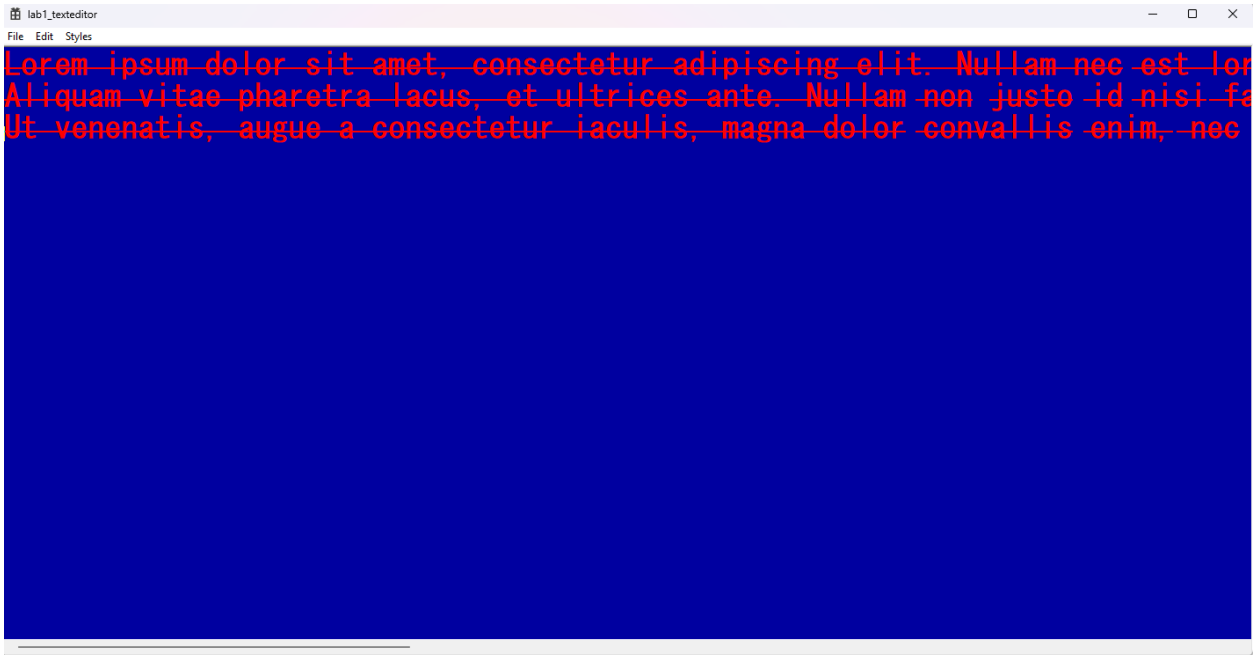

Рисунок 4 – Результат изменения фона

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] CHOOSECOLORA - Win32 apps [Электронный ресурс]. – Электронные данные. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/api/commdlg/ns-commdlg-choosecolora-r1

[2] CHOOSEFONTA (commdlg.h) - Win32 apps [Электронный ресурс]. – Электронные данные. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/api/commdlg/ns-commdlg-choosefonta

ПРИЛОЖЕНИЕ А

Листинг кода (Обязательное)
Файл lab1_texteditor.cpp

```
#include <windows.h>
#include "framework.h"
#include "lab1_texteditor.h"
#include "resource.h"
#include <commdlg.h>
#include <Richedit.h>

#define MAX_LOADSTRING 100

HINSTANCE hInst;                                // текущий экземпляр
WCHAR szTitle[MAX_LOADSTRING];                  // Текст строки заголовка
WCHAR szWindowClass[MAX_LOADSTRING];            // имя класса главного окна

ATOM MyRegisterClass(HINSTANCE hInstance);
BOOL InitInstance(HINSTANCE, int);
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
void CreateMainMenu(HWND hWnd);
HWND hEdit;


int APIENTRY wWinMain(_In_ HINSTANCE hInstance,
    _In_opt_ HINSTANCE hPrevInstance,
    _In_ LPWSTR    lpCmdLine,
    _In_ int       nCmdShow)
{
    UNREFERENCED_PARAMETER(hPrevInstance);
    UNREFERENCED_PARAMETER(lpCmdLine);

    LoadStringW(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
    LoadStringW(hInstance, IDC_LAB1TEXTEDITOR, szWindowClass, MAX_LOADSTRING);
    MyRegisterClass(hInstance);

    if (!InitInstance(hInstance, nCmdShow))
    {
        return FALSE;
    }

    MSG msg;

    while (GetMessage(&msg, nullptr, 0, 0))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    return (int)msg.wParam;
}

//регистрация класса окна
ATOM MyRegisterClass(HINSTANCE hInstance)
{
    WNDCLASSEXW wcex;

    wcex.cbSize = sizeof(WNDCLASSEX);

    wcex.style = CS_HREDRAW | CS_VREDRAW;
    wcex.lpfnWndProc = WndProc;
    wcex.cbClsExtra = 0;
    wcex.cbWndExtra = 0;
    wcex.hInstance = hInstance;
    wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_LAB1TEXTEDITOR));
```

```
    wcex.hCursor = LoadCursor(nullptr, IDC_ARROW);
    wcex.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
    wcex.lpszMenuName = nullptr;
    wcex.lpszClassName = szWindowClass;
    wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_SMALL));

    return RegisterClassExW(&wcex);
}

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
    hInst = hInstance;

    HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
       CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, nullptr, nullptr, hInstance, nullptr);

    if (!hWnd)
    {
       return FALSE;
    }

    ShowWindow(hWnd, nCmdShow);
    UpdateWindow(hWnd);

    // Создаем меню
    CreateMainMenu(hWnd);

    RegisterHotKey(hWnd, HOTKEY_CTRL_S, MOD_CONTROL, 'S');
    RegisterHotKey(hWnd, HOTKEY_CTRL_O, MOD_CONTROL, 'O');
    RegisterHotKey(hWnd, HOTKEY_CTRL_SHIFT_S, MOD_CONTROL | MOD_SHIFT, 'S');

    return TRUE;
}

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

    static HWND hEdit;
    static COLORREF clrBackground = RGB(255, 255, 255);
    static HBRUSH hbrBackground;
    switch (message)
    {
    case WM_CTLCOLORSTATIC:
    case WM_CTLCOLOREDIT:
       if ((HWND)lParam == hEdit)
       {
          HDC hdcStatic = (HDC)wParam;
          SetBkColor(hdcStatic, clrBackground);

          if (hbrBackground)
             DeleteObject(hbrBackground);
          hbrBackground = CreateSolidBrush(clrBackground);

          return (INT_PTR)hbrBackground;
       }
    break;
    case WM_HOTKEY:
    {
       int id = wParam;
       switch (id)
       {
       case HOTKEY_CTRL_O:
          SendMessage(hWnd, WM_COMMAND, IDM_OPEN, 0);
          break;
       case HOTKEY_CTRL_S:
          SendMessage(hWnd, WM_COMMAND, IDM_SAVE, 0);
          break;
       case HOTKEY_CTRL_SHIFT_S:
          SendMessage(hWnd, WM_COMMAND, IDM_SAVEAS, 0);
```

```
                    break;
                }
            }
            break;
            case WM_CREATE:
            {
                LoadLibrary(TEXT("Msftedit.dll"));
                hEdit = 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);

                if (hEdit == NULL)
                {
                    MessageBox(NULL, L"Cannot create edit control.", L"Error", MB_OK | MB_ICONERROR);
                    return -1;
                }
            }
            break;
            case WM_SIZE:
            {
                int newWidth = LOWORD(lParam);
                int newHeight = HIWORD(lParam);

                SetWindowPos(hEdit, NULL, 0, 0, newWidth, newHeight, SWP_NOZORDER);
            }
            break;
            case WM_CLOSE:
            {
                wchar_t szEditText[4096];
                GetWindowText(hEdit, szEditText, sizeof(szEditText) / sizeof(szEditText[0]));

                if (IsWindowVisible(hEdit) && wcscmp(szEditText, L"") != 0)
                {
                    int result = MessageBox(hWnd, L"Do you want to save the changes?", L"Save Changes", MB_YESNOCANCEL
| MB_ICONQUESTION);

                    if (result == IDYES)
                    {
                        SendMessage(hWnd, WM_COMMAND, IDM_SAVE, 0);
                    }
                    else if (result == IDCANCEL)
                    {
                        return 0;
                    }
                }

                UnregisterHotKey(hWnd, HOTKEY_CTRL_O);
                UnregisterHotKey(hWnd, HOTKEY_CTRL_S);
                UnregisterHotKey(hWnd, HOTKEY_CTRL_SHIFT_S);
                DestroyWindow(hWnd);
            }
            break;
            case WM_COMMAND:
            {
                int wmId = LOWORD(wParam);
                switch (wmId)
                {
                case IDM_CHANGE_BG:
                {
                    CHOOSECOLOR cc;
                    COLORREF acrCustClr[16] = { RGB(255, 255, 255) };
```

```
                ZeroMemory(&cc, sizeof(cc));
                cc.lStructSize = sizeof(cc);
                cc.hwndOwner = hWnd;
                cc.lpCustColors = (LPDWORD)acrCustClr;
                cc.rgbResult = clrBackground;
                cc.Flags = CC_FULLOPEN | CC_RGBINIT;

                if (ChooseColor(&cc))
                {
                    clrBackground = cc.rgbResult;
                    SendMessage(hEdit, EM_SETBKGNDCOLOR, 0, clrBackground);
                }

                InvalidateRect(hWnd, NULL, TRUE);
            }
            break;
            case IDM_SETFONT:
            {
                LOGFONT lfont;
                CHOOSEFONT cFont;
                ZeroMemory(&cFont, sizeof(CHOOSEFONT));
                cFont.lStructSize = sizeof(CHOOSEFONT);
                cFont.hwndOwner = hWnd;
                cFont.lpLogFont = &lfont;
                cFont.Flags = CF_SCREENFONTS | CF_EFFECTS;

                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 = ((cFont.lpLogFont->lfWeight == FW_BOLD) ? CFE_BOLD : 0) |
                        ((cFont.lpLogFont->lfItalic) ? CFE_ITALIC : 0) |
                        ((cFont.lpLogFont->lfUnderline) ? CFE_UNDERLINE : 0) |
                        ((cFont.lpLogFont->lfStrikeOut) ? CFE_STRIKEOUT : 0);

                    SendMessage(hEdit, EM_SETCHARFORMAT, SCF_ALL, (LPARAM)&cf);
                }
            }
            break;
            break;
            case IDM_NEW:
                SetWindowText(hEdit, L"");
                break;
            case IDM_OPEN:
            {
                OPENFILENAME ofn;
                wchar_t szFileName[MAX_PATH] = L"";

                ZeroMemory(&ofn, sizeof(ofn));
                ofn.lStructSize = sizeof(ofn);
                ofn.hwndOwner = hWnd;
                ofn.lpstrFilter = L"Text Files (*.txt)\0*.txt\0All Files (*.*)\0*.*\0";
                ofn.lpstrFile = szFileName;
                ofn.nMaxFile = sizeof(szFileName);
                ofn.Flags = OFN_FILEMUSTEXIST;

                if (GetOpenFileName(&ofn))
                {
                    HANDLE hFile = CreateFile(ofn.lpstrFile, GENERIC_READ, FILE_SHARE_READ, NULL,
OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
```

```
                    if (hFile != INVALID_HANDLE_VALUE)
                    {
                        DWORD dwFileSize = GetFileSize(hFile, NULL);
                        if (dwFileSize != INVALID_FILE_SIZE)
                        {
                            wchar_t* szFileContent = new wchar_t[dwFileSize / sizeof(wchar_t) + 1];
                            if (szFileContent)
                            {
                                DWORD dwBytesRead;
                                if (ReadFile(hFile, szFileContent, dwFileSize, &dwBytesRead, NULL))
                                {
                                    szFileContent[dwFileSize / sizeof(wchar_t)] = L'\0';
                                    SetWindowText(hEdit, szFileContent);
                                }
                                delete[] szFileContent;
                            }
                        }
                        CloseHandle(hFile);
                    }
                }
            }
            break;
            case IDM_SAVE:
            {
                OPENFILENAME ofn;
                wchar_t szFileName[MAX_PATH] = L"";

                ZeroMemory(&ofn, sizeof(ofn));
                ofn.lStructSize = sizeof(ofn);
                ofn.hwndOwner = hWnd;
                ofn.lpstrFilter = L"Text Files (*.txt)\0*.txt\0All Files (*.*)\0*.*\0";
                ofn.lpstrFile = szFileName;
                ofn.nMaxFile = sizeof(szFileName);
                ofn.Flags = OFN_OVERWRITEPROMPT;

                if (GetSaveFileName(&ofn))
                {
                    HANDLE hFile = CreateFile(ofn.lpstrFile, GENERIC_WRITE, 0, NULL, CREATE_ALWAYS,
FILE_ATTRIBUTE_NORMAL, NULL);
                    if (hFile != INVALID_HANDLE_VALUE)
                    {
                        wchar_t szText[4096];
                        GetWindowText(hEdit, szText, sizeof(szText) / sizeof(szText[0]));

                        DWORD dwBytesWritten;
                        WriteFile(hFile, szText, lstrlen(szText) * sizeof(wchar_t), &dwBytesWritten, NULL);

                        CloseHandle(hFile);
                    }
                }
            }
            break;
            case IDM_SAVEAS:
            {
                OPENFILENAME ofn;
                wchar_t szFileName[MAX_PATH] = L"";

                ZeroMemory(&ofn, sizeof(ofn));
                ofn.lStructSize = sizeof(ofn);
                ofn.hwndOwner = hWnd;
                ofn.lpstrFilter = L"Text Files (*.txt)\0*.txt\0All Files (*.*)\0*.*\0";
                ofn.lpstrFile = szFileName;
                ofn.nMaxFile = sizeof(szFileName);
                ofn.Flags = OFN_OVERWRITEPROMPT;

                if (GetSaveFileName(&ofn))
                {
                    HANDLE hFile = CreateFile(ofn.lpstrFile, GENERIC_WRITE, 0, NULL, CREATE_ALWAYS,
FILE_ATTRIBUTE_NORMAL, NULL);
```

```
            if (hFile != INVALID_HANDLE_VALUE)
            {
               wchar_t szText[4096];
               GetWindowText(hEdit, szText, sizeof(szText) / sizeof(szText[0]));

               DWORD dwBytesWritten;
               WriteFile(hFile, szText, lstrlen(szText) * sizeof(wchar_t), &dwBytesWritten, NULL);

               CloseHandle(hFile);
            }
         }
      }
      break;
      case IDM_COPY:
         SendMessage(hEdit, WM_COPY, 0, 0);
         break;
      case IDM_PASTE:
         SendMessage(hEdit, WM_PASTE, 0, 0);
         break;
      case IDM_CUT:
         SendMessage(hEdit, WM_CUT, 0, 0);
         break;
      case IDM_SELECTALL:
         SendMessage(hEdit, EM_SETSEL, 0, -1);
         break;

      default:
         return DefWindowProc(hWnd, message, wParam, lParam);
      }
   }
   break;
   case WM_DESTROY:
      PostQuitMessage(0);
      break;
   default:
      return DefWindowProc(hWnd, message, wParam, lParam);
   }

   return 0;
}

void CreateMainMenu(HWND hWnd)
{
   HMENU hMenu = CreateMenu();

   // Меню "File"
   HMENU hFileMenu = CreateMenu();
   AppendMenu(hFileMenu, MF_STRING, IDM_NEW, L"New");
   AppendMenu(hFileMenu, MF_STRING, IDM_OPEN, L"Open");
   AppendMenu(hFileMenu, MF_STRING, IDM_SAVE, L"Save");
   AppendMenu(hFileMenu, MF_STRING, IDM_SAVEAS, L"Save As");
   AppendMenu(hFileMenu, MF_SEPARATOR, 0, nullptr);
   //AppendMenu(hFileMenu, MF_STRING, IDM_CHANGE_BG, L"Background Color");
   //AppendMenu(hFileMenu, MF_STRING, IDM_SETFONT, L"Set font");
   AppendMenu(hFileMenu, MF_STRING, IDM_EXIT, L"Exit");
   AppendMenu(hMenu, MF_POPUP, (UINT_PTR)hFileMenu, L"File");

   // Меню "Edit"
   HMENU hEditMenu = CreateMenu();
   AppendMenu(hEditMenu, MF_STRING, IDM_COPY, L"Copy");
   AppendMenu(hEditMenu, MF_STRING, IDM_PASTE, L"Paste");
   AppendMenu(hEditMenu, MF_STRING, IDM_CUT, L"Cut");
   AppendMenu(hEditMenu, MF_SEPARATOR, 0, nullptr);
   AppendMenu(hEditMenu, MF_STRING, IDM_SELECTALL, L"Select All");
   AppendMenu(hMenu, MF_POPUP, (UINT_PTR)hEditMenu, L"Edit");

   HMENU hStylesMenu = CreateMenu();
   AppendMenu(hStylesMenu, MF_STRING, IDM_CHANGE_BG, L"Background Color");
   AppendMenu(hStylesMenu, MF_STRING, IDM_SETFONT, L"Font styles");
```

```
    AppendMenu(hMenu, MF_POPUP, (UINT_PTR)hStylesMenu, L"Styles");

    //// Меню "Help"
    //HMENU hHelpMenu = CreateMenu();
    //AppendMenu(hHelpMenu, MF_STRING, IDM_ABOUT, L"About");
    //AppendMenu(hMenu, MF_POPUP, (UINT_PTR)hHelpMenu, L"Help");

    SetMenu(hWnd, hMenu);
}
```