|  |
| --- |
| **МИНОБРНАУКИ РОССИИ**  федеральное государственное бюджетное образовательное  учреждение высшего образования  **«Национальный исследовательский университет «МЭИ»** |

|  |  |
| --- | --- |
| **Институт** | ИВТИ |
| **Кафедра** | ПМИИ |

|  |  |
| --- | --- |
|  |  |
| **Предмет: «Системное программирование»**  **Отчет по лабораторной работе №2**  **Выполнил: студент группы А-13а-19**  **Рамазанов Н. М.**  **Преподаватель: Меньшикова К. Г.**  **Москва, 2021** | |

**1. Краткое описание функций, параметров и используемых событий.**

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

TCHAR – символьный тип данных, аналог char и wchar\_t. Если определен как Unicode, то тип WCHAR, иначе CHAR. Этот тип объявлен в Winnt.h как показано ниже:

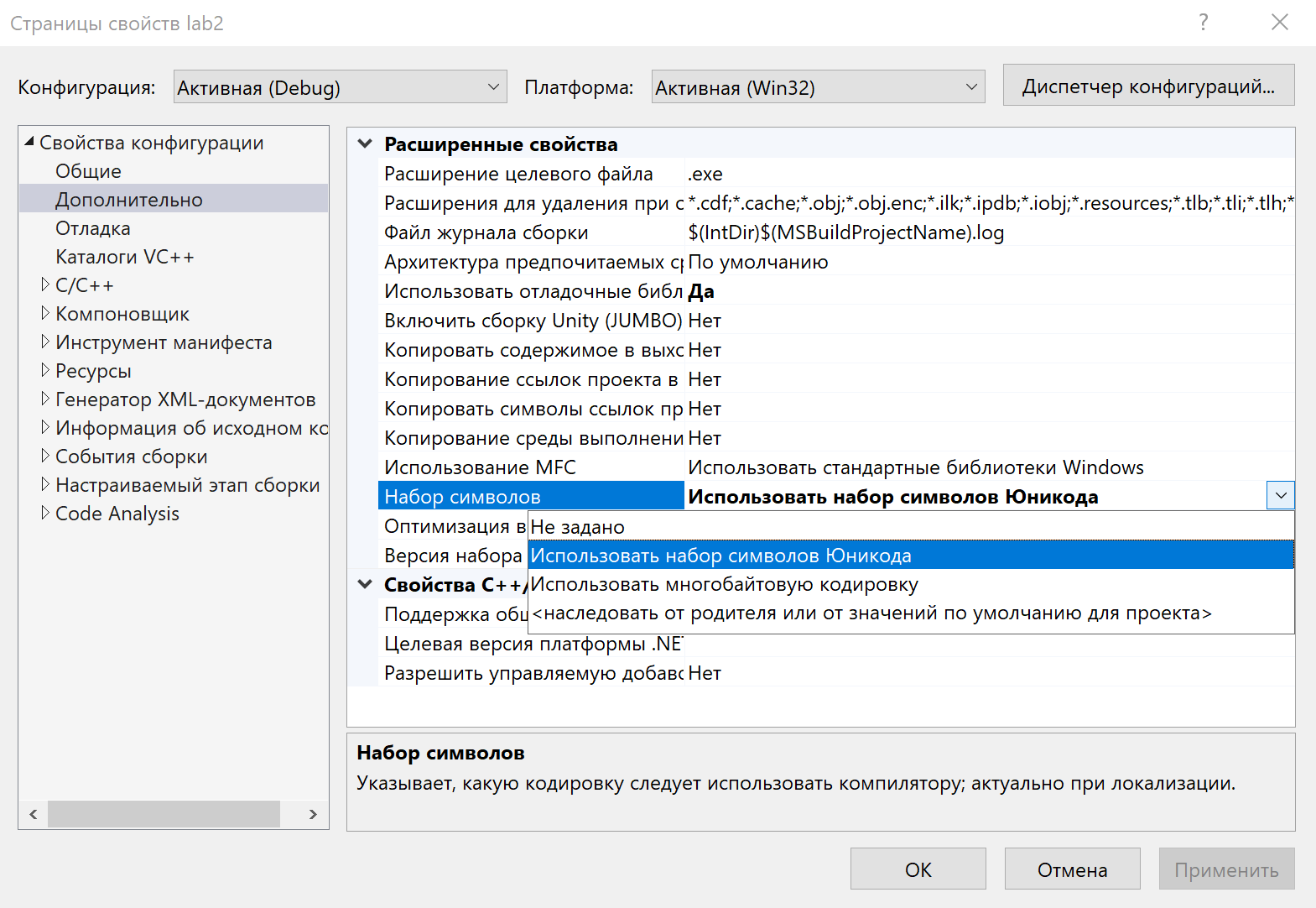
#ifdef UNICODE

typedef WCHAR TBYTE;

#else

typedef unsigned char TBYTE;

Переключение режима кодировки символов:



Функции для работы со строками:

- size\_t strlen (const char \* str);

Возвращает длину строки str. Длина строки определяется завершающим нулевым символом: длина строки C равна количеству символов между началом строки и завершающим нулевым символом (без включения самого завершающего нулевого символа).

- char \* strcpy (char \* destination, const char \* source);

Копирует строку, указанную source, в массив, указанный параметром destination, включая завершающий нулевой символ (и остановку в этой точке). Чтобы избежать переполнения, размер массива, на который указывает пункт назначения, должен быть достаточно длинным, чтобы содержать ту же строку, что и source (включая завершающий нулевой символ), и не должен перекрываться в памяти с source.

- char \* strcat (char \* destptr, const char \* srcptr);

Добавляет копию строки srcptr в конец строки destptr. Нулевой символ конца строки destptr заменяется первым символом строки srcptr, и новый нуль-символ добавляется в конец уже новой строки, сформированной объединением символов двух строк в строке destptr.

- char \* itoa (int value, char \* string, int radix);

Преобразует целое число value в строку string в формате radix. К цифрам числа value подбираются ANSI-символы типа char и записываются в строку string. Поддерживается только Visual Studio и в других IDE не имеет описания.

- int \* atoi (const char \* string);

Возвращает целое число; если в строке не было найдено целое число, функция возвращает нулевое значение.

2) LOWORD, HIWORD.

WORD LOWORD (DWORD dwValue);

Параметр: dwValue - определяет значение, которое будет преобразовано.

Макрокоманда **LOWORD** извлекает младшее слово из заданной величины.

WORD HIWORD (DWORD dwValue);

Параметр: dwValue - определяет значение, которое будет преобразовано.

Макрокоманда **HIWORD** извлекает старшее слово из данного 32-разрядного значения.

3) Функция SendMessage.

LRESULT SendMessage (HWND hWnd, UINT Msg, WPARAM wParam,

LPARAM lParam);

Параметры: hWnd - дескриптор окна, оконная процедура которого примет сообщение; Msg - определяет сообщение, которое будет отправлено; wParam, lParam - определяют дополнительную конкретизирующую сообщение информацию.

Функция SendMessage отправляет заданное сообщение окну или окнам. Функция вызывает оконную процедуру для заданного окна и не возвращает значение до тех пор, пока оконная процедура не обработает сообщение.

4) Стиль окна: CS\_DBLCLKS, сообщение WM\_LBUTTONDBLCLK.

Стиль CS\_DBLCLKS используется при необходимости отслеживать двойные щелчки мышью. Он отправляет сообщение о двойном щелчке оконной процедуре, когда курсор находится внутри окна, принадлежащего классу. При этом в функцию окна посылаются сообщения WM\_LBUTTONDBLCLK и WM\_RBUTTONDBLCLK.

Сообщение WM\_LBUTTONDBLCLK помещается в очередь, если пользователь дважды щелкает левой кнопкой мыши, в то время, когда курсор находится в рабочей области окна. Если мышь не захвачена, сообщение помещается в окно под курсором. В противном случае, сообщение помещается в окно, которое захватило мышь.

5) Сообщение WM\_NCHITTEST, процедура DefWindowProc.

Сообщение WM\_NCHITTEST посылается окну, когда курсор передвигается, или когда нажата или отжата кнопка мыши. Если мышь не захвачена, это сообщение посылается окну под курсором. Иначе, сообщение посылается окну, которое захватило мышь.

xPos = LOWORD(lParam); // горизонтальная позиция курсора

yPos = HIWORD(lParam); // вертикальная позиция курсора

Параметр lParam сообщения WM\_NCHITTEST содержит экранные координаты острия курсора. Процедура DefWindowProc проверяет эти координаты и возвращает значение местоположения курсора, которое указывает место острия. Значение местоположения курсора может быть одним из множества значений.

LRESULT DefWindowProc (HWND hWnd, UINT Msg, WPARAM wParam,

LPARAM lParam);

Параметры: hWnd - дескриптор оконной процедуры, которая получает сообщение; Msg - определяет сообщение, которое будет отправлено; wParam, lParam - определяют дополнительную конкретизирующую сообщение информацию.

6) SetCapture(hWnd); ReleaseCapture();

HWND SetCapture (HWND hWnd);

Параметр: hWnd - дескриптор окна в текущем потоке, который должен захватить мышь.

Функция SetCapture устанавливает захват мыши в заданном окне, принадлежащем текущему потоку. SetCapture захватывает ввод данных от мыши или когда мышь находится над захватывающим окном, или когда нажималась кнопка мыши, в то время, когда мышь была над захватывающим окном, а кнопка все еще находилась в нажатом состоянии. Только одно окно одновременно может захватить мышь.

Если курсор мыши находится над окном, созданным другим потоком, система направит ввод данных от мыши в указанное окно, только в том случае, если кнопка мыши находится в нажатом состоянии.

Возвращаемое значение - дескриптор окна, которое перед этим захватило мышь. Если такого окна нет, возвращаемое значение - ПУСТО (NULL).

BOOL ReleaseCapture (VOID);

Функция ReleaseCapture освобождает захват мыши окном в текущем потоке и восстанавливает обычную обработку ввода данных от мыши. Окно, которое захватило мышь, получает весь ввод данных от мыши, независимо от позиции курсора, кроме тех случаев, когда кнопкой мыши щелкают в то время, когда острие курсора находится в окне другого потока.

Если функция завершается успешно, возвращаемое значение не нуль.

Если функция завершается ошибкой, возвращаемое значение нуль.

**2. Выполнение работы.**

1. Создал приложение Win32 Project (в Microsoft Visual Studio). Запустив приложение, убедился в том, что оно работает, а окно с помощью мыши перемещается по экрану (курсор мыши находится в заголовке окна).
2. В окно приложения добавил две кнопки, поля ввода и вывода. Маленькая кнопка возводит в квадрат целое число, введенное пользователем, большая кнопка заставляет маленькую кнопку “нажаться” и выполнить код (предусмотрел два возможных варианта воздействия на маленькую кнопку).

HWND Btn1; // большая кнопка

HWND Btn2; // малая кнопка

HWND InputTxt; // поле ввода

HWND OutputTxt; // поле вывода

const static int idBtn1 = 1; // идентификаторы кнопок и полей

const static int idBtn2 = 2;

const static int idInputTxt = 3;

const static int idOutputTxt = 4;

Btn1 = CreateWindow(L"BUTTON", L"Запустить программу", WS\_VISIBLE | WS\_CHILD | BS\_PUSHBUTTON, 250, 150, 300, 50, hWnd, (HMENU)idBtn1, hInstance, NULL);

Btn2 = CreateWindow(L"BUTTON", L"Возвести в квадрат", WS\_VISIBLE | WS\_CHILD | BS\_PUSHBUTTON, 280, 100, 240, 25, hWnd, (HMENU)idBtn2, hInstance, NULL);

InputTxt = CreateWindow(L"EDIT", L"", WS\_VISIBLE | WS\_CHILD | WS\_BORDER, 175, 25, 200, 50, hWnd, (HMENU)idInputTxt, hInstance, NULL);

OutputTxt = CreateWindow(L"STATIC", L"", WS\_VISIBLE | WS\_CHILD, 425, 25, 200, 50, hWnd, (HMENU)idOutputTxt, hInstance, NULL);

case WM\_COMMAND:

{

int wmId = LOWORD(wParam);

// Разобрать выбор в меню:

switch (wmId)

{

case idBtn1: // нажатие на большую кнопку - заставляем нажаться маленькую

{

// 1 способ

//SendMessage(hWnd, WM\_COMMAND, idBtn2, 0);

// 2 способ

SendMessage(Btn2, BM\_CLICK, 0, 0);

}

break;

case idBtn2: // нажатие на маленькую кнопку - возводим число в квадрат

{

TCHAR InputTxtContent[MAX\_LOADSTRING];

int x;

GetWindowText(InputTxt, InputTxtContent, 100);

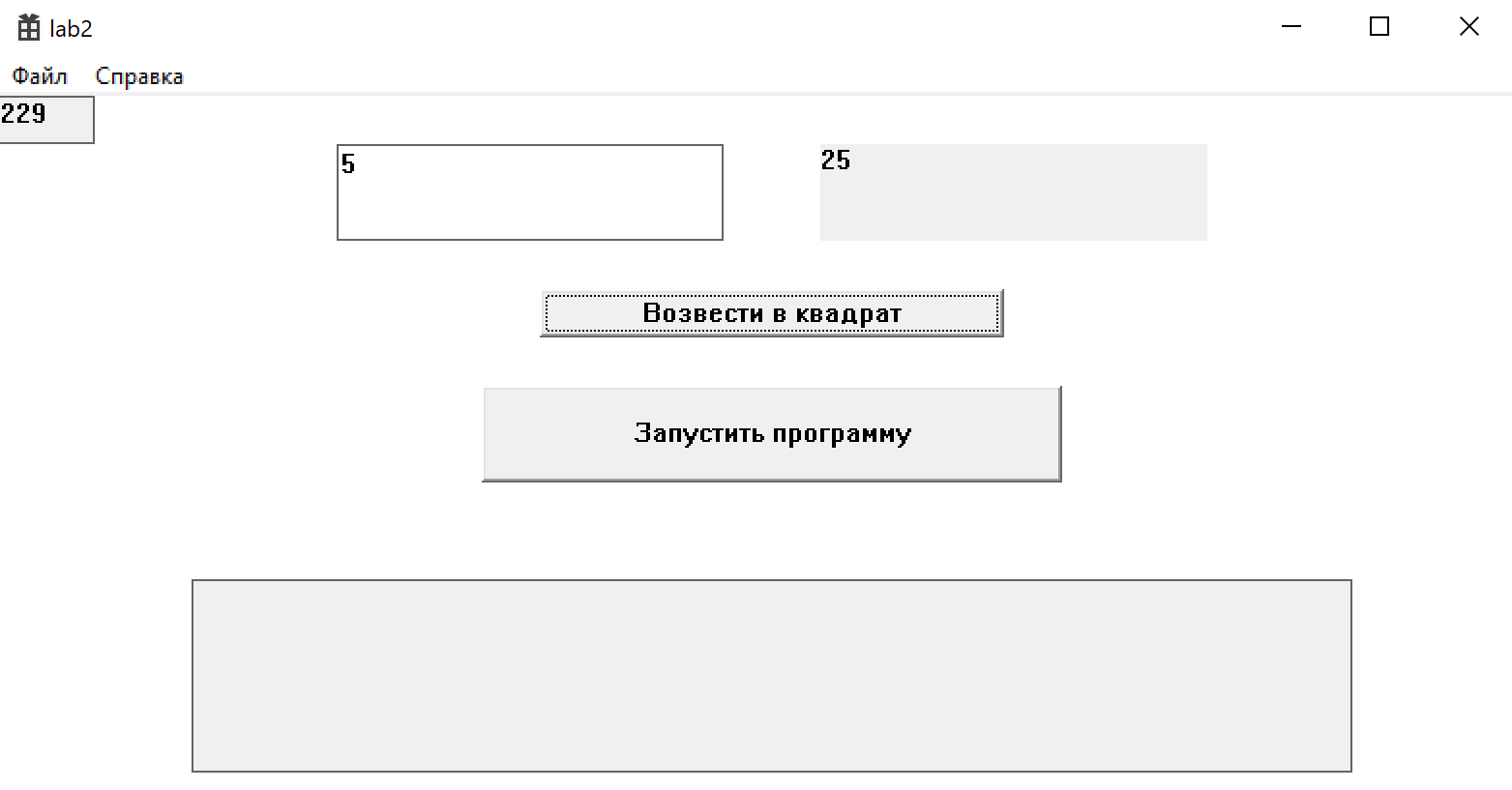
x = \_tstoi(InputTxtContent);

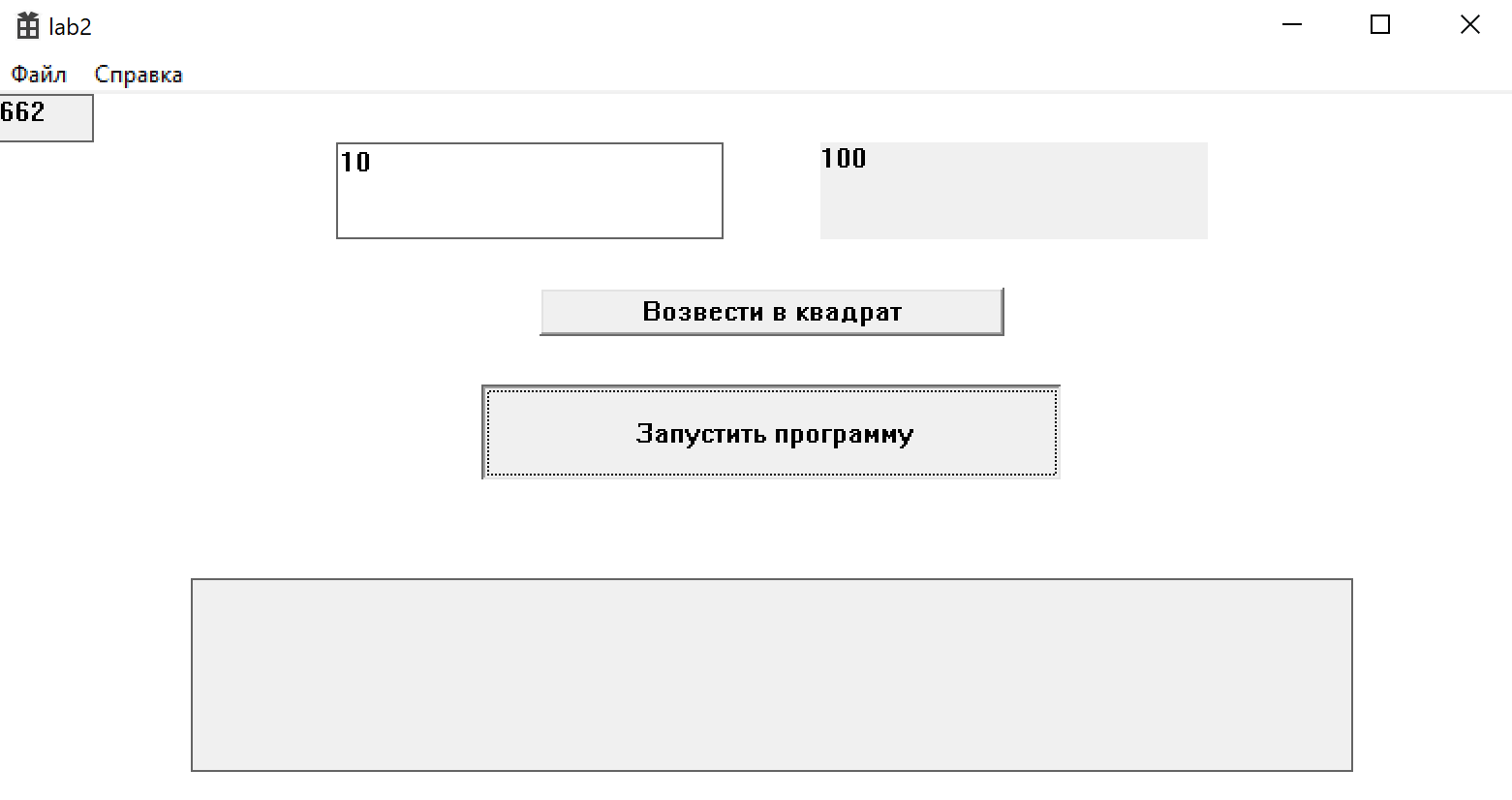
x = x \* x;

\_itot\_s(x, InputTxtContent, \_countof(InputTxtContent), 10);

SetWindowText(OutputTxt, InputTxtContent);

}





1. По щелчку на правой кнопке мыши определил и вывел координаты курсора.

case WM\_RBUTTONDOWN:

{

POINTS MousePos = MAKEPOINTS(lParam);

TCHAR strx[10], stry[10];

\_itot\_s(MousePos.x, strx, 10);

\_itot\_s(MousePos.y, stry, 10);

TCHAR res[30] = TEXT("X: ");

\_tcscat\_s(res, strx);

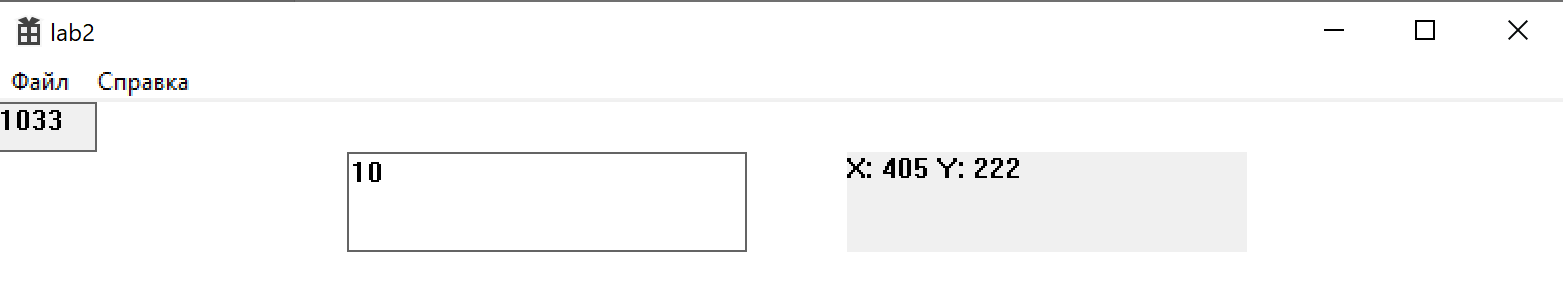
\_tcscat\_s(res, TEXT(" Y: "));

\_tcscat\_s(res, stry);

SetWindowText(OutputTxt, res);

}

break;



1. Проделал программные эксперименты и проследил за работой событий (для эксперимента записываю информацию о событиях в строковую переменную, а затем вывожу ее в окне сообщений):

* WM\_LBUTTONDBLCLK: при двойном щелчке сначала возникают события WM\_LBUTTONDOWN и WM\_LBUTTONUP, далее возникает непосредственно событие двойного щелчка левой кнопкой мыши WM\_LBUTTONDBLCLK, а затем снова WM\_LBUTTONUP);
* WM\_NCHITTEST: для подсчета, сколько раз возникает данное событие при работе с мышью, создал поле для вывода кол-ва.

HWND LogTxt; // вывод событий

HWND MouseTracker; // поле для вывода кол-ва изменений состояния мыши

TCHAR EventList[1000]; // строковая переменная

const static int idLogTxt = 5;

const static int idMouseTracker = 6;

int Counter = 0;

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

LogTxt = CreateWindow(L"STATIC", L"", WS\_VISIBLE | WS\_CHILD | WS\_BORDER, 100, 250, 600, 100, hWnd, (HMENU)idLogTxt, hInstance, NULL);

MouseTracker = CreateWindow(L"STATIC", L"", WS\_VISIBLE | WS\_CHILD | WS\_BORDER, 0, 0, 50, 25, hWnd, (HMENU)idMouseTracker, hInstance, NULL);

case WM\_LBUTTONDOWN:

\_tcscat\_s(EventList, \_T("Нажата ЛКМ! "));

SetWindowText(LogTxt, EventList);

break;

case WM\_LBUTTONUP:

\_tcscat\_s(EventList, \_T("Отпущена ЛКМ! "));

SetWindowText(LogTxt, EventList);

break;

case WM\_LBUTTONDBLCLK:

\_tcscat\_s(EventList, \_T("Двойной щелчок ЛКМ! "));

SetWindowText(LogTxt, EventList);

break;

case WM\_NCHITTEST:

TCHAR str[10];

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

SetWindowText(MouseTracker, str);

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

break;



1. Разработал программный код, который позволяет пользователю с помощью мыши перемещать окно, но мышь при этом нажимается и перемещается в клиентской области окна (также предусмотрел запрет переноса окна за заголовок).

case WM\_LBUTTONDOWN:

...

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

SendMessage(hWnd, WM\_LBUTTONUP, 0, 0); // нужно чтобы вызывать событие отжатия ЛКМ, иначе из-за строки выше оно не приходит

break;

case WM\_NCLBUTTONDOWN:

if (wParam == HTCAPTION)

return 0;

else

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

break;

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

Полный код программы доступен по ссылке: <https://github.com/programzan/SystemProgramming/blob/lab2/lab2.cpp>

Здесь же размещены все сопутствующие файлы: <https://github.com/programzan/SystemProgramming/tree/lab2>