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

ОТЧЕТ

к лабораторной работе №2

на тему

**РАСШИРЕННОЕ ИСПОЛЬЗОВАНИЕ ОКОННОГО ИНТЕРФЕЙСА WIN32 И GDI**

Студент Е. С. Багровец

Преподаватель Н. Ю. Гриценко

Минск 2023

**СОДЕРЖАНИЕ**

[1 Постановка задачи 3](#_6127s8b6t8wy)

[2 Краткие теоретические сведения 4](#_r4y5108cx8yu)

[3 Результат выполнения лабораторной работы](#_jnfz1qbvv5j4) 5

[Выводы](#_ldsbh3e5c0xh) 6

[Список использованных источников 7](#_qto16fco0nf2)

[Приложение А (обязательное) Листинг кода](#_54xlhz7mfhe) 8

# 1 ПОСТАНОВКА ЗАДАЧИ

Целью выполнения данной лабораторной работы является разработка графического приложения с использованием Win32 API и GDI, которое будет обладать расширенным функционалом для работы с оконным интерфейсом. Разработать текстовый редактор с возможностью настройки тем оформления, таких как стили текста и цвета фона. Это включает в себя реализацию пользовательских настроек для внешнего вида текста, например, его размера, шрифта, стиля (жирный, курсив), а также выбор цветовой схемы для текста и фона. Такой редактор позволит пользователям адаптировать интерфейс под свои предпочтения, что является важным аспектом в разработке современных приложений, обеспечивая удобство использования и персонализацию пользовательского опыта.

# 2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ ТРЕБОВАНИЯ

Win32 API (Windows API) представляет собой набор функций и интерфейсов, предоставляемых операционной системой Windows для разработки приложений. Этот мощный набор инструментов обеспечивает доступ к различным функциональным возможностям Windows, включая создание и управление окнами, обработку сообщений, работу с файлами и реестром, а также многие другие операции. Win32 API играет ключевую роль в разработке приложений для Windows и обеспечивает высокую степень контроля над поведением приложений.

GDI (Graphics Device Interface) – это часть Win32 API, отвечающая за графический вывод и взаимодействие с графическим оборудованием. GDI предоставляет функции для рисования графических объектов, создания изображений, управления цветами и шрифтами, а также реализации различных эффектов. Этот компонент позволяет разработчикам создавать интерфейсы пользовательских приложений, включая окна, кнопки, текст и другие графические элементы. GDI обеспечивает визуальное качество и интерактивность приложений, делая их более привлекательными и функциональными для пользователей.

Совместное использование Win32 API и GDI позволяет разработчикам создавать мощные и интерактивные графические приложения под операционной системой Windows, обеспечивая широкие возможности по работе с графикой, окнами и элементами управления.

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

* Механизм перехвата сообщения (winhooks) в Windows API, представляют собой перехват событий в системе. Существует несколько типов хуков, таких как: WH\_KEYBOARD (Перехватывает нажатия клавиш), WH\_MOUSE (Перехватывает действия мыши), WH\_SHELL (Перехватывает различные события оболочки). Они позволяют приложениям мониторить и реагировать на события, происходящие в системе, даже если эти события происходят вне контекста самого приложения.
* Для установки перехвата сообщения используется функция SetWindowsHookEx, которая принимает тип хука и указатель на функцию обратного вызова (hook procedure). Эта функция возвращает дескриптор хука, который может быть использован для его удаления с помощью функции UnhookWindowsHookEx.
* MSFTEDIT\_CLASS – это класс элемента управления в Windows API, представляющий собой более продвинутый и богатый функционалом текстовый редактор
* Класс CHARFORMAT2 представляет структуру, используемую для управления форматированием текста в элементе управления. Она включает атрибуты шрифта, цвета текста и другие свойства форматирования.

# 3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

В ходе выполнения лабораторной работы было реализовано графическое приложение, позволяющее пользователю изменять шрифты в текстовом редакторе, изменять цвет заднего фона текстового редактора и подсвечивать базовые кодовые конструкции (Рисунок 1).

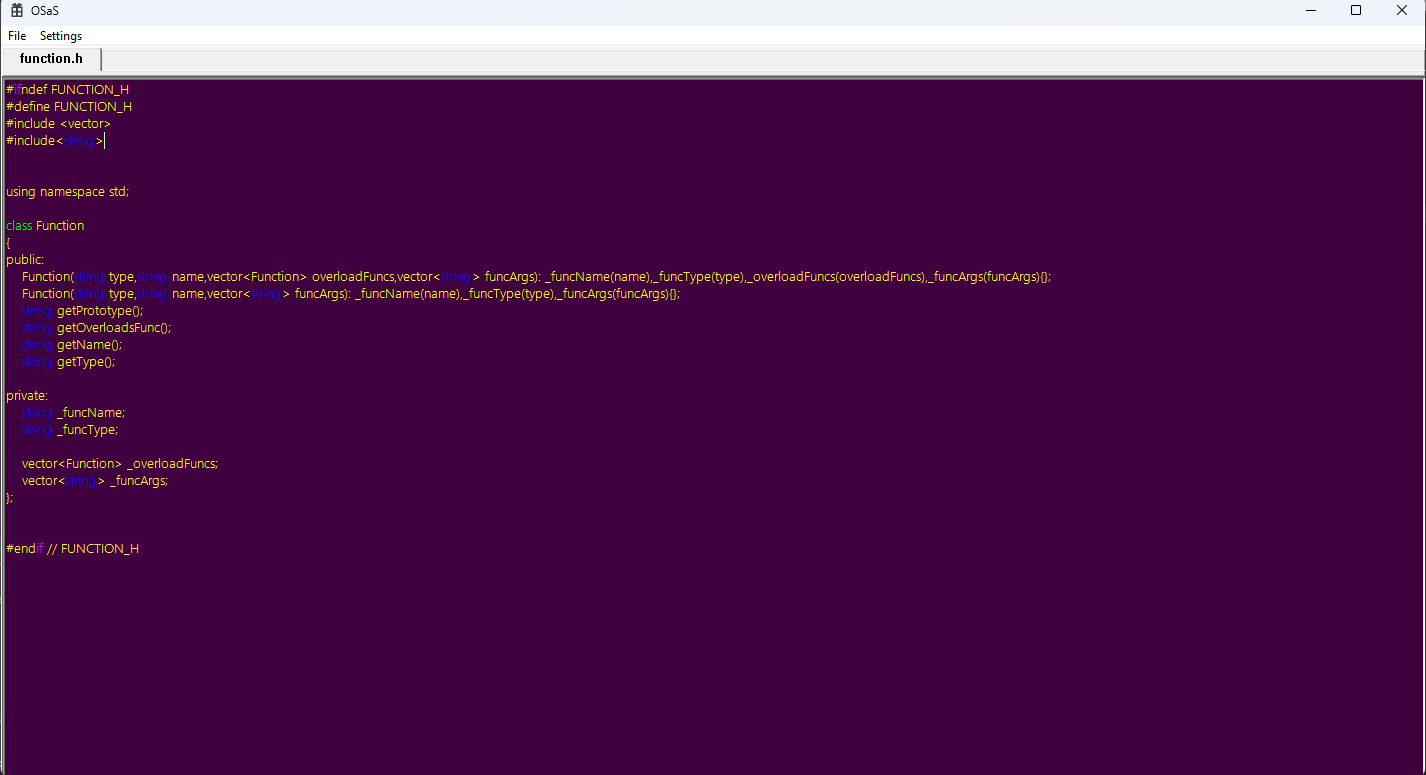


Рисунок 1 – Результат работы программы

# ВЫВОДЫ

В ходе выполнения лабораторной были изучены возможности расширенного использования оконного интерфейса Win32 и GDI в контексте создания графического приложения. Это включало изучение работы с элементами управления, обработки различных сообщений от пользователя и операционной системы, а также механизмов перехвата сообщений через Winhook.

В результате было успешно разработано оконное приложение с использованием элементов управления, что позволило создать удобный и функциональный пользовательский интерфейс. Программа обеспечивает возможность изменения шрифтов в текстовом редакторе, что дает пользователю контроль над внешним видом текста. Также пользователь может настроить цвет заднего фона текстового редактора, что способствует комфортному восприятию интерфейса, также была создана подсветка базовых кодовых конструкций, что повышает удобство работы программистов и обеспечивает наглядное выделение ключевых частей кода.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Щупак Ю. Win32 API. Разработка приложений для Windows. – СПб: Питер, 2008. – 592 с.: ип.
2. Windows GDI [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/gdi/windows-gdi

# ПРИЛОЖЕНИЕ А

# (обязательное)

# Листинг кода

Листинг 1 ­ Файл main.cpp

#include "framework.h"

#include "prototypes.h"

#include "tabHandlers.h"

#include "fileHandlers.h"

#include "commandsPrototypes.h"

#include "globalVariables.h"

#include "main.h"

int WINAPI wWinMain(HINSTANCE hInstance,

HINSTANCE hPrevInstance,

LPWSTR lpCmdLine,

int nCmdShow)

{

UNREFERENCED\_PARAMETER(hPrevInstance);

UNREFERENCED\_PARAMETER(lpCmdLine);

// Инициализация глобальных строк

LoadStringW(hInstance, IDS\_APP\_TITLE, szTitle, MAX\_LOADSTRING);

LoadStringW(hInstance, IDC\_OSAS, szWindowClass, MAX\_LOADSTRING);

MyRegisterClass(hInstance);

// Выполнить инициализацию приложения:

if (!InitInstance(hInstance, nCmdShow))

{

return FALSE;

}

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

hKeyboardHook = SetWindowsHookEx(WH\_KEYBOARD\_LL, KeyboardProc, hInstance, 0);

MSG msg;

// Цикл основного сообщения:

while (GetMessage(&msg, nullptr, 0, 0))

{

if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

UnhookWindowsHookEx(hKeyboardHook);

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\_OSAS));

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

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

wcex.lpszMenuName = MAKEINTRESOURCEW(IDC\_OSAS);

wcex.lpszClassName = szWindowClass;

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

return RegisterClassExW(&wcex);

}

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)

{

hInst = hInstance; // Сохранить маркер экземпляра в глобальной переменной

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)

{

switch (message)

{

case WM\_COMMAND:

{

int wmId = LOWORD(wParam);

switch (wmId)

{

case ENM\_UPDATE:

HighLightKeyWords();

break;

case IDM\_EXIT:

DestroyWindow(hWnd);

break;

case NEW\_FILE\_COMMAND:

NewFileCommand();

break;

case OPEN\_FILE\_COMMAND:

OpenFileCommand();

break;

case SAVE\_FILE\_COMMAND:

SaveFileCommand();

break;

case CLOSE\_FILE\_COMMAND:

CloseFileCommand();

break;

case CLOSE\_TAB\_COMMAND:

CloseFileCommand();

break;

default:

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

}

}

break;

case WM\_NOTIFY:

{

NMHDR\* nmhdr = (NMHDR\*)lParam;

if (nmhdr->code == TCN\_SELCHANGE) {

SwitchTab();

}

}

break;

case WM\_SIZE:

modifiedWidth = LOWORD(lParam);

modifiedHeight = HIWORD(lParam);

MoveWindow(tabControlWidget, 1, 1, modifiedWidth, 30, TRUE);

MoveWindow(editWidget, 1, 31, modifiedWidth, modifiedHeight, TRUE);

break;

case WM\_PAINT:

{

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hWnd, &ps);

EndPaint(hWnd, &ps);

}

break;

case WM\_CREATE:

WinWidgetsCreation(hWnd);

WinMenuCreation(hWnd);

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

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

}

return 0;

}

LRESULT CALLBACK KeyboardProc(int nCode, WPARAM wParam, LPARAM lParam)

{

if (nCode >= 0 && wParam == WM\_KEYDOWN)

{

KBDLLHOOKSTRUCT\* pKeyInfo = reinterpret\_cast<KBDLLHOOKSTRUCT\*>(lParam);

if (GetAsyncKeyState(VK\_CONTROL) & 0x8000)

{

if (GetAsyncKeyState(VK\_SHIFT) & 0x7000) {

}

switch (pKeyInfo->vkCode)

{

case 'S':

SaveFileCommand();

break;

case 'O':

OpenFileCommand();

break;

case 'C':

CloseFileCommand();

break;

case 'N':

NewFileCommand();

break;

default:

break;

}

}

}

return CallNextHookEx(hKeyboardHook, nCode, wParam, lParam);

}

void WinWidgetsCreation(HWND hWnd) {

tabControlWidget = CreateWindowEx(0, WC\_TABCONTROL, L"", WS\_CHILD | WS\_VISIBLE | TCS\_FIXEDWIDTH,

1, 0, 500, 30, hWnd, (HMENU)IDC\_TABCTRL, GetModuleHandle(NULL), NULL);

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

editWidget = CreateWindowEx(

WS\_EX\_CLIENTEDGE,

MSFTEDIT\_CLASS, L"",

WS\_BORDER | WS\_CHILD | WS\_VISIBLE | ES\_MULTILINE | WS\_VSCROLL | WS\_TABSTOP,

0, 0, 800, 600,

hWnd, NULL, hInst, NULL

);

UpdateWindow(editWidget);

}

void WinMenuCreation(HWND hWnd) {

HMENU fileMenu = CreateMenu();

HMENU fileSettings = CreateMenu();

HMENU fileSubMenu = CreateMenu();

AppendMenu(fileSubMenu, MF\_STRING, NEW\_FILE\_COMMAND, L"New");

AppendMenu(fileSubMenu, MF\_STRING, OPEN\_FILE\_COMMAND, L"Open");

AppendMenu(fileSubMenu, MF\_STRING, SAVE\_FILE\_COMMAND, L"Save");

AppendMenu(fileSubMenu, MF\_STRING, CLOSE\_FILE\_COMMAND, L"Close");

AppendMenu(fileMenu, MF\_POPUP, (UINT\_PTR)fileSubMenu, L"File");

AppendMenu(fileMenu, MF\_STRING, NULL, L"Settings");

SetMenu(hWnd, fileMenu);

}

void CloseFileCommand()

{

if (TabCtrl\_GetItemCount(tabControlWidget) == 0)

{

return;

}

int result = MessageBox(NULL, L"Do you want to save the file?", L"Save File", MB\_YESNOCANCEL | MB\_ICONQUESTION);

if (result == IDYES)

{

SaveFileCommand();

}

auto name = fileNames[tabIndex];

fileNames.erase(fileNames.begin() + tabIndex);

filePathes.erase(name);

fileData.erase(name);

TabCtrl\_DeleteItem(tabControlWidget, tabIndex);

SwitchTab();

}

void SaveFileCommand()

{

if (!TabCtrl\_GetItemCount(tabControlWidget))

{

return;

}

tabIndex = TabCtrl\_GetCurSel(tabControlWidget);

int length = GetWindowTextLength(editWidget) + 1;

LPSTR data = new CHAR[length];

length = GetWindowTextA(editWidget, data, length);

auto it = filePathes.find(fileNames[tabIndex]);

if (it != filePathes.end())

{

LPWSTR path = const\_cast<LPWSTR>((\*it).second.c\_str());

SaveFile(path, data, length);

}

else

{

OPENFILENAME ofn;

wchar\_t szFileName[1024] = L"";

ZeroMemory(&ofn, sizeof(ofn));

ofn.lStructSize = sizeof(ofn);

ofn.hwndOwner = NULL;

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

ofn.lpstrFile = szFileName;

ofn.nMaxFile = MAX\_PATH;

ofn.Flags = OFN\_OVERWRITEPROMPT | OFN\_NOCHANGEDIR;

if (GetOpenFileName(&ofn) == TRUE)

{

SaveFile(szFileName, data, length);

std::wstring name = szFileName;

name = name.substr(name.find\_last\_of('\\') + 1);

fileData[name] = fileData[fileNames[tabIndex]];

fileNames[tabIndex] = name;

filePathes[name] = szFileName;

TCITEM tcItem = { 0 };

tcItem.mask = TCIF\_TEXT;

tcItem.pszText = const\_cast<LPWSTR>(name.data());

TabCtrl\_SetItem(tabControlWidget, tabIndex, &tcItem);

}

}

}

void NewFileCommand()

{

int num = 1;

auto it = fileData.find(L"File" + std::to\_wstring(num));

while (it != fileData.end())

{

num++;

it = fileData.find(L"File" + std::to\_wstring(num));

}

std::wstring fileName = L"File" + std::to\_wstring(num);

CreateTab(const\_cast<LPWSTR>(fileName.c\_str()), L"", nullptr);

}

void OpenFileCommand()

{

OPENFILENAME ofn;

wchar\_t szFileName[1024] = L"";

std::vector<char> pathBuffer(1024);

ZeroMemory(&ofn, sizeof(ofn));

ofn.lStructSize = sizeof(ofn);

ofn.hwndOwner = NULL;

ofn.lpstrFilter = L"Text Files\0\*.cpp;\*.h\0";;

ofn.lpstrFile = szFileName;

ofn.nMaxFile = MAX\_PATH;

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

if (GetOpenFileName(&ofn) == TRUE)

{

std::vector<char> content;

WideCharToMultiByte(CP\_UTF8, 0, szFileName, -1, pathBuffer.data(), sizeof(szFileName), NULL, NULL);

OpenFile(pathBuffer.data(), content);

std::wstring name = szFileName;

name = name.substr(name.find\_last\_of('\\') + 1);

std::wstring data = std::wstring(content.begin(), content.end());

LPWSTR fileName = const\_cast<LPWSTR>(name.data());

LPCWSTR fileData = data.c\_str();

LPWSTR filePath = szFileName;

CreateTab(fileName, fileData, filePath);

}

}

void ReplaceCloseButton(int currentTabIndex)

{

/\* RECT tabRect;

TabCtrl\_GetItemRect(tabControlWidget, currentTabIndex, &tabRect);

if (closeButton)

{

MoveWindow(closeButton, tabRect.right - 20, tabRect.top + 2, 15, 15, TRUE);

}

else

{

closeButton = CreateWindowA("button", "X", WS\_CHILD | WS\_VISIBLE,

tabRect.right - 20, tabRect.top + 2, 15, 15, tabControlWidget,

(HMENU)CLOSE\_TAB\_COMMAND, NULL, NULL);

}\*/

}

void SwitchTab()

{

if (TabCtrl\_GetItemCount(tabControlWidget) == 0)

{

DestroyWindow(closeButton);

closeButton = nullptr;

SetWindowTextW(editWidget, L"");

return;

}

int length = GetWindowTextLengthW(editWidget) + 1;

LPWSTR data = new WCHAR[length];

length = GetWindowTextW(editWidget, data, length);

fileData[fileNames[tabIndex]] = data;

tabIndex = TabCtrl\_GetCurSel(tabControlWidget);

SendMessage(editWidget, EM\_SETSEL, 0, -1);

SendMessage(editWidget, EM\_REPLACESEL, TRUE,

reinterpret\_cast<LPARAM>(fileData[fileNames[tabIndex]].c\_str()));

ReplaceCloseButton(tabIndex);

}

void CreateTab(LPWSTR name, LPCWSTR data, LPWSTR path)

{

if (TabCtrl\_GetItemCount(tabControlWidget) != 0)

{

int length = GetWindowTextLengthW(editWidget) + 1;

LPWSTR newData = new WCHAR[length];

length = GetWindowTextW(editWidget, newData, length);

fileData[fileNames[tabIndex]] = newData;

}

TCITEM tie;

tie.mask = TCIF\_TEXT;

tie.pszText = name;

TabCtrl\_InsertItem(tabControlWidget, fileNames.size(), &tie);

fileNames.emplace\_back(name);

fileData.insert(std::make\_pair(fileNames.back(), std::wstring(data)));

if (path)

{

filePathes.insert(std::make\_pair(fileNames.back(), std::wstring(path)));

}

tabIndex = fileNames.size() - 1;

auto lData = reinterpret\_cast<LPARAM>(data);

SendMessage(editWidget, EM\_SETSEL, 0, -1);

SendMessage(editWidget, EM\_REPLACESEL, TRUE, lData);

ReplaceCloseButton(fileNames.size() - 1);

TabCtrl\_SetCurSel(tabControlWidget, fileNames.size() - 1);

}

Листинг 2 ­ Файл fileHandlers.cpp

#include "fileHandlers.h"

#include "Resource.h"

#include "framework.h"

bool OpenFile(LPCSTR path, std::vector<char>& content)

{

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

if (hFile != INVALID\_HANDLE\_VALUE) {

DWORD fileSize = GetFileSize(hFile, NULL);

if (fileSize == INVALID\_FILE\_SIZE && GetLastError() != NO\_ERROR) {

CloseHandle(hFile);

return false;

}

content.resize(fileSize + 1);

DWORD bytesRead;

if (ReadFile(hFile, content.data(), fileSize, &bytesRead, NULL)) {

content[bytesRead] = '\0';

}

else {

return false;

}

CloseHandle(hFile);

}

else {

return false;

}

}

bool SaveFile(LPWSTR path, LPCSTR data, int length)

{

HANDLE file = CreateFileW(path , GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

DWORD bytes;

WriteFile(file, data, length, &bytes, NULL);

CloseHandle(file);

delete[] data;

return true;

}

void HighLightKeyWords()

{

const std::unordered\_map<std::wstring, COLORREF> wordColors = {

{ L"class", RGB(0, 255, 0) },

{ L"struct", RGB(0, 255, 0) },

{ L"static", RGB(0, 0, 255) },

{ L"int", RGB(0, 0, 255) },

{ L"auto", RGB(0, 0, 255) },

{ L"float", RGB(0, 0, 255) },

{ L"void", RGB(0, 0, 255) },

{ L"string", RGB(0, 0, 255) },

{ L"double", RGB(0, 0, 255) },

{ L"static", RGB(0, 0, 255) },

{ L"while", RGB(255, 0, 255) },

{ L"if", RGB(255, 0, 255) } ,

{ L"switch", RGB(255, 0, 255) } ,

{ L"return", RGB(255, 0, 255) } ,

};

CHARFORMAT2 charFormat;

memset(&charFormat, 0, sizeof(CHARFORMAT2));

charFormat.cbSize = sizeof(CHARFORMAT2);

charFormat.dwMask = CFM\_COLOR;

charFormt.crTextColor = RGB(0, 0, 0);

int textLength = GetWindowTextLength(editWidget);

std::wstring text(textLength + 1, L'\0');

GetWindowText(editWidget, &text[0], textLength + 1);

for (auto& entry : wordColors)

{

FINDTEXTEXW findText;

findText.chrg.cpMin = 0;

findText.chrg.cpMax = -1;

findText.lpstrText = entry.first.c\_str();

while (SendMessage(editWidget, EM\_FINDTEXTEX, FR\_DOWN, (LPARAM)&findText) != -1)

{

CHARRANGE selRange;

selRange.cpMin = findText.chrgText.cpMin;

selRange.cpMax = findText.chrgText.cpMax;

SendMessage(editWidget, EM\_EXSETSEL, 0, (LPARAM)&selRange);

charFormat.crTextColor = entry.second;

SendMessage(editWidget, EM\_SETCHARFORMAT, SCF\_SELECTION, (LPARAM)&charFormat);

findText.chrg.cpMin = findText.chrgText.cpMax;

findText.chrg.cpMax = -1;

}

}

}

void OpenColorDialog()

{

CHOOSECOLOR cc = { sizeof(CHOOSECOLOR) };

static COLORREF custColors[16] = { 0 };

cc.rgbResult = RGB(255, 255, 255);

cc.lpCustColors = custColors;

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

if (ChooseColor(&cc))

{

SendMessage(editWidget, EM\_SETBKGNDCOLOR, FALSE, cc.rgbResult);

}

}

void ChangeFont()

{

CHARFORMAT2 cf;

cf.cbSize = sizeof(CHARFORMAT2);

CHOOSEFONT cfDialogParams = { 0 };

LOGFONT lf = { 0 };

cfDialogParams.lStructSize = sizeof(CHOOSEFONT);

cfDialogParams.hwndOwner = editWidget;

cfDialogParams.lpLogFont = &lf;

cfDialogParams.Flags = CF\_INITTOLOGFONTSTRUCT | CF\_EFFECTS | CF\_SCREENFONTS;

if (ChooseFont(&cfDialogParams)) {

lstrcpy(cf.szFaceName, lf.lfFaceName);

cf.yHeight = lf.lfHeight \* 20;

cf.dwEffects = 0;

cf.dwMask = CFM\_FACE | CFM\_SIZE;

SendMessage(editWidget, EM\_SETCHARFORMAT, SCF\_ALL, (LPARAM)&cf);

}

}

void ChangeFontColor() {

CHOOSECOLOR cc = {0};

ZeroMemory(&cc, sizeof(cc));

cc.lStructSize = sizeof(cc);

cc.hwndOwner = editWidget;

cc.lpCustColors = new COLORREF[16]{ 0 };

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

if (ChooseColor(&cc))

{

COLORREF chosenColor = cc.rgbResult;

CHARFORMAT2 cf = { };

cf.cbSize = sizeof(CHARFORMAT2);

cf.dwMask = CFM\_COLOR;

cf.crTextColor = chosenColor;

SendMessage(editWidget, EM\_SETCHARFORMAT, SCF\_ALL, reinterpret\_cast<LPARAM>(&cf));

}

Листинг 3 ­ Файл prototypes.h

#pragma once

#include "framework.h"

ATOM MyRegisterClass(HINSTANCE );

BOOL InitInstance(HINSTANCE, int);

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

INT\_PTR CALLBACK About(HWND, UINT, WPARAM, LPARAM);

LRESULT CALLBACK KeyboardProc(int, WPARAM, LPARAM );

void WinMenuCreation(HWND);

void WinWidgetsCreation(HWND);

bool OpenFile(LPCSTR, std::vector<char>&);

bool SaveFile(LPWSTR, LPCSTR, int);

void NewFileCommand();

void OpenFileCommand();

void CloseFileCommand();

void SaveFileCommand();

void CreateTab(LPWSTR, LPCWSTR, LPWSTR);

void SwitchTab();

bool OpenFile(LPCSTR path, std::vector<char>& content);

bool SaveFile(LPWSTR path, LPCSTR data, int length);

void ReplaceCloseButton(int);

void HighLightKeyWords();

void OpenColorDialog();

void ChangeFont();

void ChangeFontColor();

Листинг 4 ­ Файл Resources.h

#define IDS\_APP\_TITLE 103

#define IDC\_TABCTRL 120

#define IDR\_MAINFRAME 128

#define IDD\_OSAS\_DIALOG 102

#define IDD\_ABOUTBOX 103

#define IDM\_ABOUT 104

#define IDM\_EXIT 105

#define IDI\_OSAS 107

#define IDI\_SMALL 108

#define IDC\_OSAS 109

#define IDC\_MYICON 2

#ifndef IDC\_STATIC

#define IDC\_STATIC -1

#define IDC\_RICHEDIT 111

#endif

// Кастомные

#define NEW\_FILE\_COMMAND 10

#define OPEN\_FILE\_COMMAND 11

#define SAVE\_FILE\_COMMAND 12

#define CLOSE\_FILE\_COMMAND 13

#define CLOSE\_TAB\_COMMAND 14

#define CHANGE\_BG\_COLOR 15

#define CHANGE\_FONT 16

#define HIGHTLIGHT\_TEXT 17

#define CHANGE\_FONT\_COLOR 18

#ifdef APSTUDIO\_INVOKED

#ifndef APSTUDIO\_READONLY\_SYMBOLS

#define \_APS\_NO\_MFC 130

#define \_APS\_NEXT\_RESOURCE\_VALUE 129

#define \_APS\_NEXT\_COMMAND\_VALUE 32771

#define \_APS\_NEXT\_CONTROL\_VALUE 1000

#define \_APS\_NEXT\_SYMED\_VALUE 110

#endif

#endif