**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

**МОСКОВСКИЙ ЭНЕРГЕТИЧЕСКИЙ ИНСТИТУТ**

**Отчет к лабораторной работе №2**

**Дисциплины «Системное программирование»**

**Выполнил студент группы А-13-22**

**Кунавина Мария**

**Преподаватель: Меньшикова К.Г.**

**Москва 2024**

### Подготовка к лабораторной работе

1. Тип данных TCHAR, переключение режима кодировки символов, функции для работы со строками (strlen, strcpy, strcat, itoa, atoi).

TCHAR - Строка символов Win32, которая позволяет описать строки Юникода, ANSI или DBCS.

Strlen – длина строки

Strcpy – копирование из строки всех символов

Strcat – объединение двух строк

Itoa – из строки в число

Atoi – из числа в строк0443

1. LOWORD, HIWORD.

Например есть число 0x3256abcd

LOWORD – abcd HIWORD - 3256

1. Функция SendMessage

Например, передать сообщение главному окну

SendMessage (hWnd,WM\_LBUTTONDOWN,MK\_LBUTTON,MAKELONG(100,100));

или сообщение для кнопки SendMessage(hBut1,BM\_SETSTATE,TRUE,0);

Отправляет указанное сообщение в окно или окна. Функция SendMessage вызывает оконную процедуру для указанного окна и не возвращает ее до тех пор, пока оконная процедура не обработает сообщение.

Чтобы отправить сообщение и немедленно вернуться к нему, используйте функцию SendMessageCallback или SendNotifyMessage. Чтобы отправить сообщение в очередь сообщений потока и немедленно вернуться, используйте функцию PostMessage или PostThreadMessage.

**Синтаксис**

LRESULT SendMessage(

[in] HWND hWnd, [in] UINT Msg, [in] WPARAM wParam, [in] LPARAM lParam);

**Параметры**

[in] hWnd

Тип: HWND

Дескриптор окна, в окно которого будет получено сообщение. Если этот параметр имеет значение HWND\_BROADCAST ((HWND)0xffff), сообщение отправляется во все окна верхнего уровня в системе, включая отключенные или невидимые нехозяйские окна, перекрывающиеся окна и всплывающие окна; Но сообщение не отправляется в дочерние окна.Отправка сообщений регулируется UIPI. Поток процесса может отправлять сообщения только в очереди сообщений потоков в процессах с меньшим или равным уровнем целостности.

**[in] Msg**

Тип: UINT

Сообщение, которое должно быть отправлено.

Списки сообщений, предоставленных системой, см. в разделе Сообщения, определяемые системой.

**[in] wParam**

Тип: WPARAM

Дополнительная информация, относящаяся к сообщению.

**[in] lParam**

Тип: LPARAM

Дополнительная информация, относящаяся к сообщению.

Возвращаемое значение

Тип: LRESULT

Возвращаемое значение указывает на результат обработки сообщения; Это зависит от отправленного сообщения.

1. Стиль окна: CS\_DBLCLKS, сообщение WM\_LBUTTONDBLCLK.

**CS\_DBLCLKS** - Отправляет сообщение о двойном щелчке в оконную процедуру, когда пользователь дважды щелкает мышью, когда курсор находится в окне, принадлежащем классу.

**WM\_LBUTTONDBLCLK -** Публикуется, когда пользователь дважды щелкает левую кнопку мыши, когда курсор находится в клиентской области окна. Если мышь не захвачена, сообщение будет отправлено в окно под курсором. В противном случае сообщение будет отправлено в окно, в котором была сделана запись мыши.

1. Сообщения от мыши в клиентской и вне клиентской области окна.

### Cообщения мыши в области клиента

Окно получает сообщение мыши клиентской области при возникновении события мыши в клиентской области окна. Система отправляет [**сообщение WM\_MOUSEMOVE**](https://learn.microsoft.com/ru-ru/windows/win32/inputdev/wm-mousemove) в окно, когда пользователь перемещает курсор в клиентской области. Одно из следующих сообщений публикуется, когда пользователь нажимает или отпускает кнопку мыши, когда курсор находится в клиентской области.

Развернуть таблицу

| **Сообщение** | **Значение** |
| --- | --- |
| [**WM\_LBUTTONDBLCLK**](https://learn.microsoft.com/ru-ru/windows/win32/inputdev/wm-lbuttondblclk) | Двойной щелчок левой кнопки мыши. |
| [**WM\_LBUTTONDOWN**](https://learn.microsoft.com/ru-ru/windows/win32/inputdev/wm-lbuttondown) | Была нажата левая кнопка мыши. |
| [**WM\_LBUTTONUP**](https://learn.microsoft.com/ru-ru/windows/win32/inputdev/wm-lbuttonup) | Левая кнопка мыши была отпущена. |
| [**WM\_MBUTTONDBLCLK**](https://learn.microsoft.com/ru-ru/windows/win32/inputdev/wm-mbuttondblclk) | Была дважды щелкнуна средняя кнопка мыши. |
| [**WM\_MBUTTONDOWN**](https://learn.microsoft.com/ru-ru/windows/win32/inputdev/wm-mbuttondown) | Была нажата средняя кнопка мыши. |
| [**WM\_MBUTTONUP**](https://learn.microsoft.com/ru-ru/windows/win32/inputdev/wm-mbuttonup) | Отпущена средняя кнопка мыши. |
| [**WM\_RBUTTONDBLCLK**](https://learn.microsoft.com/ru-ru/windows/win32/inputdev/wm-rbuttondblclk) | Двойной щелчок правой кнопки мыши. |
| [**WM\_RBUTTONDOWN**](https://learn.microsoft.com/ru-ru/windows/win32/inputdev/wm-rbuttondown) | Была нажата правая кнопка мыши. |
| [**WM\_RBUTTONUP**](https://learn.microsoft.com/ru-ru/windows/win32/inputdev/wm-rbuttonup) | Отпущена правая кнопка мыши. |
| [**WM\_XBUTTONDBLCLK**](https://learn.microsoft.com/ru-ru/windows/win32/inputdev/wm-xbuttondblclk) | Двойной щелчок кнопки мыши X. |
| [**WM\_XBUTTONDOWN**](https://learn.microsoft.com/ru-ru/windows/win32/inputdev/wm-xbuttondown) | Нажата кнопка мыши X. |
| [**WM\_XBUTTONUP**](https://learn.microsoft.com/ru-ru/windows/win32/inputdev/wm-xbuttonup) | Кнопка мыши X была отпущена. |

Кроме того, приложение может вызвать функцию [**TrackMouseEvent**](https://learn.microsoft.com/ru-ru/windows/win32/api/winuser/nf-winuser-trackmouseevent) , чтобы система отправляла два других сообщения. Сообщение [**WM\_MOUSEHOVER**](https://learn.microsoft.com/ru-ru/windows/win32/inputdev/wm-mousehover) публикуется при наведении курсора на клиентную область в течение определенного периода времени. Он публикует [**сообщение WM\_MOUSELEAVE**](https://learn.microsoft.com/ru-ru/windows/win32/inputdev/wm-mouseleave) , когда курсор покидает клиентную область.

**Сообщения мыши с неклиентской областью**

Окно получает сообщение мыши о неклиентной области, когда событие мыши происходит в любой части окна, кроме клиентской области. Неклиентная область окна состоит из его границы, строки меню, строки заголовка, полосы прокрутки, меню окна, кнопки свернуть и кнопки развернуть.

Система создает сообщения неклиентной области в основном для собственного использования. Например, система использует сообщения неклиентной области для изменения курсора на двуглавую стрелку при перемещении курсора в границу окна. Чтобы воспользоваться преимуществами встроенного интерфейса мыши, окно должно передавать сообщения мыши в неклиентской области в функцию [**DefWindowProc**](https://learn.microsoft.com/ru-ru/windows/desktop/api/winuser/nf-winuser-defwindowproca) .

Для каждого сообщения мыши клиентской области имеется соответствующее сообщение мыши неклиентской области. Имена этих сообщений похожи, за исключением того, что именованные константы для сообщений неклиентской области содержат буквы NC. Например, при перемещении курсора в неклиентной области создается [**сообщение WM\_NCMOUSEMOVE**](https://learn.microsoft.com/ru-ru/windows/win32/inputdev/wm-ncmousemove) , а нажатие левой кнопки мыши, когда курсор находится в неклиентной области, создает [**сообщение WM\_NCLBUTTONDOWN**](https://learn.microsoft.com/ru-ru/windows/win32/inputdev/wm-nclbuttondown) .

1. Сообщение WM\_NCHITTEST, процедура DefWindowProc.

Отправляется в окно для того, чтобы определить, какая часть окна соответствует той или иной координате экрана. Это может произойти, например, при перемещении курсора, при нажатии или отпускании кнопки мыши или в ответ на вызов такой функции, как [**WindowFromPoint**](https://learn.microsoft.com/en-us/windows/desktop/api/winuser/nf-winuser-windowfrompoint). Если мышь не захвачена, сообщение отправляется в окно, расположенное под курсором. В противном случае сообщение будет отправлено в окно, в котором была сделана запись мыши.

Окно получает это сообщение через функцию [**WindowProc**](https://learn.microsoft.com/en-us/windows/win32/api/winuser/nc-winuser-wndproc).

Язык С++Копировать

#define WM\_NCHITTEST 0x0084

## Параметры

wParam

Этот параметр не используется.

lParam

Младшее слово определяет x-координату курсора. Координата указывается относительно левого верхнего угла экрана.

В старшем слове указывается y-координата курсора. Координата указывается относительно левого верхнего угла экрана.

1. Функции SetCapture(hWnd); ReleaseCapture();

**SetCapture(hWnd)** - Устанавливает захват мыши на указанное окно, принадлежащее текущему потоку. **SetCapture** захватывает ввод с помощью мыши, когда мышь находится над окном захвата, или когда кнопка мыши была нажата, когда мышь находится над окном захвата, но кнопка все еще не нажата. Только одно окно за раз может захватывать мышь.

Если курсор мыши находится над окном, созданным другим потоком, система направит ввод мышью в указанное окно только в том случае, если кнопка мыши нажата.

**ReleaseCapture() -** Освобождает захват мыши из окна в текущем потоке и восстанавливает нормальную обработку ввода с помощью мыши. Окно, захватившее мышь, получает все вводимые данные мышью, независимо от положения курсора, за исключением случаев, когда кнопка мыши нажата, когда горячая точка курсора находится в окне другого потока.

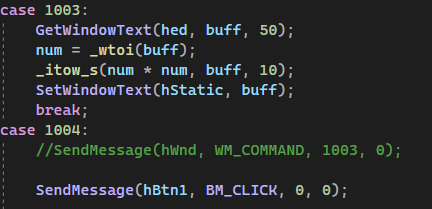
1. Сообщения WM\_CLOSE, WM\_DESTROY.

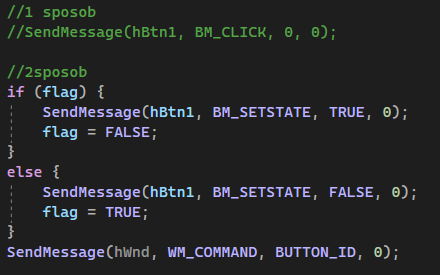
Сообщение **WM\_CLOSE** дает возможность предложить пользователю перед закрытием окна. Если вы действительно хотите закрыть окно, вызовите функцию [**DestroyWindow**](https://learn.microsoft.com/ru-ru/windows/desktop/api/winuser/nf-winuser-destroywindow) . В противном случае просто верните ноль из **сообщения WM\_CLOSE** , и операционная система проигнорирует сообщение и не уничтожит окно.

Когда окно будет уничтожено, оно получает [**WM\_DESTROY**](https://learn.microsoft.com/ru-ru/windows/desktop/winmsg/wm-destroy) сообщение. Это сообщение отправляется после удаления окна с экрана, но перед уничтожением (в частности, перед уничтожением дочерних окон).

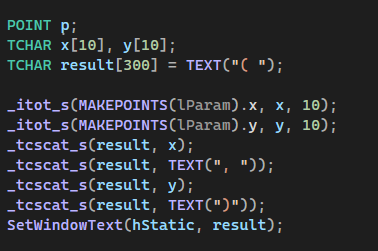
### Задание

1. Создать приложение Win32 Project (в Microsoft Visual Studio). Запустив приложение, убедиться в том, что оно работает, а окно с помощью мыши перемещается по экрану (курсор мыши находится в заголовке окна).
2. В окно приложения добавить две кнопки, поля ввода и вывода. Первая кнопка должна возводить в квадрат целое число введенное пользователем, вторая кнопка должна заставить первую кнопку “нажаться” и выполнить код (предусмотреть два способа «нажатия», т.е. два варианта команд для управления первой кнопкой).





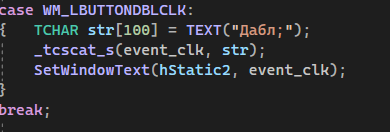
1. По щелчку на правой кнопке мыши определить и вывести координаты курсора.

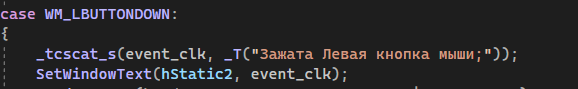


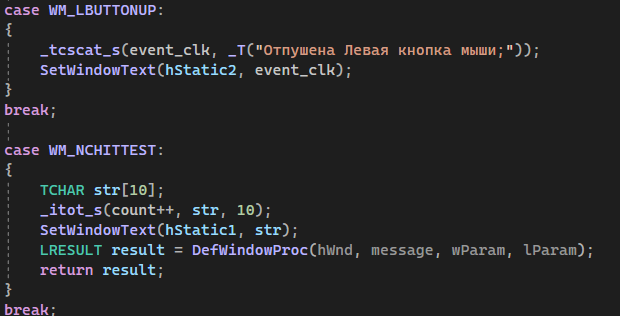
1. Проделать программные эксперименты и проследить за работой событий (для эксперимента можно записывать информацию о событии в строковую переменную, а затем вывести ее в окне сообщений):

* WM\_LBUTTONDBLCLK: какие события и сколько раз срабатывают при двойном щелчке ( при этом учитывать события: WM\_LBUTTONDBLCLK, WM\_LBUTTONDOWN, WM\_LBUTTONUP );
* WM\_NCHITTEST: сколько раз возникает данное событие при работе с мышью.





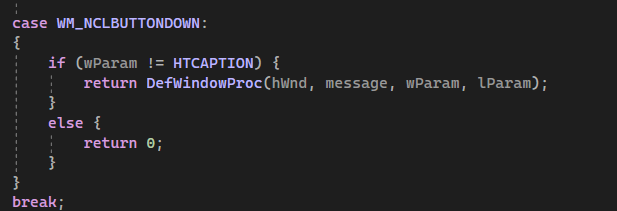
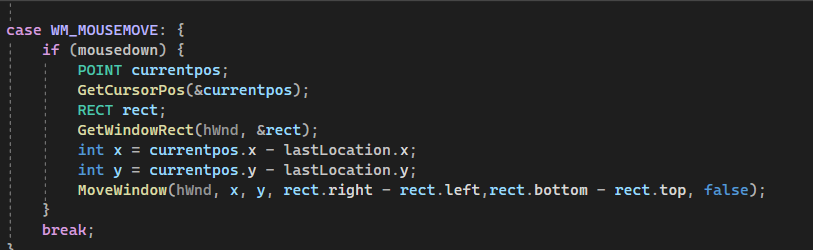
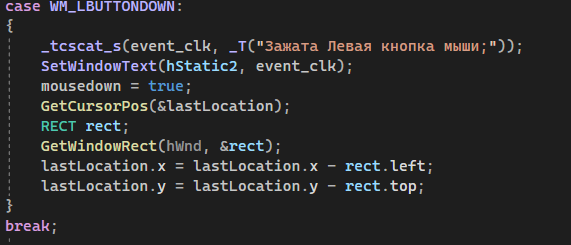




1. Предусмотреть в программе возможность для пользователя предотвратить закрытие приложения. Проверить: можно ли предотвратить закрытие при получении события WM\_DESTROY.

**ТАК ЛУЧШЕ НЕ ДЕЛАТЬ!**

1. Разработать программный код, который позволит пользователю с помощью мыши перемещать окно (мышь при этом должна нажиматься и перемещаться в клиентской области окна, а программист должен отлавливать и обрабатывать только ее сообщения). Запретить при этом перемещение окна за заголовок.



1. Сохранить разработанное приложение так, чтобы его можно было использовать в программных экспериментах следующей лабораторной работы.

**Cохранено в папке C:\Users\gagag\Desktop\3 курс\Системное программирование 2024\Лабы\2\WindowsProject2\x64**

**Листинг:**

#include "framework.h"

#include "WindowsProject2.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

HWND hStatic2; // лог событий мышки

HWND hStatic1; // поле вывода количества событий мыши

TCHAR event\_clk[500]; // для записи событий

int count = 0;

// Отправить объявления функций, включенных в этот модуль кода:

ATOM MyRegisterClass(HINSTANCE hInstance);

BOOL InitInstance(HINSTANCE, int);

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

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

int APIENTRY wWinMain(\_In\_ HINSTANCE hInstance,

\_In\_opt\_ HINSTANCE hPrevInstance,

\_In\_ LPWSTR lpCmdLine,

\_In\_ int nCmdShow)

{

UNREFERENCED\_PARAMETER(hPrevInstance);

UNREFERENCED\_PARAMETER(lpCmdLine);

// TODO: Разместите код здесь.

// Инициализация глобальных строк

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

LoadStringW(hInstance, IDC\_WINDOWSPROJECT2, szWindowClass, MAX\_LOADSTRING);

MyRegisterClass(hInstance);

// Выполнить инициализацию приложения:

if (!InitInstance(hInstance, nCmdShow))

{

return FALSE;

}

HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC\_WINDOWSPROJECT2));

MSG msg;

// Цикл основного сообщения:

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

{

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

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

return (int)msg.wParam;

}

//

// ФУНКЦИЯ: MyRegisterClass()

//

// ЦЕЛЬ: Регистрирует класс окна.

//

ATOM MyRegisterClass(HINSTANCE hInstance)

{

WNDCLASSEXW wcex;

wcex.cbSize = sizeof(WNDCLASSEX);

wcex.style = CS\_HREDRAW | CS\_VREDRAW | CS\_DBLCLKS;

wcex.lpfnWndProc = WndProc;

wcex.cbClsExtra = 0;

wcex.cbWndExtra = 0;

wcex.hInstance = hInstance;

wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI\_WINDOWSPROJECT2));

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

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

wcex.lpszMenuName = MAKEINTRESOURCEW(IDC\_WINDOWSPROJECT2);

wcex.lpszClassName = szWindowClass;

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

return RegisterClassExW(&wcex);

}

//

// ФУНКЦИЯ: InitInstance(HINSTANCE, int)

//

// ЦЕЛЬ: Сохраняет маркер экземпляра и создает главное окно

//

// КОММЕНТАРИИ:

//

// В этой функции маркер экземпляра сохраняется в глобальной переменной, а также

// создается и выводится главное окно программы.

//

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;

}

//

// ФУНКЦИЯ: WndProc(HWND, UINT, WPARAM, LPARAM)

//

// ЦЕЛЬ: Обрабатывает сообщения в главном окне.

//

// WM\_COMMAND - обработать меню приложения

// WM\_PAINT - Отрисовка главного окна

// WM\_DESTROY - отправить сообщение о выходе и вернуться

//

//

#define EDIT\_ID 1001

#define STATIC\_ID 1002

#define STATIC\_ID1 1005

#define STATIC\_ID2 1006

#define BUTTON\_ID 1003

#define BUTTON1\_ID 1004

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

{

static HWND hed;

static HWND hStatic;

static HWND hStatic1;

static HWND hStatic2;

static HWND hBtn1;

static HWND hBtn2;

int len = 0;

int num;

TCHAR buff[100] = L"";

switch (message)

{

case WM\_CREATE:

{ hBtn1 = CreateWindow(TEXT("BUTTON"), TEXT("Первая"), WS\_VISIBLE | WS\_CHILD, 150, 200, 80, 26, hWnd, (HMENU)BUTTON\_ID, NULL, NULL);

hBtn2 = CreateWindow(TEXT("BUTTON"), TEXT("Вторая"), WS\_VISIBLE | WS\_CHILD, 250, 200, 80, 26, hWnd, (HMENU)BUTTON1\_ID, NULL, NULL);

hed = CreateWindow(TEXT("EDIT"), TEXT("Tra-la-la"), WS\_VISIBLE | WS\_CHILD, 150, 240, 80, 26, hWnd, (HMENU)EDIT\_ID, NULL, NULL);

hStatic = CreateWindow(TEXT("STATIC"), TEXT("Вывод квадрата"), WS\_CHILD | WS\_VISIBLE | ES\_RIGHT, 250, 240, 150, 26, hWnd, (HMENU)STATIC\_ID, NULL, NULL);

hStatic1 = CreateWindow(TEXT("STATIC"), TEXT("Вывод количества возникающих событий"), WS\_CHILD | WS\_VISIBLE | ES\_RIGHT, 250, 300, 300, 40, hWnd, (HMENU)STATIC\_ID1, NULL, NULL);

hStatic2 = CreateWindow(TEXT("STATIC"), TEXT("События"), WS\_CHILD | WS\_VISIBLE | ES\_RIGHT, 250, 400, 500, 100, hWnd, (HMENU)STATIC\_ID2, NULL, NULL);

if (hed != NULL) SetWindowLong(hed, GWL\_ID, 10);

if (hStatic != NULL) SetWindowLong(hStatic, GWL\_ID, 11);

break;

}

case WM\_RBUTTONUP:

{

POINT p;

if (GetCursorPos(&p))

{

TCHAR x[10], y[10];

\_itot\_s(p.x, x, 10);

\_itot\_s(p.y, y, 10);

TCHAR result[30] = TEXT("( ");

\_tcscat\_s(result, x);

\_tcscat\_s(result, TEXT(", "));

\_tcscat\_s(result, y);

\_tcscat\_s(result, TEXT(")"));

SetWindowText(hStatic, result);

}

}

break;

case WM\_LBUTTONDBLCLK:

{ TCHAR str[100] = TEXT("Дабл;");

\_tcscat\_s(event\_clk, str);

SetWindowText(hStatic2, event\_clk);

}

break;

case WM\_LBUTTONDOWN:

{

\_tcscat\_s(event\_clk, \_T("Зажата Левая кнопка мыши;"));

SetWindowText(hStatic2, event\_clk);

SendMessage(hWnd, WM\_SYSCOMMAND, HTCLIENT | SC\_MOVE, 0);

SendMessage(hWnd, WM\_LBUTTONUP, 0, 0);

}

break;

case WM\_LBUTTONUP:

{

\_tcscat\_s(event\_clk, \_T("Отпушена Левая кнопка мыши;"));

SetWindowText(hStatic2, event\_clk);

}

break;

case WM\_NCHITTEST:

{

TCHAR str[10];

\_itot\_s(count++, str, 10);

SetWindowText(hStatic1, str);

LRESULT result = DefWindowProc(hWnd, message, wParam, lParam);

return result;

}

break;

case WM\_NCLBUTTONDOWN:

{

if (wParam != HTCAPTION) {

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

}

else {

return 0;

}

}

break;

case WM\_COMMAND:

{

int wmId = LOWORD(wParam);

// Разобрать выбор в меню:

switch (wmId)

{

case 1003:

GetWindowText(hed, buff, 50);

num = \_wtoi(buff);

\_itow\_s(num \* num, buff, 10);

SetWindowText(hStatic, buff);

break;

case 1004:

//SendMessage(hWnd, WM\_COMMAND, 1003, 0);

SendMessage(hBtn1, BM\_CLICK, 0, 0);

break;

case IDM\_ABOUT:

DialogBox(hInst, MAKEINTRESOURCE(IDD\_ABOUTBOX), hWnd, About);

break;

case IDM\_EXIT:

DestroyWindow(hWnd);

break;

default:

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

}

}

break;

case WM\_PAINT:

{

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hWnd, &ps);

// TODO: Добавьте сюда любой код прорисовки, использующий HDC...

EndPaint(hWnd, &ps);

}

break;

case WM\_CLOSE:

if (MessageBoxA(NULL, "Вы дейтсвительно хотите закрыть окно?", "Закрытие", MB\_OK | MB\_OKCANCEL | MB\_ICONERROR) == IDOK) {

DestroyWindow(hWnd);

}

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;

}