

Лабораторная работа №2

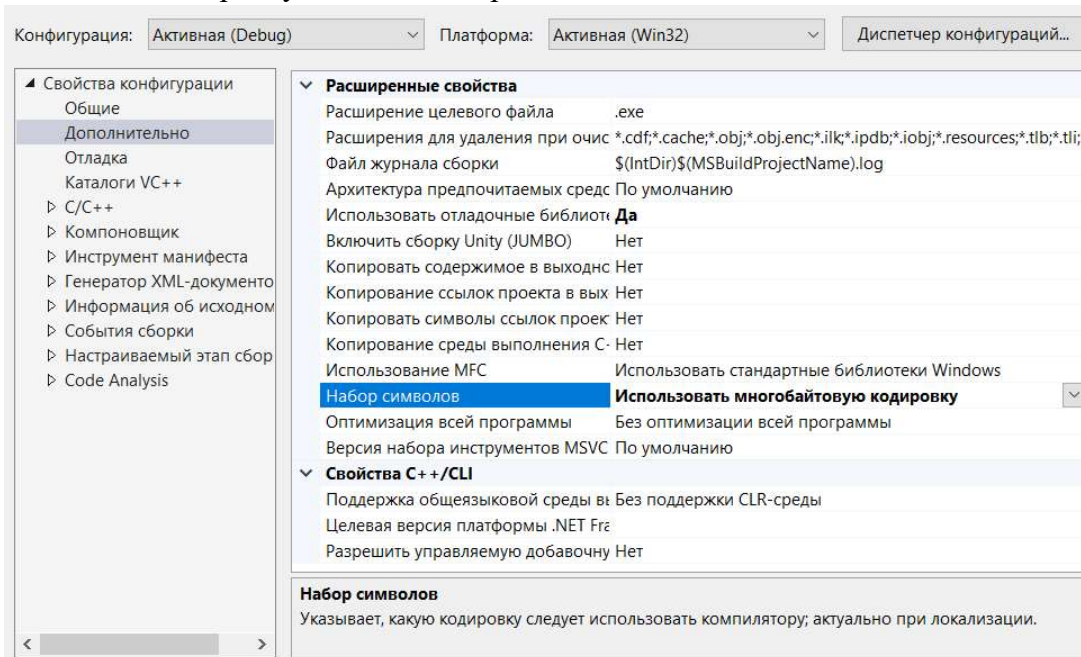
Обработка сообщений

А-13а-19 Кутдусов Р.К.

Подготовка к лабораторной работе

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

- Есть два вида кодировок символов: *ANSI* и *UNICODE*. Однобайтные символы относятся к *ANSI*, двухбайтные — к кодировке *UNICODE*. Можно с подключить (или отключить) *UNICODE* кодировку в свойствах проекта.



И тогда в коде создать переменную типа `char` в *UNICODE* можно будет так:

```
wchar_t str[10];
```

Если же мы хотим использовать кодировку *ANSI*, то мы традиционно напишем:

```
char str[10];
```

В *WINAPI*, в зависимости от того, подключён юникод или нет, используются два вида строк *UNICODE*-ные или *TCHAR*-ные.

TCHAR – символьный тип — аналог `char` и `wchar_t`.

- `strlen`, `strcpy`, `strcat`, `itoa`, `atoi` – одни из основных функций, предназначенных для работы со строками. Большинство прототипов этих функций находится в заголовочном файле `<string.h>`.
 - `int strlen(const char *s);` - Возвращает длину строки `s` - количество символов, предшествующих нулевому символу. `wcslen` использовать для `wchar_t` (юникода).
 - `char *strcpy (char *dst, const char *src);` `dst` — указатель на буфер назначения. `src` — указатель на исходную строку. Копировать одну строку в другую. Аналогичная функция `strncpy()` ограничивает количество копируемых символов (`n`). Источник (`source`) и приемник (`destination`) являются указателями типа `char *` или символьными массивами.
 - `char *strcat(char *dest, const char *scr);` - Объединяет исходную строку `scr` и результирующую строку `dest`, присоединяя первую к последней. Возвращает `dest`.
 - `char *itoa(int value, char *s, int radix);` - Преобразует значение целого типа `value` в строку `s`. Возвращает указатель на результирующую строку. Значение

radix - основание системы счисления, используемое при преобразовании (от 2 до 36). Заголовочный файл - `<stdlib.h>`.

- `int atoi(const char *s);` - Преобразует строку `s` в число типа `int`. Возвращает значение или нуль, если строку преобразовать нельзя. Заголовочный файл - `<stdlib.h>`.

2. *LOWORD, HIWORD.*

Часто используется приём, который позволяет хранить в одной переменной сразу два значения. Это позволяет, к примеру, при послыке сообщения окну с помощью функций `SendMessage` или `PostMessage`, передать сразу четыре параметра в то время, как эти функции поддерживают передачу только двух параметров. Реализуется это путём записи в 32-битовую переменную (длинное целое) двух 16-битовых значений (короткое целое, не превышающее `0xffff`). В языке программирования C для создания такого числа применяется макрос `MAKELONG`, а для извлечения значений из старших и младших 16-ти разрядов, соответственно, `HIWORD` (high word) и `LOWORD` (low word).

3. Функция `SendMessage`.

Для организации немедленной обработки сообщений следует использовать функцию:

```
LONG SendMessage(HWND, Msg, WPARAM, LPARAM)
```

Функция вызывает оконную процедуру и не возвращает управление до тех пор, пока сообщение не будет обработано (возвращаемое значение отражает результат обработки и зависит от типа сообщения).

4. Стилль окна `CS_DBLCLKS`, сообщение `WM_LBUTTONDOWNBLCLK`.

- Стилль `CS_DBLCLKS` используется при необходимости отслеживать двойные щелчки мышью. При этом в функцию окна посылаются сообщения `WM_LBUTTONDOWNBLCLK` и `WM_RBUTTONDOWNBLCLK`. Если этот стилль не будет задан, функция окна получит только идущие парами сообщения о том, что нажимается и отпускается левая или правая клавиша мыши.
- Сообщение `WM_LBUTTONDOWNBLCLK` помещается в очередь, если пользователь дважды щелкает левой кнопкой мыши, в то время, когда курсор находится в *рабочей* (клиентской) области окна. Если мышь не захвачена окном, сообщение помещается в окно под курсором. В противном случае сообщение помещается в окно, которое захватило мышь.

```
WM_LBUTTONDOWNBLCLK  
WPARAM wParam  
LPARAM lParam;
```

`wParam` указывает, находятся ли в нажатом состоянии различные виртуальные клавиши. `lParam` младшее слово устанавливает x-координату курсора. Старшее слово устанавливает y-координату курсора. Координата - относительно левого верхнего угла рабочей области. Если приложение обрабатывает это сообщение, оно должно вернуть 0.

5. Сообщение `WM_NCHITTEST`, процедура `DefWindowProc`.

- Сообщение `WM_NCHITTEST` отправляется в окно тогда, когда перемещается курсор, или когда кнопка мыши нажимается или отпускается. Если мышь не захвачена окном, сообщение отправляется в окно под курсором. В противном случае сообщение отправляется в окно, которое захватило мышь.

```
WM_NCHITTEST  
WPARAM wParam  
LPARAM lParam;
```

`wParam` этот параметр не используется

lParam младшее слово устанавливает x-координату курсора. Старшее слово устанавливает y-координату курсора. Координата - относительно левого верхнего угла рабочей области. Возвращаемое значение функцией **DefWindowProc** — это одно из значений, которое указывает позицию острия курсора.

- В функции **WndProc** мы обрабатываем сообщения от Windows. При этом, если обработку не совершаем, то вызываем функцию по умолчанию **DefWindowProc**, которая умеет обрабатывать сообщения по умолчанию.

```
LRESULT DefWindowProc
(
    HWND hWnd,          // указатель окна
    UINT Msg,           // идентификатор сообщения
    WPARAM wParam,      // первый параметр сообщения
    LPARAM lParam       // второй параметр сообщения
);
```

Функция возвращает результат обработки сообщения.

6. Функции **SetCapture** и **ReleaseCapture**.

- HWND** **SetCapture**(**HWND** hWnd);
Вызывает посылку всего ввода от курсора в окно hWnd (у мыши появляется новое захватывающее её окно), независимо от положения мыши.
hWnd - идентификатор нового захватывающего окна.
Возвращается предыдущее окно, которое принимало ввод от мыши; **NULL** - если такое окно отсутствует.
- Функция **ReleaseCapture()** освобождает захват окном мыши в текущем потоке и восстанавливает нормальную обработку ввода мыши. Окно, которое захватило мышь, получает весь ввод от мыши независимо от позиции курсора, исключая момент нажатия кнопки над окном другого потока.

Ход работы

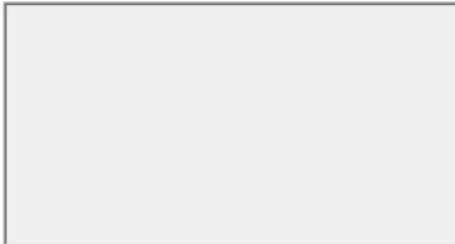
- Создаём приложение Win32 Project (в Visual Studio 2019).
Создание проекта → Мастер классических приложений → Далее → Указываем имя проекта, расположение, имя решения → Создать → Тип приложения: Классическое приложение (.exe), Дополнительные параметры: Пустой проект → ОК. Добавляем .cpp файл.
- В окно приложения добавим две кнопки, поля ввода и вывода. Первая кнопка должна возводить в квадрат введенное целое число, вторая кнопка должна заставить первую кнопку “нажаться” и выполнить код (предусмотреть два возможных варианта воздействия на первую кнопку).

Создаём кнопки и поля:

```
const int UQID_BUTTON1 = 110; HWND butt1;    // кнопка 1
const int UQID_BUTTON2 = 111; HWND butt2;    // кнопка 2
const int UQID_ED1 = 120; HWND ed1;          // поле ввода
const int UQID_ST1 = 125; HWND st1;          // поле вывода 1
const int UQID_ST2 = 130; HWND st2;          // поле вывода 2

butt1 = CreateWindowEx(WS_EX_WINDOWEDGE, _T("BUTTON"), _T("ВОЗВЕСТИ"), WS_VISIBLE | WS_CHILD, 30,
30, 90, 30, hWnd, (HMENU)UQID_BUTTON1, hInstance, NULL);
butt2 = CreateWindowEx(WS_EX_WINDOWEDGE, _T("BUTTON"), _T("КНОПКА 2"), WS_VISIBLE | WS_CHILD, 30,
80, 90, 30, hWnd, (HMENU)UQID_BUTTON2, hInstance, NULL);
ed1 = CreateWindowEx(WS_EX_CLIENTEDGE, _T("EDIT"), _T("1"), WS_VISIBLE | WS_CHILD | ES_LEFT, 150,
30, 100, 30, hWnd, (HMENU)UQID_ED1, hInstance, NULL);
st1 = CreateWindowEx(WS_EX_CLIENTEDGE, _T("STATIC"), _T("1"), WS_VISIBLE | WS_CHILD | ES_LEFT, 150,
80, 100, 30, hWnd, (HMENU)UQID_ST1, hInstance, NULL);
st2 = CreateWindowEx(WS_EX_CLIENTEDGE, _T("STATIC"), _T(""), WS_VISIBLE | WS_CHILD, 30, 150, 280,
150, hWnd, (HMENU)UQID_ST2, hInstance, NULL);
```

ВОЗВЕСТИ	1
КНОПКА 2	1



Кнопки выполняют определённые действия.

Первая кнопка возводит в квадрат введённое в поле EDIT число:

```
TCHAR number[100] = { 0 };
int num = 0;
...
case UQID_BUTT1:
    GetWindowText(ed1, number, 100);
    num = atoi(number); // строку в число
    _itoa_s(num * num, number, 10); // квадрат в строку
    SetWindowText(st1, number);
    break;
```

Результат на картинке:

ВОЗВЕСТИ	13
КНОПКА 2	169

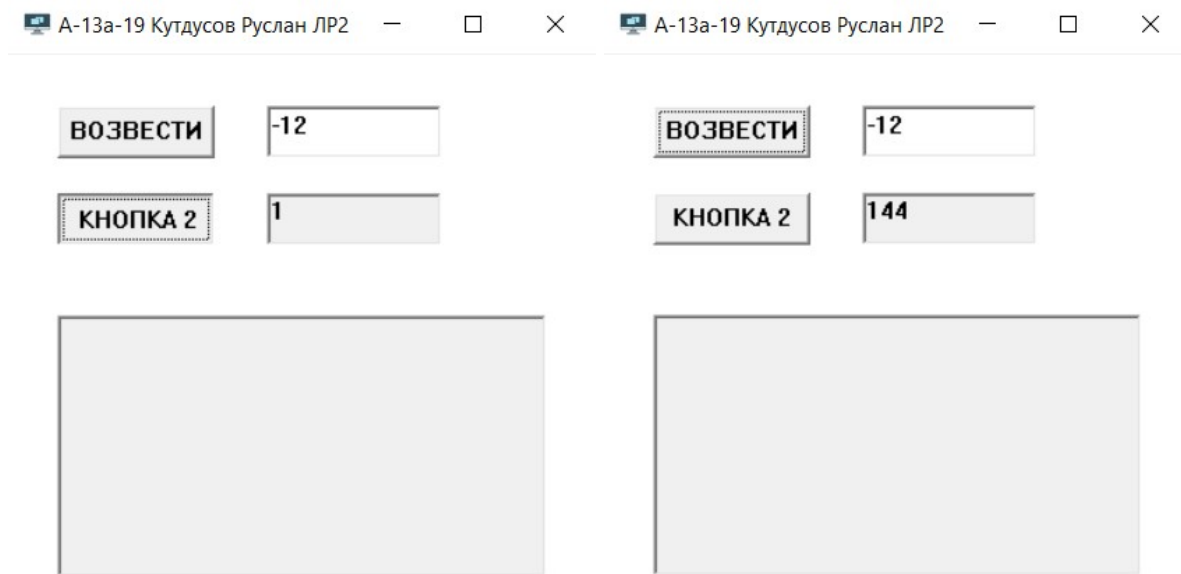


Вторая кнопка заставляет первую кнопку “нажаться” и выполнить код (возможно три возможных варианта воздействия на первую кнопку):

```
case UQID_BUTT2:
    // 1
    /*SendMessage(butt1, WM_LBUTTONDOWN, MK_LBUTTON, 0); // левая кнопка мыши вверх (посылаем
    // сообщение кнопке 1)
    SendMessage(butt1, WM_LBUTTONUP, 0, 0); // левая кнопка мыши вниз
    */
    // 2
    SendMessage(butt1, BM_CLICK, 0, 0);
    // 3
    /*SendMessage(hWnd, WM_COMMAND, UQID_BUTT1, 0);
    SendMessage(butt1, BM_SETSTATE, 1, 0); // кнопка зажимается
    Sleep(100);
```

```
SendMessage(butt1, BM_SETSTATE, 0, 0);*// отжимается
break;
```

Итого у нас получилось:

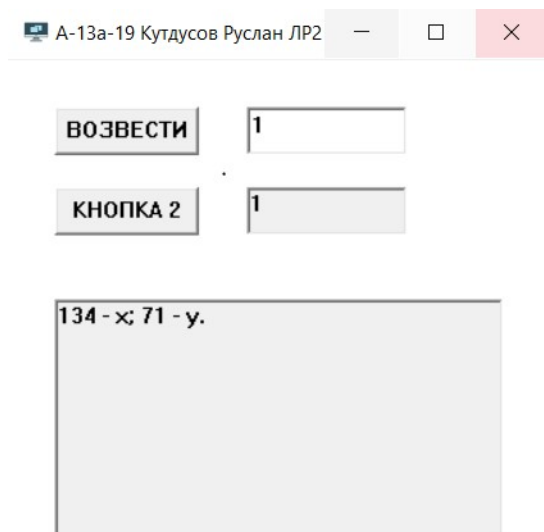


3. По щелчку правой кнопки мыши определим и выведем координаты курсора.

Известно, что у *lParam* сообщения **WM_RBUTTONDOWN** младшее слово устанавливает х-координату курсора, а старшее слово устанавливает у-координату курсора. Воспользуемся этим.

```
POINT p;
TCHAR x[20] = { 0 };
TCHAR y[10] = { 0 };
...
case WM_RBUTTONDOWN:
    p.x = LOWORD(lParam);
    p.y = HIWORD(lParam);
    _itoa_s(p.x, x, 10);
    _itoa_s(p.y, y, 10);
    strcat_s(x, _T(" - x; "));
    strcat_s(y, _T(" - y. "));
    strcat_s(x, y);
    SetWindowText(st2, x);
    break;
```

Результат (точкой отмечено место острия курсора, выведены его координаты):



4. Прделаем программные эксперименты и проследим за работой событий.

- **WM_LBUTTONDOWNBLCLK**: какие события и сколько раз срабатывают при двойном щелчке (**WM_LBUTTONDOWNBLCLK**, **WM_LBUTTONDOWN**, **WM_LBUTTONUP**).

Окно будет игнорировать события WM_LBUTTONDOWNBLCLK, если не изменить его стиль в его классе:

```
WNDCLASSEX wcex;  
wcex.style = CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS;
```

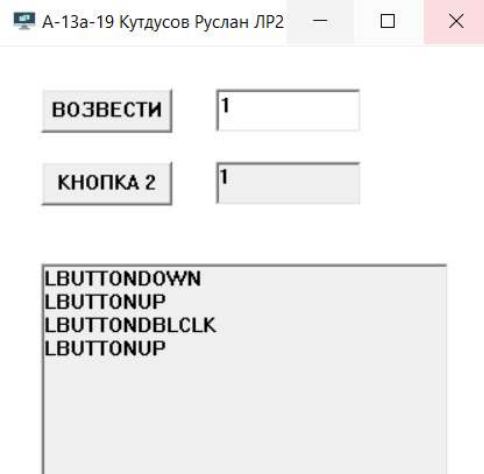
Будем записывать информацию о событиях в строковую переменную, а затем выведем в поле STATIC:

```
static TCHAR msgs[1000] = { 0 }; // сюда запоминаются события левой кнопки мыши
```

Всё готово. А теперь опишем события в переключателе:

```
case WM_LBUTTONDOWN:  
    strcat_s(msgs, _T("LBUTTONDOWN\n"));  
    SetWindowText(st2, msgs);  
    break;  
case WM_LBUTTONUP:  
    strcat_s(msgs, _T("LBUTTONUP\n"));  
    SetWindowText(st2, msgs);  
    break;  
case WM_LBUTTONDOWNBLCLK:  
    strcat_s(msgs, _T("LBUTTONDBLCLK\n"));  
    SetWindowText(st2, msgs);  
    break;
```

Запустим, сделаем двойной щелчок:



Видим, что последовательность сообщений у левой кнопки при двойном щелчке:

WM_LBUTTONDOWN, WM_LBUTTONUP, WM_LBUTTONDOWNBLCLK, WM_LBUTTONUP

- WM_NCHITTEST: узнаем сколько раз возникает данное событие при работе с мышью.
Для удобства создадим ещё одно STATIC поле:

```
const int UQID_ST3 = 135; HWND st3; // поле вывода 3  
st3 = CreateWindowEx(WS_EX_CLIENTEDGE, _T("STATIC"), _T(""), WS_VISIBLE | WS_CHILD, 30, 310, 280,  
    30, hWnd, (HMENU)UQID_ST3, hInstance, NULL);
```

Добавим переменную счётчик:

```
static int wm_nci_n = 0; // счётчик событий WM_NCHITTEST
```

Реализуем, наконец, само событие:

```
TCHAR nci[20] = { 0 };  
...  
case WM_NCHITTEST:  
    wm_nci_n++;  
    _itoa_s(wm_nci_n, nci, 10);  
    strcat_s(nci, _T(" NCHITTEST"));  
    SetWindowText(st3, nci);
```

```
return DefWindowProc(hWnd, message, wParam, lParam); // закомментировав вызов процедуры
// DefWindowProc, мы не сможем
// отслеживать остальные сообщения мыши
```

break;

Запустим программу, сделаем двойной щелчок и немного подвигаем курсор в клиентской области:



Видим, какое большое количество сообщений мыши обработала система.

5. Разработаем программный код, который позволит нам с помощью мыши перемещать окно, но мышь при этом нажимается и перемещается в клиентской области окна.

При нажатии на левую клавишу, и при движении мыши, окно начинает перемещение.

При отжатии левой клавиши – прекращает.

Заведём вспомогательные переменные:

```
static bool window_move = false; // флаг говорит о том, что левая клавиша мыши зажата
static POINT start;              // стартовая точка для определения смещения
...
POINT p;                         // переменная координат курсора
RECT rect;                       // переменная прямоугольника окна
...
case WM_LBUTTONDOWN:
    window_move = true;          // включаем режим перемещения
    GetWindowRect(hWnd, &rect);  // координаты прямоугольника окна
    start.x = LOWORD(lParam) - rect.left; // определяем стартовую позицию
    start.y = HIWORD(lParam) - rect.top;
    ClientToScreen(hWnd, &start); // клиентские координаты стартовой точки в экранные
    strcat_s(msgs, _T("LBUTTONDOWN\n"));
    SetWindowText(st2, msgs);
    break;
case WM_LBUTTONUP:
    window_move = false;         // выключаем режим перемещения
    strcat_s(msgs, _T("LBUTTONUP\n"));
    SetWindowText(st2, msgs);
    break;
case WM_MOUSEMOVE:
    if (window_move) {
        p.x = LOWORD(lParam);    // текущие координаты курсора
        p.y = HIWORD(lParam);
        ClientToScreen(hWnd, &p); // переводим в экранные
        SetWindowPos(hWnd, HWND_TOP, p.x - start.x, p.y - start.y, 0, 0, SWP_NOSIZE |
SWP_SHOWWINDOW);
    }
    break;
```

Полный код программы: <https://pastebin.com/QkYKUwCS>.