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 Win32 API. Оконное приложение Win32 с минимальной функциональной достаточностью. Обработка основных оконных сообщений

Выполнил: студент группы 153503

Киселёва Елизавета Андреевна

Проверил: Гриценко Никита Юрьевич

Минск 2023

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc146728177)

[2 Краткие теоретические сведения 4](#_Toc146728178)

[2 Результаты выполнения лабораторной работы 6](#_Toc146728179)

[Выводы 9](#_Toc146728180)

[Список использованных источников 10](#_Toc146728181)

[Приложение А (обязательное) Листинг кода 11](#_Toc146728182)

## 1 ПОСТАНОВКА ЗАДАЧИ

Целью выполнения лабораторной работы является создание оконного приложения на Win32 API с минимальной функциональной достаточностью, позволяющее отработать базовые навыки написания программы на Win32 API, использования виджетов и обработки оконных сообщений. Создать проект текстового редактора в выбранной среде разработки (Visual Studio) с минимальным функционалом – ввод, редактирование, удаление текста, сохранение и загрузка файлов, на котором отработать вышеперечисленные навыки. Внутри проекта реализовать функцию обработки оконных сообщений, которая будет обрабатывать следующие базовые сообщения: WM\_CREATE, WM\_DESTROY, WM\_COMMAND, WM\_PAINT.

## 2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Win32 – это набор API (Application Programming Interface), предоставляемых операционной системой Microsoft Windows для разработки приложений под эту операционную систему. Термин "Win32" обычно используется для обозначения API, доступного на 32-битных версиях Windows, начиная с Windows 95 и до более поздних версий Windows, включая Windows 98, Windows NT, Windows 2000, Windows XP, и так далее.

API – это набор функций и процедур, предоставляемых операционной системой, с помощью которых приложения могут взаимодействовать с ресурсами и функциональностью операционной системы.

Для выполнения данной лабораторной работы, были использованы следующие теоретические сведения:

­– Win32 API предоставляет программистам средства для создания оконных приложений под Windows, управления окнами, файлами, вводом и выводом данных, сетью, графикой и многими другими аспектами операционной системы. Win32 API также включает в себя множество библиотек и заголовочных файлов, которые программисты могут использовать при разработке Windows-приложений на языках программирования, таких как C и C++. В данной лабораторной работе использовался для создания графического интерфейса текстового редактора.

– Для сохранения и загрузки текстовых файлов использовались стандартные функции работы с файлами, такие как CreateFile, ReadFile, WriteFile и CloseHandle. Функция CloseHandle используется для закрытия дескрипторов объектов в операционной системе Windows, что важно по причинам освобождения ресурсов, предотвращения утечек ресурсов и доступа к файлу другим процессам.

– Для создания графического интерфейса приложения были использованы стандартные элементы управления Windows, такие как окна, кнопки и текстовые поля. Эти элементы были созданы и настроены с использованием Win32 API функций. Также учитывались стандартные правила удобного для пользователя интерфейса.

– В Windows-приложениях взаимодействие с пользователем осуществляется через обработку событий и сообщений. В лабораторной работе используются функции обработки сообщений, такие, как SoftwareMainProcedure, отвечающая за обработку событий, происходящих в приложении: нажатия кнопок и меню.

– Для работы с окнами и элементами управления использовалась функция CreateWindow, которая используется для создания графических элементов (окон, кнопок, текстовых полей и других элементов управления) в оконных приложениях Windows. Свойства созданных элементов задаются с помощью параметров, передаваемых в функцию (имя класса окна, заголовок окна или текст в элементе управления, стили элемента управления, координаты положения на экране, ширина и высота, дескриптор родительского окна, в котором будет расположен создаваемый элемент, идентификатор меню, связанный с элементом, дескриптор экземпляра приложения, дополнительные пользовательские данные)

– Для создания меню и подменю, а также обработки команд меню использовалась функция CreateMenu. Функция CreateMenu возвращает дескриптор (HANDLE) созданного меню. Этот дескриптор можно использовать для добавления пунктов меню и подменю с помощью функций, таких как AppendMenu, AppendSubMenu и других.

## 2 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

В ходе выполнения лабораторной работы был разработан текстовый редактор с возможностью сохранения и загрузки файлов, а также минимальной кастомизацией интерфейса. Текстовый редактор позволяет пользователю изменять текст, полностью очищать поле для ввода текста, сохранять текстовый файл, загружать уже удаленный, а также менять цвет надписи внутри градиентного поля. Также пользователь может видеть название приложения и его иконку (Рисунок 1).

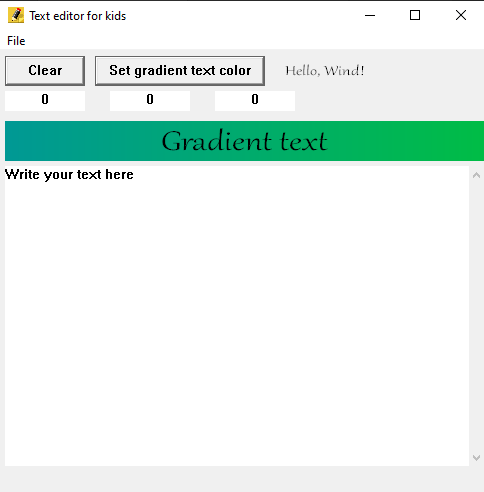


Рисунок 1 - Главное окно

Пользователь может вписать свой текст в текстовое поле (Рисунок 2).

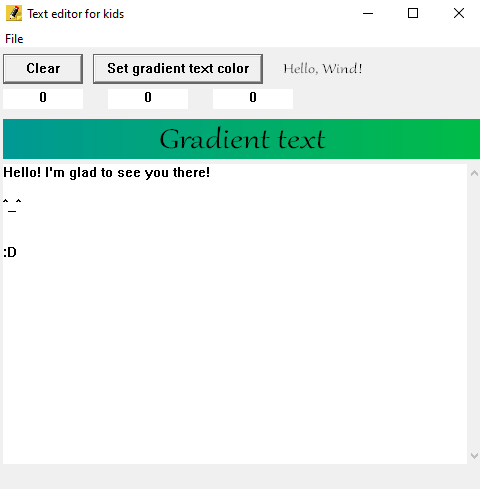


Рисунок 2 ─ Свой текст

Пользователь может сохранить файл. (Рисунок 3).

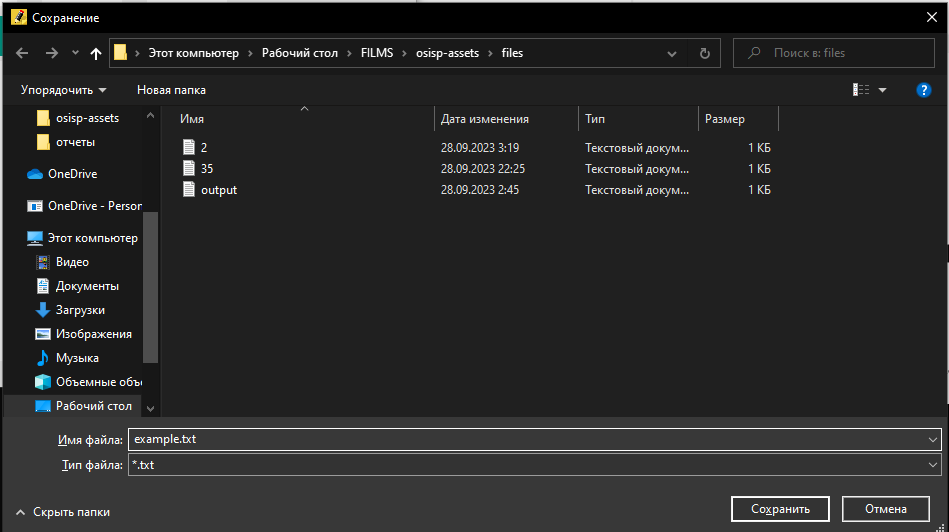


Рисунок 3 – Сохранение файла

Пользователь может загрузить уже существующий текстовый файл (Рисунок 4).

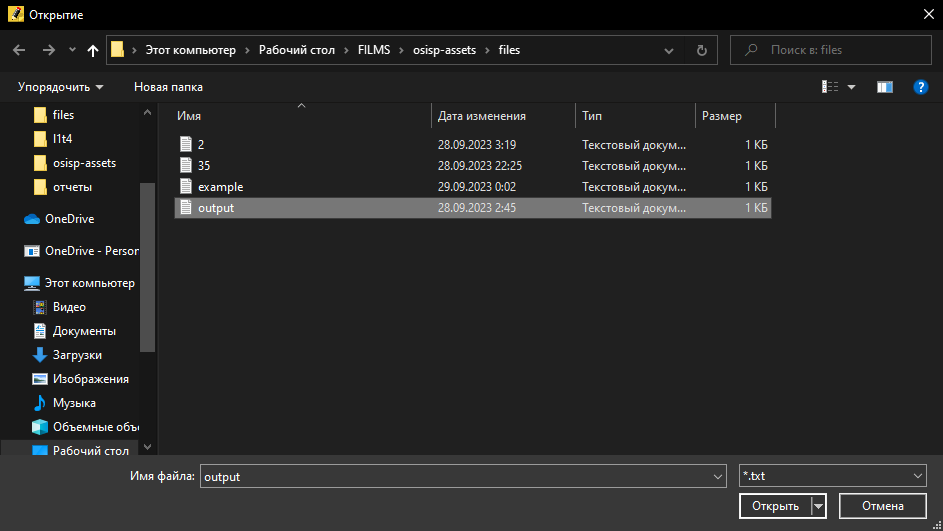


Рисунок 4 – Загрузка файла

Пользователь может очистить поле для ввода текста по нажатию на Clear (Рисунок 5).

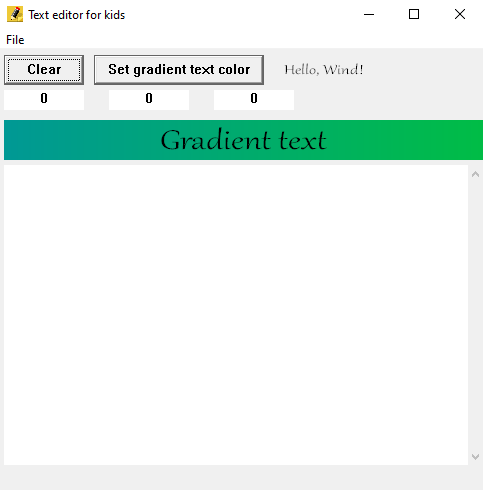


Рисунок 5 – Очистка поля ввода текста

## ВЫВОДЫ

В ходе выполнения данной лабораторной работы был разработан текстовый редактор с возможностью кастомизации интерфейса с использованием Win32 API. Проект включает в себя создание графического окна приложения, обработку оконных сообщений, пользовательский интерфейс с текстовым полем, кнопками и меню, работу с файлами, обновление интерфейса и визуализацию данных. Результатом стал рабочий текстовый редактор, написанный на языке C++ с использованием Win32 API, имеющий простое управление даже для начинающего пользователя и способный как выполнять функции текстового редактора, так и порадовать глаз пользователя привлекательным внешним видом.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Щупак Ю. Win32 API. Разработка приложений для Windows. ─ СПб: Питер, 2008. ─ 592 с.: ип.

[2] Programming reference for the Win32 API [Электронный ресурс]. ─ Режим доступа: https://learn.microsoft.com/en-us/windows/win32/api/ ─ Дата доступа 26.09.2023

## ПРИЛОЖЕНИЕ А

## (обязательное)

## Листинг кода

**SoftwareMain.cpp**

#include <Windows.h>

#include <string>

#include "SoftwareColors.h"

#include "SoftwareDefinitions.h"

#include "resource.h"

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

fontRectangle = CreateFontA(

60, 20, 0, 0, FW\_MEDIUM,

FALSE, FALSE, FALSE, DEFAULT\_CHARSET,

OUT\_OUTLINE\_PRECIS, CLIP\_DEFAULT\_PRECIS, ANTIALIASED\_QUALITY,

FF\_DECORATIVE, "SpecialFont"

);

fontStatic = CreateFontA(

30, 10, 0, 0, FW\_MEDIUM,

FALSE, FALSE, FALSE, DEFAULT\_CHARSET,

OUT\_OUTLINE\_PRECIS, CLIP\_DEFAULT\_PRECIS, ANTIALIASED\_QUALITY,

FF\_DECORATIVE, "SpecialStaticFont"

);

// Создание класса окна WNDCLASS и задание его параметров

WNDCLASS SoftwareMainClass = NewWindowClass((HBRUSH)COLOR\_WINDOW, LoadCursor(NULL, IDC\_HAND), hInst,

LoadIcon(hInst, MAKEINTRESOURCE(IDI\_ICON1)), L"MainWndClass", SoftwareMainProcedure);

// Регистрация класса окна

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

MSG SoftwareMainMessage = { 0 };

// параметр WS\_OVERLAPPEDWINDOW добавляет кнопки размер, свернуть, закрыть на окно + перемещение окна + стандартные бордюр и рамка + системное меню

CreateWindow(L"MainWndClass", L"Text editor for kids", WS\_OVERLAPPEDWINDOW | WS\_VISIBLE, 100, 100, 500, 500, 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 OnClearField:

SetWindowTextA(hEditControl, "");

break;

case OnReadColor:

colorR = GetDlgItemInt(hWnd, DigIndexColorR, FALSE, false);

colorG = GetDlgItemInt(hWnd, DigIndexColorG, FALSE, false);

colorB = GetDlgItemInt(hWnd, DigIndexColorB, FALSE, false);

//brushRectangle = CreateSolidBrush(RGB(colorR, colorG, colorB));

fontColor = RGB(colorR, colorG, colorB);

RedrawWindow(hWnd, NULL, NULL, RDW\_UPDATENOW | RDW\_INVALIDATE);

break;

case OnSaveFile:

if (GetSaveFileNameA(&ofn)) { SaveData(filename); }

break;

case OnLoadFile:

if (GetOpenFileNameA(&ofn)) { LoadData(filename); }

break;

case OnExitSoftware:

PostQuitMessage(0);

break;

default:

break;

}

break;

case WM\_PAINT:

BeginPaint(hWnd, &ps);

//FillRect(ps.hdc, &windowRectangle, brushRectangle);

GradientRect(ps.hdc, &windowRectangle, Color(0,148,153), Color(0,184,74));

SetBkMode(ps.hdc, TRANSPARENT);

SetTextColor(ps.hdc, fontColor);

SelectObject(ps.hdc, fontRectangle);

// DT\_NOCLIP - если текст не влезет в прямоугольник, то выйдет за его пределы и будет виден

DrawTextA(ps.hdc, "Gradient text", 15, &windowRectangle, DT\_SINGLELINE | DT\_CENTER | DT\_VCENTER | DT\_NOCLIP);

EndPaint(hWnd, &ps);

break;

case WM\_CREATE:

MainWndAddMenus(hWnd);

MainWndAddWidgets(hWnd);

SetOpenFileParams(hWnd);

SendMessageA(hStaticControl, WM\_SETFONT, (WPARAM)fontStatic, TRUE);

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

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

}

}

void MainWndAddMenus(HWND hWnd) {

HMENU RootMenu = CreateMenu();

HMENU SubMenu = CreateMenu();

HMENU SubActionMenu = CreateMenu();

AppendMenu(SubMenu, MF\_STRING, OnClearField, L"Clear");

AppendMenu(SubMenu, MF\_SEPARATOR, NULL, NULL);

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

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

AppendMenu(SubMenu, MF\_SEPARATOR, NULL, NULL);

AppendMenu(SubMenu, MF\_STRING, OnExitSoftware, L"Exit");

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

SetMenu(hWnd, RootMenu);

}

void MainWndAddWidgets(HWND hWnd) {

// WS\_VISIBLE - элемент видим по дефолту, WS\_CHILD - элемент является дочерним от базового окна(наодится внутри этого окна)

hStaticControl = CreateWindowA("static", "Hello, Wind!", WS\_VISIBLE | WS\_CHILD | ES\_CENTER, 275, 5, 100, 30, hWnd, NULL, NULL, NULL);

windowRectangle = { 5 + 480, 70, 5, 110 };

hEditControl = CreateWindowA("edit", "Write your text here", WS\_VISIBLE | WS\_CHILD | ES\_MULTILINE | WS\_VSCROLL, 5, 115, 480, 300, hWnd, NULL, NULL, NULL);

CreateWindowA("edit", "0", WS\_VISIBLE | WS\_CHILD | ES\_CENTER | ES\_NUMBER, 5, 40, 80, 20, hWnd, (HMENU)DigIndexColorR, NULL, NULL);

CreateWindowA("edit", "0", WS\_VISIBLE | WS\_CHILD | ES\_CENTER | ES\_NUMBER, 110, 40, 80, 20, hWnd, (HMENU)DigIndexColorG, NULL, NULL);

CreateWindowA("edit", "0", WS\_VISIBLE | WS\_CHILD | ES\_CENTER | ES\_NUMBER, 215, 40, 80, 20, hWnd, (HMENU)DigIndexColorB, NULL, NULL);

CreateWindowA("button", "Clear", WS\_VISIBLE | WS\_CHILD | ES\_CENTER, 5, 5, 80, 30, hWnd, (HMENU)OnClearField, NULL, NULL);

CreateWindowA("button", "Set gradient text color", WS\_VISIBLE | WS\_CHILD | ES\_CENTER, 95, 5, 170, 30, hWnd, (HMENU)OnReadColor, NULL, NULL);

}

void SaveData(LPCSTR path) {

HANDLE FileToSave = CreateFileA(

path, // Путь к файлу

GENERIC\_WRITE, // Режим доступа (запись)

0, // Флаги и атрибуты файла

NULL, // Дескриптор безопасности (не используется)

CREATE\_ALWAYS, // Действие при создании: создать новый файл или перезаписать существующий

FILE\_ATTRIBUTE\_NORMAL, // Атрибуты файла (обычный файл)

NULL // Шаблон файла (не используется)

);

int saveLenth = GetWindowTextLength(hEditControl) + 1;

// буфер для хранения текста

char\* data = new char[saveLenth];

// Получен текст из элемента управления и сохранен в буфере

saveLenth = GetWindowTextA(hEditControl, data, saveLenth);

// Переменная для отслеживания количества записанных байтов

DWORD bytesIterated;

// Записываем содержимое буфера data в файл

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

// Закрываем дескриптор файла

CloseHandle(FileToSave);

delete[] data;

}

void LoadData(LPCSTR path) {

HANDLE FileToLoad = CreateFileA(

path,

GENERIC\_READ, // Режим доступа (чтение)

0,

NULL,

OPEN\_EXISTING, // Открыть существующий

FILE\_ATTRIBUTE\_NORMAL,

NULL);

DWORD bytesIterated;

ReadFile(FileToLoad, Buffer, TextBufferSize, &bytesIterated, NULL);

SetWindowTextA(hEditControl, Buffer); // Считываем в буфер определенное кол-во символов и выводим их в поле edit

CloseHandle(FileToLoad);

}

void SetOpenFileParams(HWND hWnd) {

// Обнуляем структуру ofn (OPENFILENAME) перед использованием

ZeroMemory(&ofn, sizeof(ofn));

// Устанавливаем размер структуры OPENFILENAME

ofn.lStructSize = sizeof(ofn);

// Устанавливаем окно-владелец для диалогового окна выбора файла

ofn.hwndOwner = hWnd;

// Устанавливаем указатель на строку, в которой будет сохранено имя выбранного файла

ofn.lpstrFile = filename;

// Устанавливаем максимальную длину имени файла, которую может принять указатель lpstrFile

ofn.nMaxFile = sizeof(filename);

// Устанавливаем фильтр файлов для диалогового окна (только .txt файлы)

ofn.lpstrFilter = "\*.txt";

// Устанавливаем указатель на строку, в которой будет сохранено имя выбранного файла без расширения

ofn.lpstrFileTitle = NULL;

// Устанавливаем максимальную длину имени файла без расширения

ofn.nMaxFileTitle = 0;

ofn.lpstrInitialDir = "C:/Users/Acer/Desktop/films/osisp-assets/files";

// Устанавливаем флаги для диалогового окна выбора файла (директория должна существовать и файл должен существовать)

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

}

**SoftwareColors.h**

#pragma once

typedef struct Color {

int r, g, b;

Color() { r = 0; g = 0; b = 0; }

Color(COLORREF fromref) {

b = (fromref >> 16) & 0xFF;

g = (fromref >> 8) & 0xFF;

r = (fromref & 0xFF); }

Color(int R, int G, int B) { r = R; g = G; b = B; }

Color operator+(const Color& second) { return Color(r + second.r, b + second.b, g + second.g); }

Color operator-(const Color& second) { return Color(r - second.r, b - second.b, g - second.g); }

Color operator\*(int t) { return Color(r \* t, b \* t, g \* t); }

Color operator/(int t) { return Color(r / t, b / t, g / t); }

COLORREF toRGB() {

return (b << 16) | (g << 8) | r;

}

Color Lerp(Color toColor, int t, int p) {

return \*this + (toColor - \*this) \* t / p;

}

};

void GradientRect(HDC hDC, const RECT\* lprc, Color leftColor, Color rightColor) {

// Вычисляем ширину прямоугольника

int width = lprc->left - lprc->right;

// Проходим по каждой горизонтальной координате в пределах прямоугольника

for (int x = lprc->right; x < lprc->left; x++) {

// Интерполируем цвет между левым и правым цветами на основе текущей горизонтальной позиции

int curColor = leftColor.Lerp(rightColor, x - lprc->right, width).toRGB();

// Проходим по каждой вертикальной координате в пределах прямоугольника

for (int y = lprc->top; y < lprc->bottom; y++) {

// Устанавливаем текущий цвет пикселя в контексте устройства (hDC)

SetPixel(hDC, x, y, curColor);

}

}

}

**SoftwareDefinitions.h**

#pragma once

#define OnExitSoftware 1

#define OnClearField 2

#define OnReadColor 3

#define OnSaveFile 4

#define OnLoadFile 5

#define DigIndexColorR 200

#define DigIndexColorG 201

#define DigIndexColorB 202

#define TextBufferSize 256

char Buffer[TextBufferSize];

unsigned num;

HWND hStaticControl;

HWND hEditControl;

HWND hNumberControl;

char filename[260];

OPENFILENAMEA ofn;

int colorR, colorG, colorB;

HFONT fontRectangle, fontStatic;

COLORREF fontColor;

HBRUSH brushRectangle;

RECT windowRectangle;

PAINTSTRUCT ps;

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 MainWndAddMenus(HWND hWnd);

void MainWndAddWidgets(HWND hWnd);

void SetOpenFileParams(HWND hWnd);

void SaveData(LPCSTR path);

void LoadData(LPCSTR path);