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 №1
на тему

ОСНОВЫ ПРОГРАММИРОВАНИЯ В WIN 32 API. ОКОННОЕ ПРИЛОЖЕНИЕ WIN 32 С МИНИМАЛЬНОЙ ДОСТАТОЧНОЙ ФУНКЦИОНАЛЬНОСТЬЮ. ОБРАБОТКА ОСНОВНЫХ ОКОННЫХ СООБЩЕНИЙ.

БГУИР КП 1-40 04 01

Выполнил: студент группы 153502
Кузнецов Е. А.

Проверил: Гриценко Н.Ю.

Минск 2023

СОДЕРЖАНИЕ

1 Цель работы

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 разработать текстовый редактор с поддержкой множества открытых документов. Реализовать функциональность сохранения и загрузки файлов.

2 Теоретические сведения

Win32 API (*Application Programming Interface*) – это набор функций и процедур, предоставляемых операционной системой *Windows* для разработки приложений на языке программирования *C/C++*. Оконное приложение *Win32* – это приложение, которое состоит из одного или нескольких окон, в которых происходит взаимодействие с пользователем.

Для создания окна необходимо зарегистрировать класс окна с помощью функции *RegisterClassEx* и создать окно с помощью функции *CreateWindowEx*. Окно может иметь различные свойства, такие как заголовок, размеры, стиль и обработчики сообщений.

Важным аспектом программирования в *Win32 API* является обработка оконных сообщений. Оконные сообщения – это события, которые происходят в окне, например, нажатие кнопки мыши или клавиши, изменение размера окна и другие действия пользователя.

Для обработки оконных сообщений необходимо определить функцию оконной процедуры (*WndProc*), которая будет вызываться системой при возникновении сообщения. В функции *WndProc* нужно обрабатывать различные типы сообщений с помощью условных операторов и выполнять соответствующие действия.

3 Описание функций программы

Согласно формулировке задачи, были спроектированы следующие функции программы:

– Открытие файла для чтения и редактирования;

– Сохранение файла;

– Открытие и закрытие вкладок.

1) Открытие файла для чтения и редактирования

Для выбора файла для открытия необходимо нажать на кнопку «Открыть» меню «Файл» (рисунок 1).

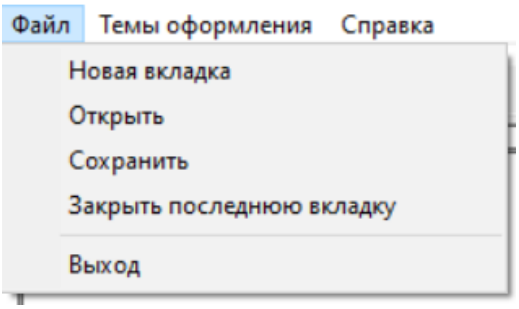

Рисунок 1 – Меню текстового редактора

Далее следует выбрать файл в открывшемся проводнике (рисунок 2) и нажать кнопку «Открыть».

.

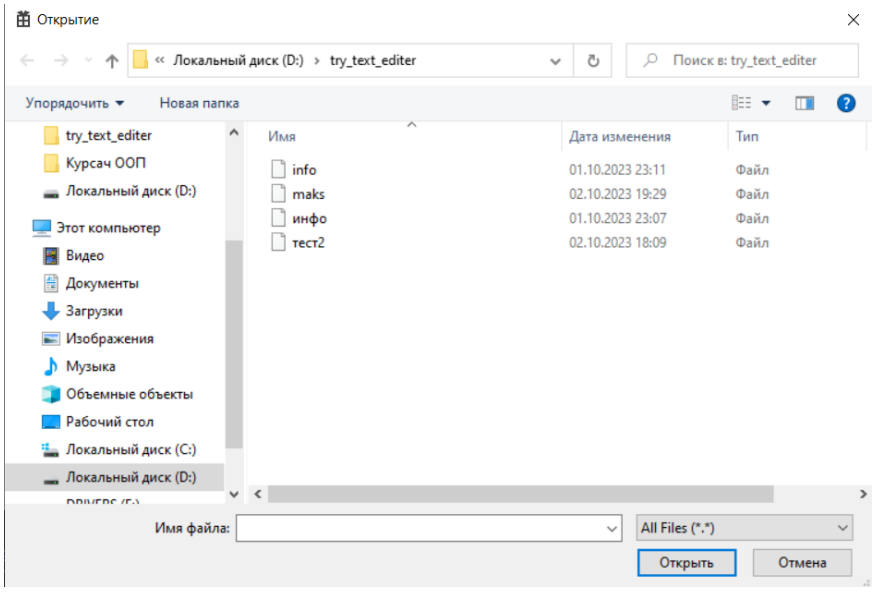

Рисунок 2 – Проводник открытия файла

2) Сохранение файла

Для сохранения файла необходимо нажать на кнопку «Сохранить» меню «Файл» (рисунок 1). Далее написать имя файла для сохранения в открывшемся проводнике (рисунок 3) и нажать кнопку «Сохранить».

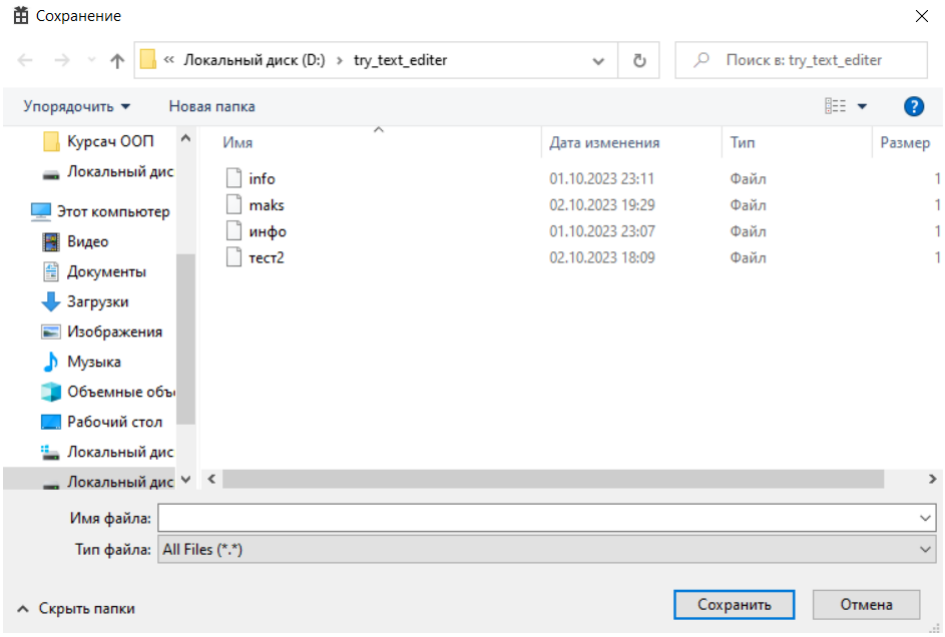

Рисунок 3 – Проводник сохранения файла

3) Открытие и закрытие вкладок

При запуске приложения открывается окно с пустой новой вкладкой (рисунок 4).

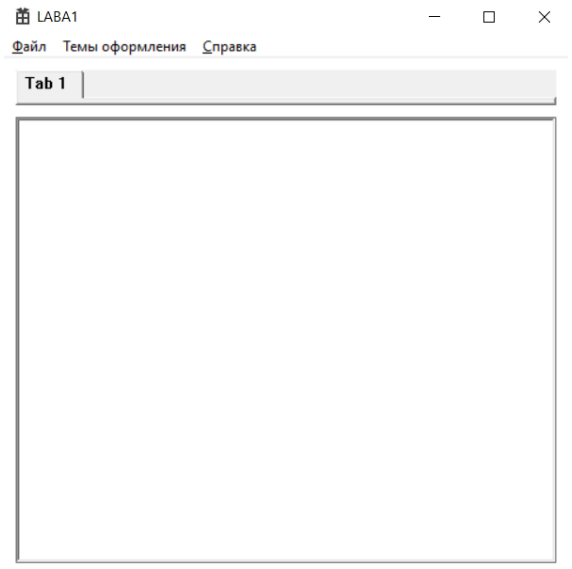

Рисунок 4 – Начальное состояние текстового редактора

Для открытия новой вкладки необходимо нажать на кнопку «Новая вкладка» меню «Файл» (рисунок 1). После нажатия данной кнопки открывается новая вкладка с редактором. (Рисунок 5).

Рисунок 5 – Открытие новой вкладки

Для закрытия последней вкладки необходимо нажать на кнопку «Закрыть последнюю вкладку» меню «Файл» (рисунок 1). После нажатия данной кнопки последняя вкладка закрывается. (Рисунок 6).

Рисунок 6 – Закрытие последней вкладки

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Начало работы с классическими приложениями для *Windows*, которые используют *API Win32* [электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/desktop-programming
Дата доступа 05.10.2023.
[2] Создание элемента управления "Вкладка" в главном окне [электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/controls/create-a-tab-control-in-the-main-window
Дата доступа 05.10.2023.

ПРИЛОЖЕНИЕ А
Листинг кода

Файл LABA1.h

```
#include "framework.h"
#include <Richedit.h>
#include "LABA1.h"
#include <string>
#include <commdlg.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <commctrl.h>



#define MAX_LOADSTRING 100
#define IDC_TABCONTROL 1101
#define IDC_EDIT_TAB_START 2000

// Глобальные переменные:
HINSTANCE hInst
WCHAR szTitle[MAX_LOADSTRING];
WCHAR szWindowClass[MAX_LOADSTRING];          окна
HWND tabControl;
int tabIndexCounter = 0;                                    // счетчик вкладок
static COLORREF acrCustClr[16];

ATOM              MyRegisterClass(HINSTANCE hInstance);     // будет
регистрировать класс окна
BOOL              InitInstance(HINSTANCE, int);             // будет
инициализировать экземпляр приложения.
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);        //
обработка сообщений для главного окна
INT_PTR CALLBACK    About(HWND, UINT, WPARAM, LPARAM);          //
обрабатывать сообщения для диалогового окна "О программе".

int APIENTRY wWinMain(_In_ HINSTANCE hInstance,
        _In_opt_ HINSTANCE hPrevInstance, //предыдущий экземпляр
        _In_ LPWSTR    lpCmdLine, // аргументы командной строки
```

```
    _In_ int       nCmdShow) // флаги отображения окна
{
    UNREFERENCED_PARAMETER(hPrevInstance); //предназначены для
предотвращения
    UNREFERENCED_PARAMETER(lpCmdLine); //предупреждений о
неиспользуемых параметрах функции.
    LoadLibrary(L"Msftedit.dll");

    // Инициализация глобальных строк
    LoadStringW(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
    LoadStringW(hInstance, IDC_LABA1, szWindowClass,
MAX_LOADSTRING);
    MyRegisterClass(hInstance);

    // Выполнить инициализацию приложения:
    if (!InitInstance(hInstance, nCmdShow)) // создания и отображения главного
окна приложения
    {
        return FALSE;
    }

    HACCEL hAccelTable = LoadAccelerators(hInstance,
MAKEINTRESOURCE(IDC_LABA1));

    MSG msg; // хранение сообщений

    // Цикл основного сообщения:
    while (GetMessage(&msg, nullptr, 0, 0))
    {
        if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    }

    return (int)msg.wParam;
}
```

```
ATOM MyRegisterClass(HINSTANCE hInstance)
{
	WNDCLASSEXW wcex; //  информация о классе окна

	wcex.cbSize = sizeof(WNDCLASSEX);

	wcex.style = CS_HREDRAW | CS_VREDRAW; //окно должно быть перерисовано, если его горизонтальный и вертикальный размер изменяется.
	wcex.hbrBackground = CreateSolidBrush(0x00FFFFFF);
	wcex.lpfnWndProc = WndProc; //оконная процедуру, которая будет обрабатывать сообщения для окна
	wcex.cbClsExtra = 0; //  Резервированные поля для
	wcex.cbWndExtra = 0; // дополнительных данных класса и окна
	wcex.hInstance = hInstance; // Установка экземпляра приложения, полученного как параметр функции
	wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_LABA1)); //значок приложения в окне
	wcex.hCursor = LoadCursor(nullptr, IDC_ARROW); // курсор мыши
	wcex.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1); // фон окна
	wcex.lpszMenuName = MAKEINTRESOURCEW(IDC_LABA1); // меню для окна
	wcex.lpszClassName = szWindowClass; // именя класса окна
	wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_SMALL)); // маленький значок приложения


	return RegisterClassExW(&wcex); // Регистрация класса окна и возврат атома класса
}
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
	hInst = hInstance; // Сохранить маркер экземпляра в глобальной переменной

	HWND hWnd = CreateWindow(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
		CW_USEDEFAULT, CW_USEDEFAULT, 500, 500, nullptr, nullptr, hInstance, nullptr);

	if (!hWnd)  // Проверка, было ли успешно создано окно
	{
```

```
		return FALSE;
	}

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

	return TRUE;
}
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM
wParam, LPARAM lParam)
{
	switch (message)
	{
	case WM_CREATE:
	{
		tabControl = CreateWindowEx(0, WC_TABCONTROL, L"",
WS_CHILD | WS_VISIBLE,
			10, 10, 460, 30, hWnd, (HMENU)IDC_TABCONTROL,
(HINSTANCE)GetWindowLong(hWnd, GWLP_HINSTANCE), nullptr);

		if (tabControl)
		{
			TCITEM tie;
			tie.mask = TCIF_TEXT;
			tabIndexCounter++;

			// Создайте первую вкладку
			tie.pszText = (LPWSTR)L"Tab 1";
			TabCtrl_InsertItem(tabControl, 0, &tie);

			CreateWindow(
				L"RICHEDIT50W",		// Используйте класс Rich Edit
Control
				L"",			// Текст по умолчанию
				WS_BORDER | WS_CHILD | WS_VISIBLE |
WS_VSCROLL | WS_HSCROLL | ES_MULTILINE,
				10, 50, 460, 380,	// Размеры и положение
				hWnd, (HMENU)IDC_EDIT_TAB_START, hInst, nullptr
			);
		}
	}
```

```
        break;
        case WM_NOTIFY:  // переключение вкладок
        {
                NMHDR* pnmhdr = (NMHDR*)lParam;
                if (pnmhdr->code == TCN_SELCHANGE)
                {
                        int currentTab = TabCtrl_GetCurSel(tabControl);

                        // Скрыть все текстовые поля
                        for (int i = 0; i < tabIndexCounter; i++)
                        {
                                ShowWindow(GetDlgItem(hWnd,
IDC_EDIT_TAB_START + i), SW_HIDE);
                        }

                        // Показать текстовое поле для выбранной вкладки
                        ShowWindow(GetDlgItem(hWnd, IDC_EDIT_TAB_START +
currentTab), SW_SHOW);
                }
        }
        break;
        case WM_COMMAND:
        {
                int wmId = LOWORD(wParam); // Извлечение идентификатора
команды
                int wmEvent = HIWORD(wParam);

                if (wmId == IDC_TABCONTROL) {
                        // Обработка событий переключения вкладок
                        int currentTab = TabCtrl_GetCurSel(tabControl);

                        // Скрыть все текстовые поля
                        for (int i = 0; i < tabIndexCounter; i++)
                        {
                                ShowWindow(GetDlgItem(hWnd,
IDC_EDIT_TAB_START + i), SW_HIDE);
                        }

                        // Показать текстовое поле для выбранной вкладки
                        ShowWindow(GetDlgItem(hWnd, IDC_EDIT_TAB_START +
currentTab), SW_SHOW);
```

```
            }
            else
            {
                switch (wmId)
                {
                case IDM_ABOUT:
                    DialogBox(hInst,
MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
                    break;
                case IDM_EXIT:
                    if (MessageBox(hWnd, L"Вы действительно хотите
закрыть окно?", L"Выход", MB_OKCANCEL) == IDOK) {
                        DestroyWindow(hWnd);
                    }
                    return 0;
                    break;
                case ID_32772: // SAVE
                {
                    int currentTab = TabCtrl_GetCurSel(tabControl);
                    int controlId = IDC_EDIT_TAB_START + currentTab;
                    HWND textEdit = GetDlgItem(hWnd, controlId);

                    static std::wstring fileName(MAX_PATH, L'\0'); //
хранение имени файла
                    OPENFILENAME ofn{}; // структура для настройки
диалогового окна сохранения файла
                    ofn.lStructSize = sizeof(OPENFILENAME); // размер
структуры
                    ofn.hwndOwner = hWnd; // дескриптор окна-владельца
(главного окна приложения)
                    ofn.lpstrFilter = L"Text Files (*.txt)\0*.txt\0All Files
(*.*)\0*.*\0";
                    ofn.lpstrFile = &fileName[0]; // указатель на строку, в
которую будет сохранено имя выбранного файла.
                    ofn.nMaxFile = MAX_PATH; //  максимальная длина
имени файла
                    ofn.Flags = OFN_OVERWRITEPROMPT; // флаг на
перезапись файла (если файл с таким именем существует)


                    if (!GetSaveFileName(&ofn)) {
```

```
                    MessageBox(hWnd, L"Не удалось получить имя
файла", L"Ошибка", MB_ICONINFORMATION);
                    return 0;
                }

                // открытие файла для записи
                std::wofstream file(fileName, std::ios::binary);

                if (file.is_open()) {
                    //Получения текста из элемента textEdit
                    int textLength = GetWindowTextLength(textEdit);
                    std::wstring text;
                    text.resize(textLength + 1);
                    file.imbue(std::locale("ru_RU.utf8"));
                    GetWindowText(textEdit, &text[0], textLength + 1);

                    // Запись текста в файл
                    file << text;
                    file.close();

                    MessageBox(hWnd, L"Данные успешно
сохранены!", L"Сохранение", MB_OK);
                }
                else {
                    MessageBox(hWnd, L"Не удалось открыть файл
для записи!", L"Ошибка", MB_OK | MB_ICONERROR);
                }
            }
            break;
            case ID_32771: // OPEN
            {
                int currentTab = TabCtrl_GetCurSel(tabControl);
                int controlId = IDC_EDIT_TAB_START + currentTab;
                HWND textEdit = GetDlgItem(hWnd, controlId);

                static std::wstring fileName(MAX_PATH, L'\0');
                OPENFILENAME ofn{};
                ofn.lStructSize = sizeof(OPENFILENAME);
                ofn.hwndOwner = hWnd;
                ofn.lpstrFilter = L"Text Files (*.txt)\0*.txt\0All Files
(*.*)\0*.*\0";
```

```
                        ofn.lpstrFile = &fileName[0];
                        ofn.nMaxFile = MAX_PATH;
                        ofn.Flags = OFN_FILEMUSTEXIST |
OFN_PATHMUSTEXIST | OFN_HIDEREADONLY;
                        // флаги: выбор существующего файла и пути (2й флаг)
и скрыть файлы только для чтения

                        if (!GetOpenFileName(&ofn)) {
                                MessageBox(hWnd, L"Не удалось получить имя
файла", L"Ошибка", MB_ICONINFORMATION);
                                return 0;
                        }

                        std::wifstream file(fileName, std::ios::binary);
                        std::wstringstream buf;
                        std::wstring file_content;

                        if (file.is_open()) {
                                int textLength = GetWindowTextLength(textEdit);
                                std::wstring text;
                                text.resize(textLength + 1);
                                file.imbue(std::locale("ru_RU.utf8"));
                                GetWindowText(textEdit, &text[0], textLength + 1);

                                buf << file.rdbuf();
                                file.close();
                                file_content = buf.str();

                                SetWindowTextW(textEdit, file_content.c_str());
                        }
                }
                break;
                case ID_32775:  // new tab
                {
                        // Добавление новой вкладки
                                TCITEM tie;
                                tie.mask = TCIF_TEXT;
                                tabIndexCounter++;

                                WCHAR tabText[64];
                                swprintf_s(tabText, L"Tab %d", tabIndexCounter);
```

```
                                tie.pszText = tabText;
                                TabCtrl_InsertItem(tabControl, tabIndexCounter - 1,
&tie); // вставка новой вкладки

                                CreateWindow(
                                        L"RICHEDIT50W",
                                        L"",
                                        WS_BORDER | WS_CHILD | WS_VISIBLE |
WS_VSCROLL | WS_HSCROLL | ES_MULTILINE,
                                        10, 50, 460, 380,
                                        hWnd, (HMENU)(IDC_EDIT_TAB_START +
tabIndexCounter - 1), hInst, nullptr
                                );

                                // Переключение на новую вкладку
                                TabCtrl_SetCurSel(tabControl, tabIndexCounter - 1);
                                SendMessage(hWnd, WM_COMMAND,
MAKEWPARAM(IDC_TABCONTROL, 0), 0);

                        }
                        break;
                        case ID_32778: // Закрыть последнюю вкладку
                        {
                                if (tabIndexCounter > 0)
                                {
                                        int currentTabIndex = tabIndexCounter - 1;
                                        int lastTabIndex = tabIndexCounter - 1;

                                        // Удаляем вкладку и связанное с ней текстовое
поле
                                        TabCtrl_DeleteItem(tabControl, lastTabIndex);
                                        DestroyWindow(GetDlgItem(hWnd,
IDC_EDIT_TAB_START + lastTabIndex));

                                        // Обновляем счетчик вкладок
                                        tabIndexCounter--;

                                        // Если закрыта текущая вкладка, обновляем
текущую выбранную вкладку

                                        if (currentTabIndex >= tabIndexCounter)
```

```
                                {
                                        currentTabIndex = tabIndexCounter - 1;
                                        TabCtrl_SetCurSel(tabControl,
currentTabIndex);
                                        SendMessage(hWnd, WM_COMMAND,
MAKEWPARAM(IDC_TABCONTROL, 0), 0);
                                }

                        }
                }
                break;

                default: return DefWindowProc(hWnd, message, wParam,
lParam);

                }
        }
    }
    break;
    case WM_CLOSE:
        if (MessageBox(hWnd, L"Вы действительно хотите закрыть окно?",
L"Выход", MB_OKCANCEL) == IDOK) {
            DestroyWindow(hWnd);
        }
        return 0;
    break;
    case WM_DESTROY:
        PostQuitMessage(0);
    break;
    default: return DefWindowProc(hWnd, message, wParam, lParam);

    }
    return 0;
}

// Обработчик сообщений для окна "О программе".
INT_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam,
LPARAM lParam)
{
    UNREFERENCED_PARAMETER(lParam);
    switch (message)
```

```
	{
	case WM_INITDIALOG:
		return (INT_PTR)TRUE;

	case WM_COMMAND:
		if (LOWORD(wParam) == IDOK || LOWORD(wParam) ==
IDCANCEL)
		{
			EndDialog(hDlg, LOWORD(wParam));
			return (INT_PTR)TRUE;
		}
		break;
	}
	return (INT_PTR)FALSE;
}
```