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 №2
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).

БГУИР КП 1-40 04 01

Выполнил: студент группы 153502
Кузнецов Е. А.

Проверил: Гриценко Н.Ю.

Минск 2023

СОДЕРЖАНИЕ

1 Цель работы

Целью выполнения лабораторной работы является создание оконного приложения на *Win32 API*, позволяющим отработать такие навыки, как: расширенное использование оконного интерфейса *Win 32* и *GDI*, формирование сложных изображений, создание и использование элементов управления, обработка различных сообщений, механизм перехвата сообщений (*winhook*).

В качестве задачи необходимо разработать текстовый редактор с поддержкой настраиваемых тем оформления (стили текста, цвета фона).

2 Теоретические сведения

Интерфейс графических устройств *Microsoft Windows* (*GDI*) позволяет приложениям использовать графику и отформатированный текст как на видеодисплее, так и на принтере. Приложения на основе *Windows* не обращаются к графическому оборудованию напрямую. Вместо этого *GDI* взаимодействует с драйверами устройств от имени приложений.

Цветовая палитра – это массив, содержащий значения цветов, определяющие цвета, которые в настоящее время могут отображаться или рисоваться на устройстве вывода. Цветовые палитры используются устройствами, которые способны генерировать много цветов, но могут отображать или рисовать их подмножество в любой момент времени. Для таких устройств система поддерживает системную палитру для отслеживания текущих цветов устройства и управления ими. Приложения не имеют прямого доступа к системной палитре. Вместо этого система связывает палитру по умолчанию с каждым контекстом устройства.

Приложения могут использовать цвета в палитре по умолчанию или определять собственные цвета, создавая логические палитры и связывая их с контекстами отдельных устройств.

Функция *ChooseColor* позволяет создать диалоговое окно, позволяющее пользователю выбрать цвет. Если пользователь нажимает кнопку «ОК» в диалоговом окне, возвращаемое значение не равно нулю. Член *rgbResult* структуры *CHOOSECOLOR* содержит значение цвета *RGB* для цвета, выбранного пользователем. Если пользователь отменяет или закрывает диалоговое окно или возникает ошибка, возвращаемое значение равно нулю.

3 Описание функций программы

Согласно формулировке задачи, были спроектированы следующие функции программы:

– Выбор стиля текста;

– Выбор цвета фона.

1) Выбор стиля текста

Для выбора стиля текста необходимо нажать на кнопку «Стиль текста» в меню «Темы оформления» (рисунок 1).

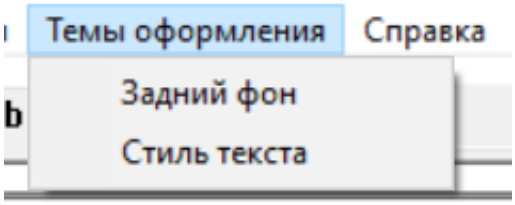

Рисунок 1 – Меню для выбора стиля текста или цвета фона

Далее следует выбрать стили в открывшемся окне (рисунок 2) и нажать кнопку «Ок».

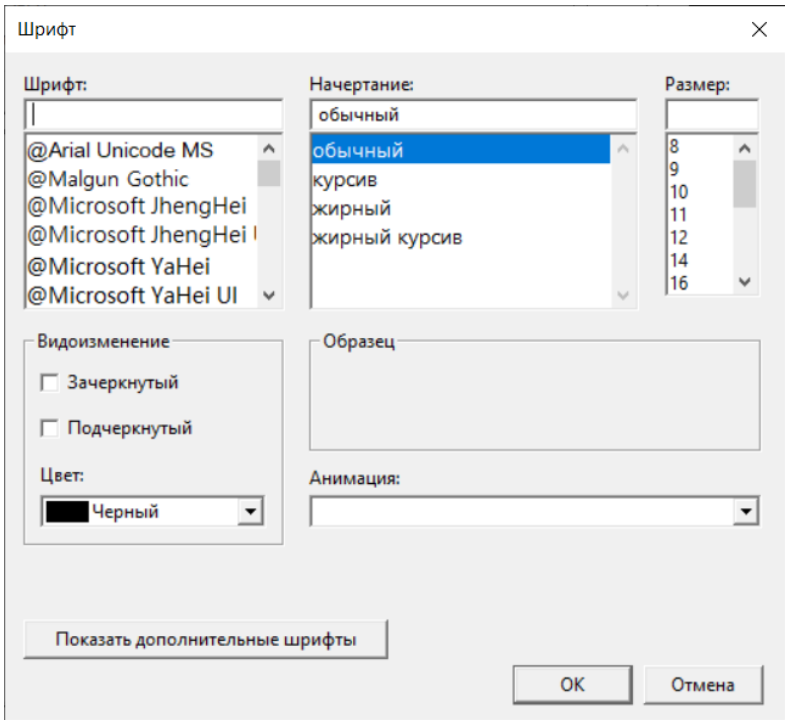

Рисунок 2 – Окно выбора стиля текста

2) Выбор цвета фона

Для выбора цвета фона необходимо нажать на кнопку «Задний фон» в меню «Темы оформления» (рисунок 1).

Далее следует выбрать цвет фона в открывшемся окне (рисунок 3) и нажать кнопку «Ок».

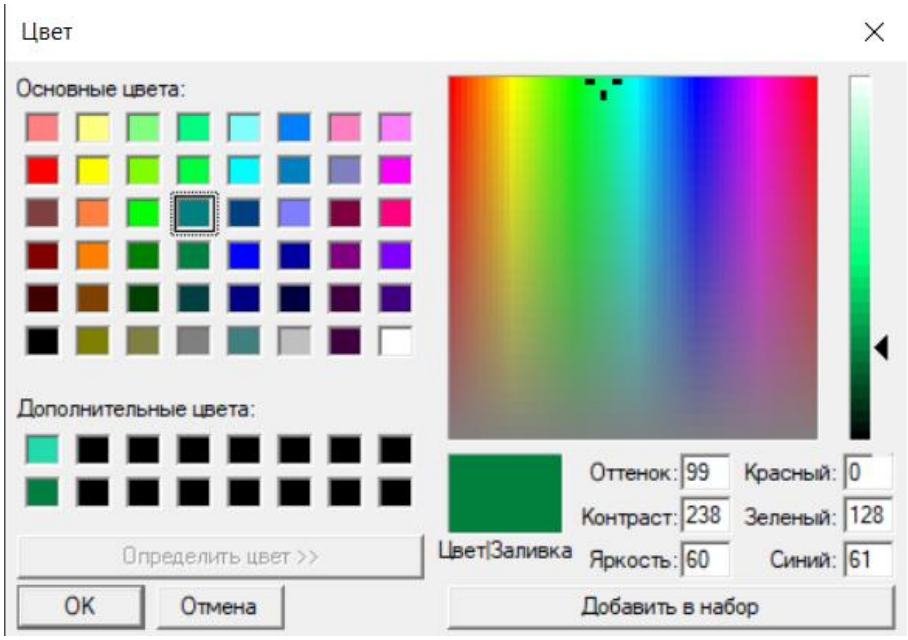

Рисунок 3 – Окно выбора цвета фона

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Windows GDI [электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/gdi/windows-gdi
Дата доступа 05.10.2023.
[2] Системная палитра [электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/gdi/system-palette
Дата доступа 05.10.2023.
[3] Разработка приложений с помощью WinAPI. [электронный ресурс]. – Режим доступа: https://shorturl.at/BDJW8 Дата доступа 05.10.2023.

ПРИЛОЖЕНИЕ А
Листинг кода

Файл LABA1.h

```
#include "framework.h"
#include <Richedit.h>
#include "LABA1.h"
#include <string>
#include <commdlg.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <commctrl.h>

#define MAX_LOADSTRING 100
#define IDC_TABCONTROL 1101
#define IDC_EDIT_TAB_START 2000

// Глобальные переменные:
HINSTANCE hInst;
WCHAR szTitle[MAX_LOADSTRING];
WCHAR szWindowClass[MAX_LOADSTRING];
HWND tabControl;                        // дескриптер вкладок в интерфейсе

int tabIndexCounter = 0;
static COLORREF acrCustClr[16];

ATOM                MyRegisterClass(HINSTANCE hInstance);    BOOL
InitInstance(HINSTANCE, int);                        // будет инициализировать
экземпляр приложения.
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);       //
обработка сообщений для главного окна
INT_PTR CALLBACK    About(HWND, UINT, WPARAM, LPARAM);           //
обрабатывать сообщения для диалогового окна "О программе".

int APIENTRY wWinMain(_In_ HINSTANCE hInstance,
        _In_opt_ HINSTANCE hPrevInstance, //предыдущий экземпляр
        _In_ LPWSTR    lpCmdLine, // аргументы командной строки
        _In_ int       nCmdShow) // флаги отображения окна
```

```
{
	UNREFERENCED_PARAMETER(hPrevInstance); //предназначены для
предотвращения
	UNREFERENCED_PARAMETER(lpCmdLine); //предупреждений о
неиспользуемых параметрах функции.
	LoadLibrary(L"Msftedit.dll");

	// Инициализация глобальных строк
	LoadStringW(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
	LoadStringW(hInstance, IDC_LABA1, szWindowClass,
MAX_LOADSTRING);
	MyRegisterClass(hInstance);

	// Выполнить инициализацию приложения:
	if (!InitInstance(hInstance, nCmdShow)) // создания и отображения главного
окна приложения
	{
		return FALSE;
	}

	HACCEL hAccelTable = LoadAccelerators(hInstance,
MAKEINTRESOURCE(IDC_LABA1));

	MSG msg; // хранение сообщений

	// Цикл основного сообщения:
	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; //  информация о классе окна

	wcex.cbSize = sizeof(WNDCLASSEX);

	wcex.style = CS_HREDRAW | CS_VREDRAW; //окно должно быть перерисовано, если его горизонтальный и вертикальный размер изменяется.
	wcex.hbrBackground = CreateSolidBrush(0x00FFFFFF);
	wcex.lpfnWndProc = WndProc; //оконная процедуру, которая будет обрабатывать сообщения для окна
	wcex.cbClsExtra = 0; //  Резервированные поля для
	wcex.cbWndExtra = 0; // дополнительных данных класса и окна
	wcex.hInstance = hInstance; // Установка экземпляра приложения, полученного как параметр функции
	wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_LABA1)); //значок приложения в окне
	wcex.hCursor = LoadCursor(nullptr, IDC_ARROW); // курсор мыши
	wcex.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1); // фон окна
	wcex.lpszMenuName = MAKEINTRESOURCEW(IDC_LABA1); // меню для окна
	wcex.lpszClassName = szWindowClass; // именя класса окна
	wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_SMALL)); // маленький значок приложения


	return RegisterClassExW(&wcex); // Регистрация класса окна и возврат атома класса
}

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
	hInst = hInstance; // Сохранить маркер экземпляра в глобальной переменной

	HWND hWnd = CreateWindow(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
		CW_USEDEFAULT, CW_USEDEFAULT, 500, 500, nullptr, nullptr, hInstance, nullptr);

	if (!hWnd)  // Проверка, было ли успешно создано окно
	{
		return FALSE;
```

```
	}

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

	return TRUE;
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM
wParam, LPARAM lParam)
{
	switch (message)
	{
	case WM_CREATE:
	{
		tabControl = CreateWindowEx(0, WC_TABCONTROL, L"",
WS_CHILD | WS_VISIBLE,
			10, 10, 460, 30, hWnd, (HMENU)IDC_TABCONTROL,
(HINSTANCE)GetWindowLong(hWnd, GWLP_HINSTANCE), nullptr);

		if (tabControl)
		{
			TCITEM tie;
			tie.mask = TCIF_TEXT;
			tabIndexCounter++;

			// Создайте первую вкладку
			tie.pszText = (LPWSTR)L"Tab 1";
			TabCtrl_InsertItem(tabControl, 0, &tie);

			CreateWindow(
				L"RICHEDIT50W",		// Используйте класс Rich Edit
Control
				L"",			// Текст по умолчанию
				WS_BORDER | WS_CHILD | WS_VISIBLE |
WS_VSCROLL | WS_HSCROLL | ES_MULTILINE,
				10, 50, 460, 380,	// Размеры и положение
				hWnd, (HMENU)IDC_EDIT_TAB_START, hInst, nullptr
			);
		}
	}
```

```
        break;
        case WM_NOTIFY:  // переключение вкладок
        {
                NMHDR* pnmhdr = (NMHDR*)lParam;
                if (pnmhdr->code == TCN_SELCHANGE)
                {
                        int currentTab = TabCtrl_GetCurSel(tabControl);

                        // Скрыть все текстовые поля
                        for (int i = 0; i < tabIndexCounter; i++)
                        {
                                ShowWindow(GetDlgItem(hWnd,
IDC_EDIT_TAB_START + i), SW_HIDE);
                        }

                        // Показать текстовое поле для выбранной вкладки
                        ShowWindow(GetDlgItem(hWnd, IDC_EDIT_TAB_START +
currentTab), SW_SHOW);
                }
        }
        break;
        case WM_COMMAND:
        {
                int wmId = LOWORD(wParam); // Извлечение идентификатора
команды
                int wmEvent = HIWORD(wParam);

                if (wmId == IDC_TABCONTROL) {
                        // Обработка событий переключения вкладок
                        int currentTab = TabCtrl_GetCurSel(tabControl);

                        // Скрыть все текстовые поля
                        for (int i = 0; i < tabIndexCounter; i++)
                        {
                                ShowWindow(GetDlgItem(hWnd,
IDC_EDIT_TAB_START + i), SW_HIDE);
                        }

                        // Показать текстовое поле для выбранной вкладки
                        ShowWindow(GetDlgItem(hWnd, IDC_EDIT_TAB_START +
currentTab), SW_SHOW);
```

```
            }
            else
            {
                switch (wmId)
                {
                case IDM_ABOUT:
                    DialogBox(hInst,
MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
                    break;
                case IDM_EXIT:
                    if (MessageBox(hWnd, L"Вы действительно хотите
закрыть окно?", L"Выход", MB_OKCANCEL) == IDOK) {
                        DestroyWindow(hWnd);
                    }
                    return 0;
                    break;

                case ID_32775:  // new tab
                {
                    // Добавление новой вкладки
                        TCITEM tie;
                        tie.mask = TCIF_TEXT;
                        tabIndexCounter++;

                        WCHAR tabText[64];
                        swprintf_s(tabText, L"Tab %d", tabIndexCounter);
                        tie.pszText = tabText;
                        TabCtrl_InsertItem(tabControl, tabIndexCounter - 1,
&tie); // вставка новой вкладки

                        CreateWindow(
                            L"RICHEDIT50W",
                            L"",
                            WS_BORDER | WS_CHILD | WS_VISIBLE |
WS_VSCROLL | WS_HSCROLL | ES_MULTILINE,
                            10, 50, 460, 380,
                            hWnd, (HMENU)(IDC_EDIT_TAB_START +
tabIndexCounter - 1), hInst, nullptr
                        );

                        // Переключение на новую вкладку
```

```
                    TabCtrl_SetCurSel(tabControl, tabIndexCounter - 1);
                    SendMessage(hWnd, WM_COMMAND,
MAKEWPARAM(IDC_TABCONTROL, 0), 0);

                }
                break;

                case ID_32776: // Обработка выбора заднего фона
                {
                    int currentTab = TabCtrl_GetCurSel(tabControl);
                    int controlId = IDC_EDIT_TAB_START + currentTab;
                    HWND textEdit = GetDlgItem(hWnd, controlId);

                    CHOOSECOLOR cc{ 0 };
                    cc.lStructSize = sizeof(cc);
                    cc.hwndOwner = hWnd;
                    cc.lpCustColors = (LPDWORD)acrCustClr;
                    cc.Flags = CC_FULLOPEN | CC_RGBINIT;
                    if (ChooseColor(&cc)) {
                        SendMessage(textEdit, EM_SETBKGNDCOLOR,
FALSE, (LPARAM)cc.rgbResult);
                    }
                }
                break;

                case ID_32777: // Обработка изменения стилей текста
                {
                    int currentTab = TabCtrl_GetCurSel(tabControl);
                    int controlId = IDC_EDIT_TAB_START + currentTab;
                    HWND textEdit = GetDlgItem(hWnd, controlId);

                    CHOOSEFONT cf; // Структура для диалога выбора
шрифта
                    LOGFONT lf; // Структура для хранения информации о
шрифте

                    ZeroMemory(&cf, sizeof(CHOOSEFONT));
                    ZeroMemory(&lf, sizeof(LOGFONT));

                    cf.lStructSize = sizeof(CHOOSEFONT);
                    cf.hwndOwner = hWnd;
```

```
                        cf.lpLogFont = &lf;
                        cf.Flags = CF_SCREENFONTS | CF_EFFECTS |
CF_INITTOLOGFONTSTRUCT;
                        cf.rgbColors = RGB(0, 0, 0); // Начальный цвет текста
(черный)

                        if (ChooseFont(&cf)) // Отображение диалога выбора
шрифта
                        {
                                // Применение выбранного шрифта и цвета текста
                                HFONT hFont = CreateFontIndirect(&lf);
                                SendMessage(textEdit, WM_SETFONT,
(WPARAM)hFont, TRUE);
                                SendMessage(textEdit, EM_SETCHARFORMAT,
SCF_ALL, (LPARAM)&cf);
                        }
                }
                break;
                case ID_32778: // Закрыть последнюю вкладку
                {
                        if (tabIndexCounter > 0)
                        {
                                int currentTabIndex = tabIndexCounter - 1;
                                int lastTabIndex = tabIndexCounter - 1;

                                // Удаляем вкладку и связанное с ней текстовое
поле
                                TabCtrl_DeleteItem(tabControl, lastTabIndex);
                                DestroyWindow(GetDlgItem(hWnd,
IDC_EDIT_TAB_START + lastTabIndex));

                                // Обновляем счетчик вкладок
                                tabIndexCounter--;

                                // Если закрыта текущая вкладка, обновляем
текущую выбранную вкладку

                                if (currentTabIndex >= tabIndexCounter)
                                {
                                        currentTabIndex = tabIndexCounter - 1;
```

```
                                        TabCtrl_SetCurSel(tabControl,
currentTabIndex);
                                        SendMessage(hWnd, WM_COMMAND,
MAKEWPARAM(IDC_TABCONTROL, 0), 0);
                                    }

                                }
                            }
                            break;

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

                            }
                        }
                    }
                    break;
                    case WM_CLOSE:
                        if (MessageBox(hWnd, L"Вы действительно хотите закрыть окно?",
L"Выход", MB_OKCANCEL) == IDOK) {
                            DestroyWindow(hWnd);
                        }
                        return 0;
                    break;
                    case WM_DESTROY:
                        PostQuitMessage(0);
                    break;
                    default: return DefWindowProc(hWnd, message, wParam, lParam);

                    }
                    return 0;
}

// Обработчик сообщений для окна "О программе".
INT_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam,
LPARAM lParam)
{
    UNREFERENCED_PARAMETER(lParam);
    switch (message)
    {
    case WM_INITDIALOG:
```

```
            return (INT_PTR)TRUE;

        case WM_COMMAND:
            if (LOWORD(wParam) == IDOK || LOWORD(wParam) ==
IDCANCEL)
            {
                EndDialog(hDlg, LOWORD(wParam));
                return (INT_PTR)TRUE;
            }
            break;
        }
        return (INT_PTR)FALSE;
}
```