Введение в WinAPI. Часть первая. Создание окна.

Венгерская форма записи

Весь код, который мы встретим в WinAPI написан в венгерской форме. Это такое соглашение по написанию кода.

При этом перед именем переменной ставится начальная буква типа. Все слова в именах переменных и функций начинаются с заглавной буквы.

Вот несколько префиксов:

b - переменная типа bool.
l - переменная типа long integer.
w - от word (слово) - 16 бит. Переменная типа unsigned short.
dw - от double word (двойное слово) - 32 бита. Переменная типа unsigned long.
sz - строка заканчивающаяся нулём (string terminated zero). Просто обычная строка, которую мы постоянно использовали.
p или lp - указатель (от pointer). lp (от long pointer) - данные указатели перешли из прошлого. Сейчас lp и p означают одно и то же.
h - описатель (от handle).

Например, указатель будет называться вот так:

void pData;*

Данная форма записи используется Microsoft. Многие критикуют этот способ именования переменных. Но подобные вещи (соглашения о кодировании) в больших компаниях жизненно необходимы.

Напомню, что идентификаторы констант обычно состоят только из заглавных букв: WM_DESTROY. WM_DESTOY - это 2, константа определена через define.

Кроме того, в winAPI используется очень много переопределённых типов. Вот на этой страничке - http://msdn.microsoft.com/en-us/library/aa383751(VS.85).aspx , можете найти описания всех типов Windows (на английском).

И ещё одна вещь, которую мы не разбирали. Указателям часто присваивается значение NULL. Считайте, что это просто 0 и указатели которым присвоено значение NULL (ноль), не указывают ни на какой участок памяти.

Windows API (WinAPI)

Все программы под Windows используют специальный интерфейс программирования WinAPI. Это набор функций и структур на языке C, благодаря которым ваша программа становится совместимой с Windows.

Windows API обладает огромными возможностями по работе с операционной системой. Можно даже сказать - безграничными.

Мы не рассмотрим даже один процент всех возможностей WinAPI. Первоначально я хотел взять больше материала, но это заняло бы слишком много времени, и увязнув в болоте WinAPI, до DirectX'a мы добрались бы через пару лет. Описание WinAPI займёт два урока (включая этот). В них мы рассмотрим только каркас приложения под Windows.

Программа под Windows точно так же как и программа под DOS, имеет главную функцию. Здесь эта функция называется WinMain.

Функция WinMain

Программа под Windows состоит из следующих частей (всё это происходит внутри WinMain):

Создание и регистрация класса окна. Не путайте с классами C++. WinAPI написана на C, здесь нет классов в привычном для нас понимании этого слова.
Создание окна программы.
Основной цикл, в котором обрабатываются сообщения.
Обработка сообщений программы в оконной процедуре. Оконная процедура представляет собой обычную функцию.
Вот эти четыре пункта - основа программы Windows. В течение этого и следующего урока мы разберём всё это подробно. Если вы запутаетесь в описании программы, то вернитесь к этим пунктам.

Теперь разберём всё это подробно:

WinAPI: Структура WNDCLASS

Прежде всего нужно создать и заполнить структурную переменную WNDCLASS, а затем на её основе зарегистрировать оконный класс.

Вот как выглядит эта структура:

код на языке c++
```
typedef struct {
  UINT style;          // стиль окна
  WNDPROC lpfnWndProc; // указатель  на оконную процедуру
  int cbClsExtra;      // дополнительные байты после класса. Всегда ставьте 0
```

```
  int cbWndExtra;     // дополнительные байты после экземпляра окна. Всегда ставьте 0
  HINSTANCE hInstance; // экземпляр приложения. Передаётся в виде параметра в WinMain
  HICON hIcon;          // иконка приложения
  HCURSOR hCursor;      // курсор приложения
  HBRUSH hbrBackground;  // цвет фона
  LPCTSTR lpszMenuName;  // имя меню
  LPCTSTR lpszClassName; // имя класса
} WNDCLASS, *PWNDCLASS;
```

Структура WNDCLASS в составе WinAPI определяет базовые свойства создаваемого окна: иконки, вид курсора мыши, есть ли меню у окна, какому приложению будет принадлежать окно...

После того как вы заполните эту структуру, на её основе можно зарегистрировать оконный класс. Речь идёт не о таких классах как в C++. Скорее можно считать, что оконный класс это такой шаблон, вы его зарегистрировали в системе, и теперь на основе этого шаблона можно создать несколько окон. И все эти окна будут обладать свойствами, которые вы определили в структурной переменной WNDCLASS.

WinAPI: Функция CreateWindow

После регистрации оконного класса, на его основе создаётся главное окно приложения (мы сейчас приступили ко второму пункту). Делается это с помощью функции CreateWindow. Она имеет следующий прототип:

код на языке c++

```
HWND CreateWindow(
  LPCTSTR lpClassName,  // имя класса
  LPCTSTR lpWindowName, // имя окна (отображается в заголовке)
  DWORD dwStyle, // стиль окна
  int x,         // координата по горизонтали от левого края экрана
  int y,         // координата по вертикали от верхнего края экрана
  int nWidth,    // ширина окна
  int nHeight,   // высота окна
  HWND hWndParent, // родительское окно
  HMENU hMenu,     // описатель меню
  HINSTANCE hInstance, // экземпляр приложения
  LPVOID lpParam       // параметр; всегда ставьте NULL
);
```

Если в оконном классе (структуре WNDCLASS) задаются базовые свойства окна, то здесь - более специфические для каждого окна: размер окна, координаты...

Данная функция возвращает описатель окна. С помощью описателя можно обращаться к окну, это примерно как идентификатор.

Обратите внимание, что здесь очень много новых типов. На самом деле они все старые, просто переопределены. Например: HWND - это переопределение типа HANDLE, который в свою очередь является переопределением PVOID, который в свою очередь является переопределением void*. Как глубоко закопана правда! Но всё же тип HWND - это указатель на void.

Окно состоит из нескольких частей. Практически в каждой программе вы увидите: заголовок окна, системное меню (если нажать на иконку приложения в левой верхней части окна), три системные кнопки для работы с окном: свернуть, развернуть на весь экран и закрыть. Также, практически всегда в приложении присутствует меню. Вот как раз последнего у нас точно не будет. И, конечно же, большую часть окна занимает т.н. клиентская область, в которой обычно и работает пользователь.

Это что касается оконного режима. Довольно долго мы будем практиковаться с DiectX именно в окне - не будем пользоваться полноэкранным режимом.

Обработка сообщений (Message handling)

Основным отличием всех наших предыдущих программ от программ под Windows является обработка сообщений.

Например, когда пользователь нажимает какую-нибудь клавишу на клавиатуре, генерируется сообщение, что была нажата клавиша. Затем это сообщение поступает в приложение, которое было активным, когда пользователь нажал клавишу.

Здесь у нас произошло событие (event) - была нажата клавиша.

Событием может быть: перемещение курсора мыши, смена фокуса приложения, нажатие клавиши клавиатуры, закрытие окна. Событий очень много. Очень! За секунду в операционной системе может происходить десятки событий.

Так вот, когда происходит какое-либо событие, операционная система создаёт сообщение: была нажата такая-то клавиша, координаты курсора мыши изменились, открылось новое окно.

Сообщения может создавать как операционная система, так и различные приложения.

Сообщение представляет собой структуру, и выглядят следующим образом:

код на языке c++
```
typedef struct tagMSG {
HWND hwnd;      // окно, которое получит это сообщение
UINT message;   // код сообщения
WPARAM wParam;  // параметр
LPARAM lParam;  // параметр
DWORD time;     // время, когда произошло сообщение
POINT pt;       // координаты курсора мыши
} MSG;
```

Обратите внимание, как с помощью typedef переопределяются структуры.

Чтобы создать данную структуру можно воспользоваться следующим кодом:

```
MSG msg;
```

Всё. Дальше можете использовать эту структуру как обычно.

код на языке c++
```
msg.messgae == 2;          // эти две строки эквивалентны так как
msg.message == WM_DESTROY; // константа WM_DESTROY равна двум
```

Здесь, поле, в котором содержится код сообщения (имя сообщения, сравнивается с константой WM_DESTROY. WM - от Windows Message (сообщение Windows). WM_DESTROY - это сообщение, которое генерируется при закрытии окна (destroy - уничтожить).

Коды сообщений определены с помощью констант и имеют префикс WM_: WM_CLOSE, WM_CREATE и др.

В структуре MSG встречается тип HWND - от Window Handle (дескриптор окна или описатель окна). Это такая штука, которая "описывает" окно. Это что-то вроде идентификатора (имени окна).

Запомните это слово - handle (описатель, дескриптор). В Windows это понятие используется очень часто. Почти все типы Windows, которые начинаются с H - описатели: описатель иконки, описатель шрифта, описатель экземпляра приложения. Их штук тридцать насколько я помню.

Все взаимодействия между приложениями в Windows осуществляются с помощью этих самых описателей окон (HWND).

Существует ещё один важный описатель - описатель приложения (HINSTANCE - первый параметр WinMain) - это уникальный идентификатор приложения, благодаря которому операционная система не сможет перепутать две разных программы. Это примерно как штрих-код. Мы рассмотрим его позже.

Каждый раз, когда пользователь совершает какое-то действие, создаётся и заполняется сообщение: задаётся описатель окна, которое должно получить данное сообщение, задаётся идентификатор сообщения, заполняются параметры, заполняется время (текущее) и указываются координаты курсора мыши (смотрите структуру).

После этого данное сообщение помещается в очередь сообщений операционной системы. Когда доходит очередь до нашего сообщения, оно отправляется нужному окну (windows знает какому окну отправлять каждое сообщение благодаря описателям). Когда сообщение приходит в приложение, оно помещается в очередь сообщений приложения. Как только до него доходит очередь, оно обрабатывается.

Смотрите, между тем моментом, когда пользователь совершил какое-либо действие (произошло событие и сгенерировалось сообщение) и тем моментом, когда программа среагировала на это действие (сообщение было обработано программой) происходит много событий. Ведь как в очереди сообщений Windows так и в очереди сообщений приложения может быть много сообщений. В первом случае речь может идти о сотнях, во втором случае как минимум о нескольких.

Оконная процедура (Window procedure - WndProc)

Продолжаем с того момента, как сообщение попало в очередь сообщений приложения. Как только до него дошла очередь, оно обрабатывается.
Для обработки сообщений в каждой программе должна существовать специальная функция - оконная процедура. Обычно она называется WndProc (от Window Procedure). Вызов оконной процедуры расположен в основном цикле программы и выполняется при каждой итерации цикла.

Сообщения (в виде структурных переменных MSG) попадают в данную функцию в виде параметров: описатель окна, идентификатор сообщения

и два параметра. Обратите внимание, что в оконную процедуру не передаются поля time и pt. То есть сообщение уже "разобрано".

Внутри оконной процедуры расположено ветвление switch, в котором идёт проверка идентификатора сообщения. Вот пример простой оконной процедуры (она полностью рабочая):

код на языке c++

```
//  не обращайте пока внимания  на HRESULT  и __stdcall. Мы рассмотрим их позже
HRESULT __stdcall WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  switch (message)
  {
  case WM_PAINT:
    // код обработки сообщения WM_PAINT
    return 0;
  case WM_DESTROY:
    // код обработки сообщения WM_DESTROY
    return 0;
  }
  // обработчик всех остальных сообщений
}
```

И последнее - основной цикл. Он очень прост. Каждую итерацию цикла проверяется очередь сообщений приложения. Если в очереди сообщений есть сообщение, оно вытаскивается из очереди. Затем в теле цикла происходит вызов оконной процедуры, чтобы обработать взятое из очереди сообщение.

Вот, в общем-то, и всё на сегодня. Уже видно, что программа под WinAPI намного сложнее программы под DOS. Как я уже писал выше, в следующем уроке мы разберём код работающей программы.

В качестве упражнения создайте новый проект. В окне New Project (Новый проект) выберите шаблон (template) - Win32Project (до сих пор мы выбирали Win32 Console Application). В одном из следующих окон не ставьте флажок Empty Project (пустой проект) и IDE сгенерирует заготовку программы.

Если вы внимательно посмотрите на код файла имя_проекта.cpp, то вы обнаружите все вещи, которые мы обсуждали: структурную переменную MSG, заполнение структуры WNDCLASS, создание окна функцией CreateWindow, основной цикл программы. Кроме того, в файле определена функция WndProc. В ней происходит обработка нескольких сообщений в ветвях switch: WM_COMMAND, WM_PAINT, WM_DESTROY. Найдите всё это в файле.

Кроме того что мы рассмотрели, в программе содержится много дополнительного кода. В следующем выпуске мы рассмотрим код программы, в котором будет вырезано всё лишнее. Он будет намного проще и понятнее того, что генерирует IDE.

Введение в WinAPI. Часть вторая.

Сегодня мы разберём работающую под Windows программу, написанную с использованием WinAPI. В данном варианте программы я вырезал всё ненужное, оставив самый минимум без которого программа не запустится (ну, вообще-то ещё можно кое-что порезать, но я сдержался).

В этот раз я не дам полный листинг программы. Чтобы запустить её, вам понадобится внимательно проследить за описанием, и самостоятельно собрать весь код. Оставляю вам это удовольствие.

Скорее всего вам понадобится и предыдущий выпуск - чтобы увидеть общую картину, так сказать.

В окне New Project (Новый проект) выберите шаблон (template) - Win32Project (до сих пор мы выбирали Win32 Console Application). В одном из следующих окон поставьте флажок Empty Project (пустой проект). Затем добавьте пустой файл к проекту.
Начинаем как обычно - с глобальной области видимости. Включаем заголовочный файл windows.h. Дальше идёт прототип функции:

код на языке c++
```
LRESULT __stdcall WndProc(HWND hWnd, UINT message,
                  WPARAM wParam, LPARAM lParam);
```

Далее идёт определение функции WinMain. Данная функция - аналог консольной main, у неё те же задачи. Заголовок функции выглядит так:

код на языке c++
```
int __stdcall WinMain  (HINSTANCE hInstance,
                HINSTANCE hPrevInstance,
                LPSTR lpCmdLine, int nCmdShow)
```

Как мы видим, функция возвращает целое число. Не обращайте пока внимания на __stdcall.

В функции четыре параметра. Эти параметры передаются операционной системой. В первый пораметр типа HINSTANCE (instance - экземпляр) передаётся экземпляр приложения. Экземпляр приложения - это уникальный идентификатор, благодаря которому операционная система различает

приложения. Т.е. для операционной системы hInstance - это имя вашего приложения. hInstance - целое беззнаковое число unsigned int (UINT).

Второй параметр также обладает типом HINSTANCE. В данный момент он не используется. Раньше (когда у компьютеров было недостаточно памяти) через этот параметр передавался экземпляр родительского приложения.

Третий параметр lpCmdLine типа LPSTR (long pointer to string - указатель на строку) хранит аргументы командной строки. Мы им пользоваться не будем.

Четвёртый параметр nCmdShow - набор флагов. Через этот параметр передаётся начальное состояние окна приложения. Например: будет ли окно при запуске приложения свёрнуто.

Функция WinMain

В теле функции WinMain прежде всего нужно создать и заполнить оконный класс.

WNDCLASS wc; // WNDCLASS - структура!
Данная структура обладает десятью полями (подробности в прошлом уроке).

1. Поле style хранит стиль окна. Например CS_HREDRAW позволяет перерисовывать окно, если была изменена его ширина. Задайте это поле как CS_OWNDC. CS - от Class Style (стиль окна). В данном случае окну присваивается свой собственнй контекст устройства (необходимо для рисования в окне).

2. Следующее поле структуры lpfnWndProc - здесь нужно указать имя оконной процедуры. Присвойте WndProc.

3,4. cbClsExtra и cbWndExtra присвойте ноль.

5. hInstance. Присвойте данному полю экземпляр приложения (первый параметр функции WinMain).

6. hIcon - это иконка, которая будет отображаться на панели задач. Мы воспользуемся стандартной:

wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
7. Поле hCursor - курсор мыши, который будет отображаться, когда курсор находится в клиентской области окна. Будем использовать стандартный:

wc.hCursor = LoadCursor(NULL, IDC_ARROW);
8. Следующее поле hbrBackground. Данное поле определяет цвет фона

клиентской области. У нас фон будет белым (можете подобрать другие цвета, изменив значение в скобках):

wc.hbrBackground = (HBRUSH)(6);
9. lpszMenuName - меню. У нас меню не будет, присвойте этому полю NULL.

10. lpszClassName - имя класса окна. Присваиваем текстовую строку. Перед кавычками необходимо поставить символ L для преобразования строки типа wchar_t. Без преобразования, программа не скомпилируется:

wc.lpszClassName = L"class";
Теперь, когда у нас есть заполненная структура WNDCLASS, можно зарегистрировать оконный класс. Класс регистрируется с помощью функции RegisterClass. Аргумент - адрес структуры.

RegisterClass(&wc);
Теперь в операционной системе зарегистрирован новый класс окна и можно создать окно этого класса.

код на языке c++
```
HWND hWnd = CreateWindow(L"class",L"заготовка программы",
                    WS_OVERLAPPEDWINDOW,
                    150,100,500,400,
                    NULL,NULL,hInstance,NULL);
```

Функция возвращает описатель окна. Описатели окон используются чтобы отличать окна друг от друга. Нам нужна переменная, которая будет хранить описатель созданного окна, для обработки сообщений.

Первый параметр - имя класса. Имя должно совпадать с полем оконного класса lpszClassName.

Затем идёт заголовок окна. Обе константные строки (имя класса и заголовок окна) нужно преобразовывать к типу wchar_t. То есть перед кавычками нужно поставить букву L.

Следующий параметр определяет стиль окна. Здесь мы задали WS_OVERLAPPEDWINDOW (WS - от Window Style (стиль окна)). Данная константа задаёт стандартный стиль окна.

Следующие четыре параметра: x-координата, y-координата, ширина, высота.

Затем идут: родительское окно и меню. Для обоих задаём NULL.

Предпоследний параметр - описатель приложения, которому принадлежит данное окно (первый параметр WinMain).

И последний - дополнительные параметры. Использовать не будем, поэтому присваиваем NULL.

Всё, с этого момента в вашей программе существует окно. Следующий шаг - показать его.

ShowWindow(hWnd, nCmdShow);
Первый аргумент - описатель созданного нами окна, второй - четвёртый аргумент WinMain. Т.е. в данной функции мы говорим созданному окну (hWnd) как оно должно отображаться (через nCmdShow).

Ну и собственно рисуем окно:

UpdateWindow(hWnd);

В данной функции окну посылается сообщение WM_PAINT и оно может нарисовать себя.

В функции WinMain осталось рассмотреть только основной цикл.

Перед циклом определите переменную msg типа MSG.

Основной цикл

Полностью, тело основного цикла выглядит следующим образом:

код на языке c++
```
while (true)
{
 if (PeekMessage(&msg,hWnd,0,0,PM_REMOVE))
 {
   if (msg.message == WM_QUIT)
     break;

   TranslateMessage(&msg);
   DispatchMessage(&msg);
 }
}
```

Условие выхода из цикла - простое:

while (true)

Можно поменять его на:

while (1)

Это, как вы конечно знаете, одно и то же.

Тело цикла тоже довольно простое. Состоит оно из одного блока if. Условие в блоке следующее:

PeekMessage(&msg,hWnd,0,0,PM_REMOVE)

То есть - простой вызов функции PeekMessage. Эта функция проверяет очередь сообщений приложения. Если там есть сообщение, то функция возвращает единицу, и выполняется код в блоке if. Кроме того, если в очереди есть сообщение, то оно копируется в нашу структуру msg, адрес которой мы передали.

Вторым аргументом является описатель окна. Этот аргумент говорит функции PeekMessage, сообщения какого окна нужно проверять. Если в этот аргумент передать NULL, то функция будет проверять сообщения всех окон данного приложения. Мы же передаём значение hWnd - описатель созданного окна. Поэтому данная функция будет искать в очереди сообщений только те, которые предназначены окну hWnd.

Третий и четвёртый аргументы - фильтры. Пока что передавайте 0.

И последний аргумент - константа PM_REMOVE. Этот флаг говорит, что если в очереди есть сообщение, то оно удаляется из очереди. Мы могли бы использовать флаг PM_NOREMOVE. Тогда бы функция PeekMessage не удаляла бы из очереди сообщение, а просто проверяла очередь на наличие сообщений.

Теперь, если функция PeekMessage вернула единицу (в очереди есть хотя бы одно сообщение), выполняется тело ветвления if, а наша структурная переменна msg заполнена и в ней хранится сообщение.

Далее мы проверяем поле message и если в нём хранится WM_QUIT, то выходим из цикла (завершение программы). Если же в message другое сообщение, то выполняются две функции TranslateMessage и DispatchMessage, в каждую из которых мы передаём адрес структурной переменной msg.

Функция TranslateMessage нужна для обработки нажатий клавиш клавиатуры, что конретно делает эта функция мы ещё рассмотрим.

Функция DispatchMessage обрабатывает сообщение, вытаскивает из него всю информацию и передаёт её в виде аргументов в оконную процедуру WndProc. Т.е. WndProc вызывается внутри DispatchMessage.

В конце функции WinMain, после окончания цикла while не забудьте добавить оператор:

return 0;

Параметры MSG

Небольшое лирическое отступление про параметры сообщений.

В структуре MSG используются два параметра: wParam и lParam. Сейчас они одного размера - 32 бита. Раньше wParam был равен 16-и битам. w - от word (слово, 16 бит), l - от long (32 бита). В данных параметрах хранится дополнительная информация сообщения.

Параметры могут быть закодированы в словах. Оба параметра равны 32-ум битам. Т.е. в каждом по два слова (слово равно 16 битам). Чтобы получить доступ к нижнему слову (биты от 0 до 15) можно воспользоваться макросом LOWORD. Пока считайте, что макрос - это функция:

LOWORD(msg.lParam); // доступ к нижнему слову поля wParam

Кроме того, можно получить доступ к верхнему слову (биты от 16 до 32) с помощью макроса HIWORD:

HIWORD(msg.wParam); // доступ к верхнему слову поля lParam.

В параметрах передаётся дополнительная информация сообщения.
В некоторых сообщениях параметры не используются. Например, в сообщении WM_DESTROY

Оконная процедура WndProc

Пользователь не вызывает эту функцию явно, она вызывается в функции DispatchMessage. DispatchMessage знает имя WndProc, так как мы его записали в поле оконного класса WNDCLASS.

WndProc принимает четыре аргумента. Все они - поля структурной переменной msg, в которую мы сохранили сообщение с помощью фукнции PeekMessage. Адрес этой структурной переменной мы передали в DispatchMessage.

Аргументы WndProc:
hWnd - описатель окна, которому было отправлено сообщение.
message - идентификатор сообщения. Например: WM_PAINT (перерисовать окно).
wParam - параметр.
lParam - параметр.

Внутри функции находится ветвление switch, которое проверяет message. Обычно создание программы под Windows состоит в заполнении разных блоков этого ветвления.

У нас пока будет только одна ветвь:

case WM_DESTROY:
PostQuitMessage(0);
return 0;
Если отправленное сообщение является WM_DESTROY, мы посылаем сообщение WM_QUIT (которое проверяется в основном цикле). Сообщение WM_QUIT мы можем отправить с помощью функции PostQuitMessage. Мы можем послать аргумент - в нашем случае 0. Когда мы получим это сообщение из очереди, этот ноль будет храниться в поле wParam.

Кратко о закрытии приложений. Когда мы нажимаем на крестик в правом верхнем углу окна, этому окну посылается сообщение WM_CLOSE. Мы можем обработать данное сообщение в WndProc, добавив ветвь:

код на языке c++
```
case WM_CLOSE:
  // код обработки сообщения
  return 0;
```

Вы наверное не раз сталкивались с ситуацией, когда при закрытии документов предлагается сохранить изменения. Вот, эти действия описаны в обработчике WM_CLOSE.

Если вы пишите обработчик WM_CLOSE, то в этом блоке вы обязательно должны отправить сообщение WM_DESTROY. Оно служит сигналом уничтожения окна. Так как в нашей программе не обрабатывается WM_CLOSE, WM_DESTROY посылается автоматически.

В нашей программе есть обработка сообщения WM_DESTROY - мы просто посылаем сообщение WM_QUIT (с помощью функции PostQuitMessage).

Обратите внимание, что в нашем случае WM_QUIT не обрабатывается в WndProc. Мы проверяем сообщения (поле message структурной переменной msg) в основном цикле, и если текущим сообщением является WM_QUIT, мы выходим из цикла. Соответственно и из программы.

Все сообщения, которые мы не обрабатываем самостоятельно, должны быть направлены в обработчик сообщений по умолчанию. Делается это так (добавьте эту строку после switch или в блок default):

return DefWindowProc(hWnd,message,wParam,lParam);
В нашем примере, сюда отправляется WM_CLOSE, чтобы было отправлено сообщение WM_DESTROY.

"Как же всё сложно!" - воскликнет внимательный читатель - "Чтобы только закрыть программу, отправляется три сообщения: WM_CLOSE, WM_DESTROY, WM_QUIT". Могу только добавить, что выполнение других действий не проще.

Посмотрите сколько пришлось написать кода, чтобы всего-лишь создать небольшое окно с белым фоном. К счастью для нас, мы почти не будем использовать дополнительные возможности WinAPI. Но к несчастью для нас, мы будем пользоваться библиотекой DirectX. А там всё сложнее в разы!!! MWHAAAAA-HA-HA-HA-HA!!!

На этом почти всё.

Соглашение о вызове функций stdcall и cdecl

Мы не рассмотрели ещё одну особенность функций WndProc и WinMain. В заголовках этих функций стоит ключевое слово __stdcall. __stdcall это такое соглашение по вызову функций. По умолчанию используется __cdecl. То есть во всех наших функциях использовалось именно __cdecl, просто его не нужно писать явно (хотя и можно). Функции Windows требуют использования __stdcall.

Разницу между двумя способами вызовов функций: __cdecl и __stdcall мы рассматривать не будем, но вообще, данные ключевые слова влияют на то, как функция будет взаимодействовать со стеком. Кстати, если вы не знали, в любой программе обязательно используется стек.

В следующем уроке мы начнём рассматривать программу использующую DirectX. К этому времени у вас уже должен быть установлен DirectX SDK. Как его устанавливать мы уже обсуждали в одном из прошлых уроков.

Упражнения

Собрать программу из выпуска.
Поменяйте стиль окна на WS_POPUP.
В программе есть ошибка: когда вы закрываете окно (нажимаете на крестик), то окно закрывается, а программа не завершается. Это можно понять, так как отладчик ещё работает, ну или можете посмотреть в Диспетчере задач Windows. Так вот, где ошибка в программе? В уроке, кстати, содержится ответ (это если внимательно читать).
Неплохо было бы, если бы вы могли написать эту программу, хотя бы минут за пятнадцать, и используя только предыдущий выпуск.