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 Макаренко А.И.

Проверил ассистент кафедры информатики Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 ФОРМУЛИРОВКА ЗАДАЧИ 3](file:///C:\Users\alyona\Downloads\Telegram%20Desktop\1%20ЛР.docx#_Toc145669228)

[2 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ 4](file:///C:\Users\alyona\Downloads\Telegram%20Desktop\1%20ЛР.docx#_Toc145669229)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 6](file:///C:\Users\alyona\Downloads\Telegram%20Desktop\1%20ЛР.docx#_Toc145669230)

[ПРИЛОЖЕНИЕ А 7](file:///C:\Users\alyona\Downloads\Telegram%20Desktop\1%20ЛР.docx#_Toc145669231)

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;

– Копирование выделенного текста в буфер обмена;

– Сохранение файла.

1. **Открытие файла**

Для того, чтобы открыть файл, необходимо нажать на кнопку File в меню редактора, далее на кнопку Open, затем в диалоговом окне выбрать нужный файл.

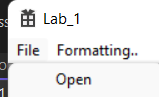


Рисунок 1 – Кнопка для выбора файла

1. **Копирование в буфер обмена**

Чтобы скопировать выделенный текст в буфер обмена, необходимо нажать Ctrl + C или нажать на кнопку copy, расположенную в правой части экрана.

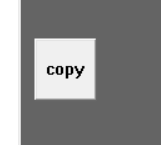


Рисунок 2 – Кнопка copy для копирования выделенного текста

1. **Сохранение файла**

Для сохранения файла необходимо нажать на кнопку File в меню редактора, далее на кнопку Save, затем в диалоговом окне выбрать нужную директорию для сохранения.

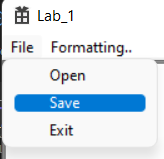


Рисунок 3 – Кнопка Save для сохранения

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Основы программирования для Win32 API [Электронный ресурс]. – Электронные данные. – Режим доступа: https://dims.karelia.ru/win32/

[2] Build desktop Windows apps using the Win32 API [Электронный ресурс]. – Электронные данные. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/

[3] Windows API Guide: OPENFILENAME Structure [Электронный ресурс]. – Электронные данные. – Режим доступа: http://www.jasinskionline.com/windowsapi/ref/o/openfilename.html

[4] Win32 File API – CreateFile function [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.easefilter.com/kb/fileapi-createfile.htm

ПРИЛОЖЕНИЕ А

Листинг кода

Файл Lab\_1.cpp

#include "framework.h"

#include "Lab\_1.h"

#define MAX\_LOADSTRING 100

HINSTANCE hInst; // current instance

WCHAR szTitle[MAX\_LOADSTRING]; // The title bar text

WCHAR szWindowClass[MAX\_LOADSTRING]; // the main window class name

ATOM MyRegisterClass(HINSTANCE hInstance);

BOOL InitInstance(HINSTANCE, int);

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

HWND CreateRichEdit(HWND hwndOwner, int x, int y, int width, int height, HINSTANCE hinst);

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\_LAB1, szWindowClass, MAX\_LOADSTRING);

MyRegisterClass(hInstance);

if (!InitInstance (hInstance, nCmdShow))

{

return FALSE;

}

HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC\_LAB1));

MSG msg;

// Main message loop:

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;

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\_LAB1));

wcex.hCursor = LoadCursor(nullptr, IDC\_ARROW);

wcex.hbrBackground = (HBRUSH)(COLOR\_WINDOW+1);

wcex.lpszMenuName = MAKEINTRESOURCEW(IDC\_LAB1);

wcex.lpszClassName = szWindowClass;

wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI\_SMALL));

return RegisterClassExW(&wcex);

}

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)

{

hInst = hInstance; // Store instance handle in our global variable

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);

return TRUE;

}

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

{

static HWND hREdit; // handle for the edit

static OPENFILENAME ofn; // structure for the file dialog

int lenEdit;

TCHAR textBuf[1024]{};

size\_t memSize;

HGLOBAL hGlob;

static COLORREF acrCustClr[16];

static DWORD bgColorCurr;

switch (message)

{

case WM\_CREATE:

{

hREdit = CreateRichEdit(hWnd, 0, 0, 1000, 500, hInst);

//hEdit = CreateWindowEx(WS\_EX\_CLIENTEDGE, L"EDIT", L"Enter text right here ", WS\_CHILD | WS\_HSCROLL | WS\_VISIBLE | WS\_VSCROLL | ES\_MULTILINE | ES\_AUTOVSCROLL | ES\_AUTOHSCROLL | ES\_WANTRETURN,

// 0, 0, 1000, 500, hWnd, NULL, hInst, NULL);

if (hREdit == NULL)

{

MessageBox(hWnd, L"Edit field failed to create", L"Error", MB\_ICONERROR);

return -1;

}

HWND btnCopy = CreateWindowW(L"Button", L"copy", WS\_VISIBLE | WS\_CHILD | BS\_PUSHBUTTON, 1010, 150, 50, 50, hWnd, (HMENU)IDM\_COPY, hInst, NULL);

if (btnCopy == NULL)

{

MessageBox(hWnd, L"Button failed to create", L"Error", MB\_ICONERROR);

return -1;

}

}

break;

case WM\_COMMAND:

{

int wmId = LOWORD(wParam);

switch (wmId)

{

case IDM\_OPEN:

{

WCHAR szFileName[MAX\_PATH] = L"";

ZeroMemory(&ofn, sizeof(OPENFILENAME));

ofn.lStructSize = sizeof(OPENFILENAME);

ofn.hwndOwner = hWnd;

ofn.lpstrFilter = L"Text Files (\*.txt)\0\*.txt\0All Files (\*.\*)\0\*.\*\0";

ofn.lpstrFile = szFileName;

ofn.nMaxFile = MAX\_PATH;

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

if (GetOpenFileName(&ofn))

{

HANDLE hFile = CreateFile(szFileName, GENERIC\_READ, 0, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (hFile != INVALID\_HANDLE\_VALUE)

{

DWORD dwFileSize = GetFileSize(hFile, NULL);

if (dwFileSize != INVALID\_FILE\_SIZE)

{

LPSTR lpFileText = (LPSTR)GlobalAlloc(GPTR, (dwFileSize + 1) \* sizeof(WCHAR)); //+1 means extra byte for null termination

if (lpFileText != NULL)

{

DWORD dwRead;

if (ReadFile(hFile, lpFileText, dwFileSize, &dwRead, NULL))

{

lpFileText[dwFileSize] = '\0'; // we add null to pass it to the other parts as a string

int wideCharCount = MultiByteToWideChar(CP\_UTF8, 0, lpFileText, -1, NULL, 0); //recalculating buffer size to change encoding

if (wideCharCount > 0)

{

LPWSTR lpWideText = (LPWSTR)GlobalAlloc(GPTR, wideCharCount \* sizeof(WCHAR));

if (lpWideText != NULL)

{

MultiByteToWideChar(CP\_UTF8, 0, lpFileText, -1, lpWideText, wideCharCount);

SetWindowTextW(hREdit, lpWideText);

GlobalFree(lpWideText);

}

}

}

else

{

MessageBox(hWnd, L"Failed to read the file", L"Error", MB\_ICONERROR);

}

GlobalFree(lpFileText);

}

else

{

MessageBox(hWnd, L"Failed to allocate memory", L"Error", MB\_ICONERROR);

}

}

CloseHandle(hFile);

}

else

{

MessageBox(hWnd, L"Failed to open the file", L"Error", MB\_ICONERROR);

}

}

}

break;

case IDM\_SAVE:

{

WCHAR szFileName[MAX\_PATH] = L"";

ZeroMemory(&ofn, sizeof(OPENFILENAME));

ofn.lStructSize = sizeof(OPENFILENAME);

ofn.hwndOwner = hWnd;

ofn.lpstrFilter = L"Text Files (\*.txt)\0\*.txt\0All Files (\*.\*)\0\*.\*\0";

ofn.lpstrFile = szFileName;

ofn.nMaxFile = MAX\_PATH;

ofn.Flags = OFN\_OVERWRITEPROMPT;

if (GetSaveFileName(&ofn))

{

HANDLE hFile = CreateFile(szFileName, GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (hFile != INVALID\_HANDLE\_VALUE)

{

int len = GetWindowTextLength(hREdit);

LPWSTR lpFileText = (LPWSTR)GlobalAlloc(GPTR, (len + 1) \* sizeof(WCHAR));

if (lpFileText != NULL)

{

GetWindowText(hREdit, lpFileText, len + 1);

DWORD dwWritten;

int utf8Len = WideCharToMultiByte(CP\_UTF8, 0, lpFileText, -1, NULL, 0, NULL, NULL);

if (utf8Len > 0) {

LPSTR lpFileTextUtf8 = (LPSTR)GlobalAlloc(GPTR, utf8Len);

if (lpFileTextUtf8 != NULL) {

WideCharToMultiByte(CP\_UTF8, 0, lpFileText, -1, lpFileTextUtf8, utf8Len, NULL, NULL);

if (WriteFile(hFile, lpFileTextUtf8, utf8Len - 1, &dwWritten, NULL))

{

MessageBox(hWnd, L"Saved successfully", L"Success", MB\_ICONINFORMATION);

}

else

{

MessageBox(hWnd, L"Failed to write the file", L"Error", MB\_ICONERROR);

}

GlobalFree(lpFileTextUtf8);

}

}

GlobalFree(lpFileText);

}

CloseHandle(hFile);

}

else

{

MessageBox(hWnd, L"Failed to create the file", L"Error", MB\_ICONERROR);

}

}

}

break;

case IDM\_EXIT:

DestroyWindow(hWnd);

break;

case IDM\_COPY:

{;

CHARRANGE cr;

SendMessage(hREdit, EM\_EXGETSEL, 0, (LPARAM)&cr);

if (cr.cpMin != cr.cpMax) {

SendMessage(hREdit, WM\_COPY, 0, 0);

}

}

break;

case IDM\_COLOR:

{

CHOOSECOLOR cc{0};

cc.lStructSize = sizeof(cc);

cc.hwndOwner = hWnd;

cc.lpCustColors = (LPDWORD)acrCustClr;

cc.rgbResult = bgColorCurr;

cc.Flags = CC\_FULLOPEN | CC\_RGBINIT;

if (ChooseColor(&cc) == TRUE)

{

bgColorCurr = cc.rgbResult;

SendMessage(hREdit, EM\_SETBKGNDCOLOR, FALSE, (LPARAM)bgColorCurr);

}

}

break;

case IDM\_FONT:

{

HDC hdc = (HDC)hREdit;

CHOOSEFONT cf;

static LOGFONT lf;

static DWORD rgbCurrent;

HFONT hFont, hfontPrev;

DWORD rgbPrev;

ZeroMemory(&cf, sizeof(cf));

cf.lStructSize = sizeof(cf);

cf.hwndOwner = hWnd;

cf.lpLogFont = &lf;

cf.rgbColors = rgbCurrent;

cf.Flags = CF\_SCREENFONTS | CF\_EFFECTS;

if (ChooseFont(&cf) == TRUE)

{

hFont = CreateFontIndirect(cf.lpLogFont);

hfontPrev = (HFONT)SelectObject(hdc, hFont);

rgbCurrent = cf.rgbColors;

rgbPrev = SetTextColor(hdc, rgbCurrent);

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

COLORREF color = cf.rgbColors;

CHARFORMAT chf;

chf.cbSize = sizeof(chf);

chf.dwEffects = CFE\_BOLD;

chf.crTextColor = color;

chf.dwMask = CFM\_BOLD | CFM\_COLOR;

SendMessage(hREdit, EM\_SETCHARFORMAT, SCF\_DEFAULT, (LPARAM)&chf);

}

}

break;

default:

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

}

}

break;

case WM\_PAINT:

{

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hWnd, &ps);

FillRect(hdc, &ps.rcPaint, (HBRUSH)(COLOR\_WINDOW + 2));

EndPaint(hWnd, &ps);

}

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

case WM\_CLOSE:

if (MessageBox(hWnd, L"Really quit?", L"My application", MB\_OKCANCEL) == IDOK)

{

DestroyWindow(hWnd);

}

break;

default:

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

}

return 0;

}

HWND CreateRichEdit(HWND hwndOwner, int x, int y, int width, int height, HINSTANCE hinst)

{

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

HWND hwndEdit = CreateWindowEx(0, MSFTEDIT\_CLASS, TEXT("Type here"),

WS\_CHILD | WS\_HSCROLL | WS\_VISIBLE | WS\_VSCROLL | ES\_MULTILINE | ES\_AUTOVSCROLL | ES\_AUTOHSCROLL | ES\_WANTRETURN | WS\_BORDER | WS\_TABSTOP,

x, y, width, height,

hwndOwner, NULL, hinst, NULL);

return hwndEdit;

}