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.

Выполнил студент гр.153502 Легоньков Н.В.

Проверил ассистент кафедры информатики Гриценко Н.Ю.

Минск 2023

СОДЕРЖАНИЕ

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 построить приложение для чтения и редактирования текстовых документов с возможностью выделения и копирования текста в буфер обмена.

2 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

Согласно формулировке задачи, были спроектированы следующие функции программы:

– Открытие файла;
– Редактирование текста;
– Копирование выделенного текста в буфер обмена;
– Сохранение отредактированного текста в файл.

1. Открытие файла

Для открытия текстового файла необходимо в меню нажать File-Open и в открывшемся диалоговом окне выбрать нужный файл. Для корректного открытия текстовых файлов с разными кодировками была использована библиотека icu.

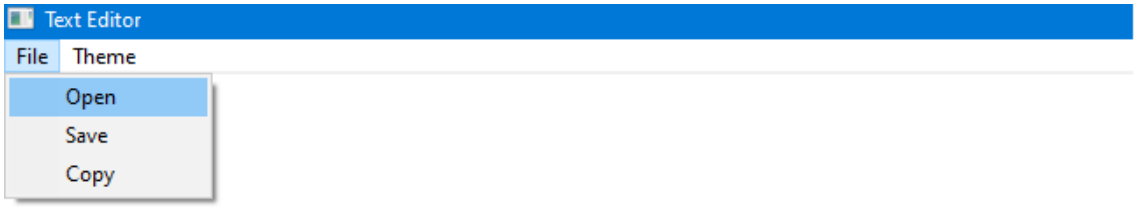

Рисунок 1 – Открытие файла

2. Редактирование текста и копирование выделенного текста в буфер обмена

После открытия файла в окне редактирования появляется текст из него, который можно отредактировать. Для копирования выделенного текста в буфер обмена можно воспользоваться меню: File-Copy.

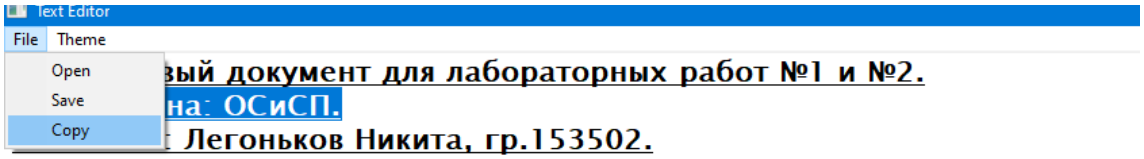

Рисунок 2 – Копирование выделенного текста в буфер обмена

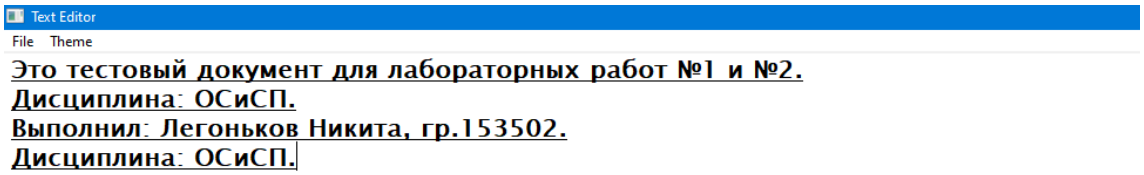

Рисунок 3 – Результат вставки текста после копирования

3. Сохранение отредактированного текста в файл

Для сохранения отредактированного текста в файл необходимо воспользоваться меню: File-Save. Сохранение произойдет в файл, который был открыт вначале.

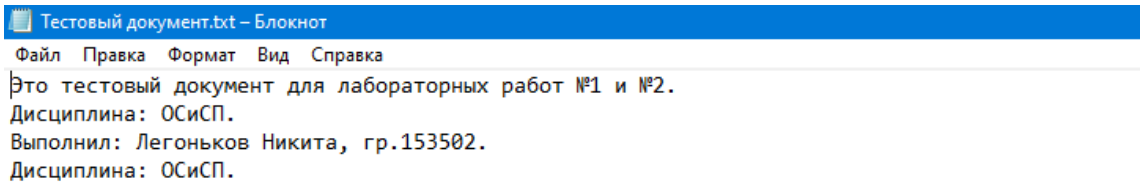

Рисунок 4 – Результаты сохранения

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] Основные сообщения ОС Windows (Win32 API). Программирование в ОС Windows. Лекция 1. – Электронные данные. – Режим доступа: https://www.youtube.com/watch?v=wTArIolxch0

ПРИЛОЖЕНИЕ А

Листинг кода

Файл Lab_1.cpp

```
#include <windows.h>
#include <tchar.h>
#include <commdlg.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <unicode/ucsdet.h>
#include <unicode/ustring.h>
#include <vector>

// Глобальные переменные
HWND hEditControl;  // Дескриптор кастомного окна редактирования
std::wstring filePath;
HFONT g_hFont; // Дескриптор шрифта
COLORREF g_textColor = RGB(0, 0, 0);
COLORREF g_bgColor = RGB(255, 255, 255);

// Идентификаторы команд
#define IDM_FILE_OPEN 1001
#define IDM_FILE_SAVE 1002
#define IDM_COPY 1003
#define IDM_FONT_DIALOG 1004
#define IDM_FONT_SIZE_DIALOG 1005
#define IDM_THEME_SETTINGS 1006
#define IDM_CHANGE_BG_COLOR 1007


// Прототипы функций
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);
void OpenTextFile(HWND hWnd);
std::string DetectFileEncoding(const std::wstring& filePath);
void SaveTextToFile(const std::wstring& filePath, const std::wstring& text);
void ChangeFont();
COLORREF ChooseBackgroundColor(HWND hWnd);
void MainWindAddWidgets(HWND hwnd);

std::wstring GetTextFromEditControl(HWND hEdit)
{
    int textLength = GetWindowTextLengthW(hEdit);
    if (textLength > 0)
    {
        std::wstring text;
        text.resize(textLength + 1);

        GetWindowTextW(hEdit, &text[0], textLength + 1);
        text.pop_back();
        return text;
    }
    return L"";
}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow)
{
    WNDCLASSEX wcex = { sizeof(WNDCLASSEX), CS_HREDRAW | CS_VREDRAW, WndProc, 0, 0,
GetModuleHandle(nullptr), nullptr, nullptr, nullptr, nullptr, _T("TextEditorClass"), nullptr };
    RegisterClassEx(&wcex);
```

```
    HWND hWnd = CreateWindow(_T("TextEditorClass"), _T("Text Editor"), WS_OVERLAPPEDWINDOW,
CW_USEDEFAULT, CW_USEDEFAULT, 800, 600, nullptr, nullptr, hInstance, nullptr);

    if (!hWnd)
    {
        return -1;
    }

    HMENU hMenu = CreateMenu();
    HMENU hFileMenu = CreateMenu();
    HMENU hThemeMenu = CreateMenu();
    AppendMenu(hFileMenu, MF_STRING, IDM_FILE_OPEN, _T("Open"));
    AppendMenu(hFileMenu, MF_STRING, IDM_FILE_SAVE, _T("Save"));
    AppendMenu(hFileMenu, MF_STRING, IDM_COPY, _T("Copy"));
    AppendMenu(hMenu, MF_POPUP, (UINT_PTR)hFileMenu, _T("File"));
    AppendMenu(hThemeMenu, MF_STRING, IDM_CHANGE_BG_COLOR, _T("Select Background Color"));
    AppendMenu(hThemeMenu, MF_STRING, IDM_FONT_DIALOG, _T("Change Font"));
    AppendMenu(hMenu, MF_POPUP, (UINT_PTR)hThemeMenu, _T("Theme"));
    SetMenu(hWnd, hMenu);

    MainWindAddWidgets(hWnd);

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

    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);

    MSG msg;
    while (GetMessage(&msg, nullptr, 0, 0))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    if (g_hFont)
    {
        DeleteObject(g_hFont);
    }
    return static_cast<int>(msg.wParam);
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
    case WM_DESTROY:
        PostQuitMessage(0);
        break;
    case WM_COPY:
        if (OpenClipboard(hWnd))
        {
            EmptyClipboard();
            int textLength = GetWindowTextLength(hEditControl);
            if (textLength > 0)
            {
                HGLOBAL hClipboardData = GlobalAlloc(GMEM_DDESHARE, (textLength + 1) * sizeof(TCHAR));
                if (hClipboardData)
                {
                    LPTSTR pBuffer = static_cast<LPTSTR>(GlobalLock(hClipboardData));
                    if (pBuffer)
```

```
                {
                    GetWindowText(hEditControl, pBuffer, textLength + 1);
                    GlobalUnlock(hClipboardData);
                    SetClipboardData(CF_UNICODETEXT, hClipboardData);
                }
            }
        }
        CloseClipboard();
    }
    break;
case WM_PAINT:
    {
    PAINTSTRUCT ps;
    HDC hdc = BeginPaint(hWnd, &ps);
    HBRUSH hBrush = CreateSolidBrush(g_bgColor);
    FillRect(hdc, &ps.rcPaint, hBrush);
    DeleteObject(hBrush);
    EndPaint(hWnd, &ps);
    }
    break;
case WM_CTLCOLOREDIT:
{
    HDC hdcEdit = (HDC)wParam;
    SetBkColor(hdcEdit, g_bgColor);
    return (LRESULT)CreateSolidBrush(g_bgColor);
}
case WM_COMMAND:
    switch (LOWORD(wParam))
    {
    case IDM_FILE_OPEN:
        OpenTextFile(hWnd);
        break;
    case IDM_FILE_SAVE:
        SaveTextToFile(filePath, GetTextFromEditControl(hEditControl));
        break;
    case IDM_COPY:
        SendMessage(hEditControl, WM_COPY, 0, 0);
        break;
    case IDM_FONT_DIALOG:
        ChangeFont();
        break;
    case IDM_CHANGE_BG_COLOR:
    {
        COLORREF newColor = ChooseBackgroundColor(hWnd);
        if (newColor != g_bgColor)
        {
            g_bgColor = newColor;
            InvalidateRect(hWnd, nullptr, TRUE);
        }
    }
    break;
    }
    break;
case WM_SIZE:
{
    int newWidth = LOWORD(lParam);
    int newHeight = HIWORD(lParam);

    SetWindowPos(hEditControl, NULL, 1, 0, newWidth - 20, newHeight - 20, SWP_NOMOVE |
SWP_NOZORDER);
}
break;
```

```
    default:
        return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}

std::string DetectFileEncoding(const std::wstring& filePath)
{
    UErrorCode status = U_ZERO_ERROR;
    UCharsetDetector* detector = ucsdet_open(&status);

    if (U_SUCCESS(status) && detector)
    {
        FILE* file = nullptr;
        if (_wfopen_s(&file, filePath.c_str(), L"rb") == 0 && file)
        {
            fseek(file, 0, SEEK_END);
            long fileSize = ftell(file);
            fseek(file, 0, SEEK_SET);

            char* buffer = new char[fileSize];
            fread(buffer, 1, fileSize, file);
            fclose(file);

            ucsdet_setText(detector, buffer, fileSize, &status);
            const UCharsetMatch* match = ucsdet_detect(detector, &status);

            if (U_SUCCESS(status) && match)
            {
                const char* encoding = ucsdet_getName(match, &status);
                delete[] buffer;
                ucsdet_close(detector);
                return encoding ? encoding : "Unknown";
            }
            delete[] buffer;
        }
        ucsdet_close(detector);
    }
    return "Unknown";
}

void OpenTextFile(HWND hWnd)
{
    OPENFILENAME ofn;
    WCHAR szFile[260] = { 0 };

    ZeroMemory(&ofn, sizeof(ofn));
    ofn.lStructSize = sizeof(ofn);
    ofn.hwndOwner = hWnd;
    ofn.lpstrFile = szFile;
    ofn.nMaxFile = sizeof(szFile);
    ofn.lpstrFilter = _T("Text Files (*.txt)\0*.txt\0All Files (*.*)\0*.*\0");
    ofn.nFilterIndex = 1;
    ofn.lpstrFileTitle = nullptr;
    ofn.nMaxFileTitle = 0;
    ofn.lpstrInitialDir = nullptr;
    ofn.Flags = OFN_PATHMUSTEXIST | OFN_FILEMUSTEXIST;

    if (GetOpenFileName(&ofn))
    {
        filePath = ofn.lpstrFile;
```

```
std::wstring fileName = filePath.substr(filePath.find_last_of(L"\\") + 1);
std::wstring windowTitle = L"Text Editor - " + fileName;
SetWindowText(hWnd, windowTitle.c_str());

std::wstring filePathW = filePath;
std::string encoding = DetectFileEncoding(filePathW);

if (encoding == "UTF-8")
{
    std::ifstream file(filePathW, std::ios::binary);
    if (file.is_open())
    {
        std::stringstream buffer;
        buffer << file.rdbuf();
        std::string fileContents = buffer.str();

        // Преобразование из UTF-8 в UTF-16LE
        int utf16Length = MultiByteToWideChar(CP_UTF8, 0, fileContents.c_str(), -1, nullptr, 0);
        if (utf16Length > 0)
        {
            wchar_t* utf16Buffer = new wchar_t[utf16Length];
            MultiByteToWideChar(CP_UTF8, 0, fileContents.c_str(), -1, utf16Buffer, utf16Length);
            SetWindowTextW(hEditControl, utf16Buffer);
            delete[] utf16Buffer;
        }
    }
}
else if (encoding == "UTF-16LE")
{
    // Обработка UTF-16LE
    FILE* file = nullptr;
    if (_wfopen_s(&file, filePathW.c_str(), L"rb") == 0 && file)
    {
        fseek(file, 0, SEEK_END);
        long fileSize = ftell(file);
        fseek(file, 0, SEEK_SET);

        wchar_t* wideBuffer = new wchar_t[fileSize / sizeof(wchar_t) + 1];
        fread(wideBuffer, sizeof(wchar_t), fileSize / sizeof(wchar_t), file);
        wideBuffer[fileSize / sizeof(wchar_t)] = L'\0';

        SetWindowTextW(hEditControl, wideBuffer);
        delete[] wideBuffer;

        fclose(file);
    }
}
else if (encoding == "UTF-16BE")
{
    FILE* file = nullptr;
    if (_wfopen_s(&file, filePathW.c_str(), L"rb") == 0 && file)
    {
        fseek(file, 0, SEEK_END);
        long fileSize = ftell(file);
        fseek(file, 0, SEEK_SET);

        // Преобразование UTF-16BE в UTF-16LE
        wchar_t* wideBuffer = new wchar_t[fileSize / sizeof(wchar_t) + 1];
        fread(wideBuffer, sizeof(wchar_t), fileSize / sizeof(wchar_t), file);

        // Смена байтов местами для преобразования
        for (int i = 0; i < fileSize / sizeof(wchar_t); i++)
```

```
            {
                wideBuffer[i] = ((wideBuffer[i] >> 8) & 0xFF) | ((wideBuffer[i] << 8) & 0xFF00);
            }

            wideBuffer[fileSize / sizeof(wchar_t)] = L'\0';
            SetWindowTextW(hEditControl, wideBuffer);
            delete[] wideBuffer;

            fclose(file);
        }
    }
    else
    {
        MessageBox(hWnd, _T("Unsupported encoding"), _T("Error"), MB_OK | MB_ICONERROR);
    }
  }
}

void SaveTextToFile(const std::wstring& filePath, const std::wstring& text)
{
  HANDLE hFile = CreateFile(filePath.c_str(), GENERIC_WRITE, 0, nullptr, CREATE_ALWAYS,
FILE_ATTRIBUTE_NORMAL, nullptr);

  if (hFile != INVALID_HANDLE_VALUE)
  {
    DWORD bytesWritten = 0;
    WriteFile(hFile, text.c_str(), static_cast<DWORD>(text.size() * sizeof(wchar_t)), &bytesWritten, nullptr);
    CloseHandle(hFile);
  }
  else
  {
    MessageBox(nullptr, L"Unable to open the file for writing.", L"Error", MB_OK | MB_ICONERROR);
  }
}

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);
  }
}

COLORREF ChooseBackgroundColor(HWND hWnd)
{
  CHOOSECOLOR cc = { sizeof(CHOOSECOLOR) };
  static COLORREF customColors[16];

  cc.hwndOwner = hWnd;
  cc.lpCustColors = customColors;
```

```
    cc.rgbResult = g_bgColor;

    cc.Flags = CC_FULLOPEN | CC_RGBINIT;

    // Открываем диалог выбора цвета
    if (ChooseColor(&cc))
    {
        return cc.rgbResult;
    }
    return g_bgColor;
}

void MainWindAddWidgets(HWND hwnd)
{
    RECT windowRect;
    GetWindowRect(hwnd, &windowRect);

    int windowWidth = windowRect.right - windowRect.left;
    int windowHeight = windowRect.bottom - windowRect.top;

    hEditControl = CreateWindowW(L"Edit", L"", WS_CHILD | WS_VISIBLE | ES_LEFT |
        WS_VSCROLL | ES_MULTILINE | ES_AUTOVSCROLL,
        0, 0, windowWidth, windowHeight, hwnd, NULL, NULL, NULL);

    HBRUSH hEditBgBrush = CreateSolidBrush(g_bgColor);
    SendMessage(hEditControl, WM_CTLCOLOREDIT, (WPARAM)GetDC(hEditControl),
(LPARAM)hEditBgBrush);
}
}
```