

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 № 1

на тему «Расширенное использование оконного интерфейса Win 32 и GDI. Формирование сложных изображений, создание и использование элементов управления, обработка различных сообщений, механизм перехвата сообщений (winhook)»

Выполнил:  
студент гр. 153504  
Багровец Е.С.

Проверил:  
Гриценко Н.Ю.

## СОДЕРЖАНИЕ

1 Цели работы .....	3
2 Краткие теоретические сведения .....	4
3 Полученные результаты .....	18
Выводы.....	19

## **1 Цели работы**

1. Изучить расширенное использование оконного интерфейса Win 32 и GDI.
2. Формирование сложных изображений, создание и использование элементов управления, обработка различных сообщений, механизм перехвата сообщений (winhook).
3. Разработать текстовый редактор с поддержкой настраиваемых тем оформления (стили текста, цвета фона).

## 2 Краткие теоретические сведения

В Windows API для взаимодействия между приложениями и операционной системой используется система сообщений. Процессы и потоки общаются друг с другом и с операционной системой, отправляя и обрабатывая сообщения. Эти сообщения могут содержать различные типы информации, такие как пользовательский ввод, системные уведомления и команды. Для реализации коммуникации между процессами используется функция `SendMessage`. Эта функция отправляет сообщение указанному окну и блокирует вызывающий поток до тех пор, пока не будет обработано сообщение.

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`.

Когда событие, подпадающее под перехват, происходит, система вызывает функцию обратного вызова (hook procedure), что позволяет приложению выполнить необходимые действия перед или после события.

`MSFTEDIT_CLASS` - это класс элемента управления в Windows API, представляющий собой более продвинутый и богатый функционалом текстовый редактор. Этот класс представляет собой текстовое поле с расширенными возможностями форматирования текста, что делает его более функциональным, чем стандартный элемент управления `Edit Control`.

Такой текстовый редактор подходит для более сложных задач редактирования текста, требующих форматирования и манипуляций с богатым контентом. Он часто используется в приложениях, где требуется работа с различными аспектами текста и его визуализации.

Класс `CHARFORMAT2` представляет структуру, используемую для управления форматированием текста в элементе управления. Она включает атрибуты шрифта, цвета текста и другие свойства форматирования.

Класс `CHOOSECOLOR` представляет диалог выбора цвета. Он содержит информацию о выбранном цвете и настройках диалога.

Класс `LOGFONT` представляет структуру для хранения атрибутов шрифта, таких как имя шрифта, размер и стиль.

Листинг 1 — Код исходной программы:  
**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);
}

```

## 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;
    charFormat.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)
{
    FINDTEXTW findText;
    findText.chrg.cpMin = 0;
    findText.chrg.cpMax = -1;
    findText.lpstrText = entry.first.c_str();
    while (SendMessage(editWidget, EM_FINDTEXT, FR_DOWN,
(LPPARAM)&findText) != -1)
    {
        CHARRANGE selRange;
        selRange.cpMin = findText.chrgText.cpMin;
        selRange.cpMax = findText.chrgText.cpMax;
        SendMessage(editWidget, EM_EXSETSEL, 0,
(LPPARAM)&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_SETBKGDNDCOLOR, 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,
(LPPARAM)&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));
    }
}

```

## 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();

```

## 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
#ifdef 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 HIGHLIGHT_TEXT 17
#define CHANGE_FONT_COLOR 18
#ifdef APSTUDIO_INVOKED
#ifdef 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

```

## globalVariables.h

```

#pragma once
#include "framework.h"
#define MAX_LOADSTRING 100
// Глобальные переменные:
HINSTANCE hInst; // текущий экземпляр
WCHAR szTitle[MAX_LOADSTRING]; // Текст строки заголовка
WCHAR szWindowClass[MAX_LOADSTRING]; // имя класса главного окна
HHOOK hKeyboardHook; // хук для отловления нажатия горячих клавиш
std::vector<char> content;
HWND tabControlWidget;
HWND editWidget;
HWND closeButton = nullptr;
int tabIndex;
int modifiedHeight;
int modifiedWidth;
std::vector<std::wstring> fileNames; // название открытых файлов
std::unordered_map<std::wstring, std::wstring> fileData;

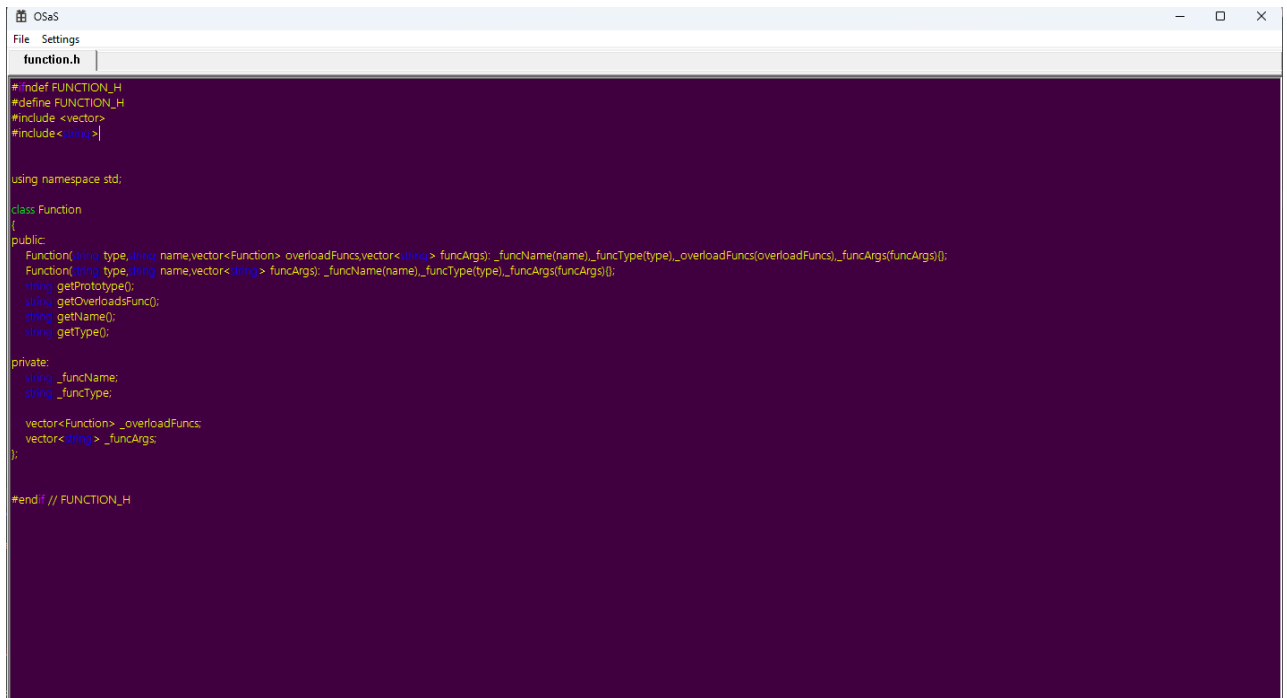
```



```
// содержимое файлов файлов  
std::unordered_map<std::wstring, std::wstring> filePathes;  
// пути уже существующих фалов
```

### 3 Полученные результаты

Результат работы программы показан на рисунке 3.1.



```
#ifndef FUNCTION_H
#define FUNCTION_H
#include <vector>
#include <string>

using namespace std;

class Function
{
public:
    Function(string type, string name, vector<Function> overloadFuncs, vector<string> funcArgs, _funcName(name), _funcType(type), _overloadFuncs(overloadFuncs), _funcArgs(funcArgs));
    Function(string type, string name, vector<string> funcArgs, _funcName(name), _funcType(type), _funcArgs(funcArgs));
    string getPrototype();
    string getOverloadsFunc();
    string getName();
    string getType();

private:
    string _funcName;
    string _funcType;

    vector<Function> _overloadFuncs;
    vector<string> _funcArgs;
};

#endif // FUNCTION_H
```

Рисунок 3.1 – Результат работы программы

## **Выводы**

В ходе выполнения лабораторной работы были изучены расширенное использование оконного интерфейса Win 32 и GDI. Как результат, создано оконное приложение с использованием элементов управления, обработкой различных сообщений, механизмом перехвата сообщений (winhook). Программа позволяет пользователю изменять шрифты в текстовом редакторе, изменять цвет заднего фона текстового редактора и подсвечивать базовые кодовые конструкции.