**Методические указания к лабораторной работе №1**

**Каркас Windows-приложения**

В отличие от программы, выполняемой в операционной системе MS-DOS, даже для создания простейшего приложения под Windows придется проделать намного больше работы. Чтобы иметь возможность работать с оконным интерфейсом, заготовка или каркас Windows-приложения должна выполнить некоторые стандартные действия:

1. Определить *класс окна*.
2. Зарегистрировать окно.
3. Создать окно данного класса.
4. Отобразить окно.
5. Запустить цикл обработки сообщений.

Рассмотрим, как можно "вручную" создать минимальное Win32-приложение. Загрузив Visual Studio 2010, выполним команду **File | New | Project**, и выберем тип проекта — **Win32 Project**. В раскрывающемся списке **Location** вы берем путь к рабочей папке, а в поле **Name** имя проекта. В следующем диалоговом окне, нажимаем кнопку **Next**, а в окне опции проекта выберем флажок **Empty project** (Пустой проект) и нажмем кнопку **Finish** — по­лучим пустой проект, в котором нет ни одного файла.

Добавление к проекту нового объекта с помощью меню **Project | Add new Item** и выберем тип файла C++ File (.cpp), а в поле **Name** имя файла.

С помощью листинга 1.1 рассмотрим "скелет" Windows-приложения.

**Листинг 1.1. Минимальный код каркаса Windows-приложения**

#include <Windows.h>

#include <tchar.h>

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

TCHAR WinName[] = \_T("MainFrame");

int WINAPI \_tWinMain(HINSTANCE This, // Дескриптор текущего приложения

HINSTANCE Prev, // В современных системах всегда 0

LPTSTR cmd, // Командная строка

int mode) // Режим отображения окна

{

HWND hWnd; // Дескриптор главного окна программы

MSG msg; // Структура для хранения сообщения

WNDCLASS wc; // Класс окна

// Определение класса окна

wc.hInstance = This;

wc.lpszClassName = WinName; // Имя класса окна

wc.lpfnWndProc = WndProc; // Функция окна

wc.style = CS\_HREDRAW | CS\_VREDRAW; // Стиль окна

wc.hIcon = LoadIcon(NULL, IDI\_APPLICATION); // Стандартная иконка

wc.hCursor = LoadCursor(NULL, IDC\_ARROW); // Стандартный курсор

wc.lpszMenuName = NULL; // Нет меню

wc.cbClsExtra = 0; // Нет дополнительных данных класса

wc.cbWndExtra = 0; // Нет дополнительных данных окна

wc.hbrBackground = (HBRUSH) (COLOR\_WINDOW+1); // Заполнение окна белым цветом

// Регистрация класса окна

if (!RegisterClass(&wc)) return 0;

// Создание окна

hWnd = CreateWindow(WinName, // Имя класса окна

\_T("Каркас Windows-приложения"), // Заголовок окна

WS\_OVERLAPPEDWINDOW, // Стиль окна

CW\_USEDEFAULT, // x

CW\_USEDEFAULT, // y Размеры окна

CW\_USEDEFAULT, // width

CW\_USEDEFAULT, // Height

HWND\_DESKTOP, // Дескриптор родительского окна

NULL, // Нет меню

This, // Дескриптор приложения

NULL); // Дополнительной информации нет

ShowWindow(hWnd, mode); // Показать окно

// Цикл обработки сообщений

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

{

TranslateMessage(&msg); // Функция трансляции кодов нажатой клавиши

DispatchMessage(&msg); // Посылает сообщение функции WndProc()

}

return 0;

}

// Оконная функция вызывается операционной системой

// и получает сообщения из очереди для данного приложения

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

{

switch(message) // Обработчик сообщений

{

case WM\_DESTROY :

PostQuitMessage(0);

break; // Завершение программы

default : // Обработка сообщения по умолчанию

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

}

return 0;

}

Программа не делает ничего полезного, поэтому, запустив ее на выполнение кнопкой ► (**Start Debugging**), мы получим пустое окно, имею­щее заголовок и набор стандартных кнопок.

**Исследование каркаса Windows-приложения**

Давайте подробно рассмотрим текст нашей программы. Первая строка содержит файл включений, который обязательно присутствует во всех Windows-программах:

#include <Windows.h>

Если в ранних версиях Visual Studio этот файл содержал основные определения, то сейчас он служит для вызова других файлов включений, основные из которых windef.h, winbase.h, wingdi.h, winuser.h; а также несколько дополнитель­ных файлов, в которых помещены определения API-функций, констант и макросов.

Дополнительно подключим:

#include <tchar.h>

В этом файле содержатся определения некоторых полезных макросов, например ма­крос \_T() служит для создания строки Unicode на этапе компиляции и определен примерно так:

#define \_Т(х) \_\_Т(х)

#ifdef \_UNICODE

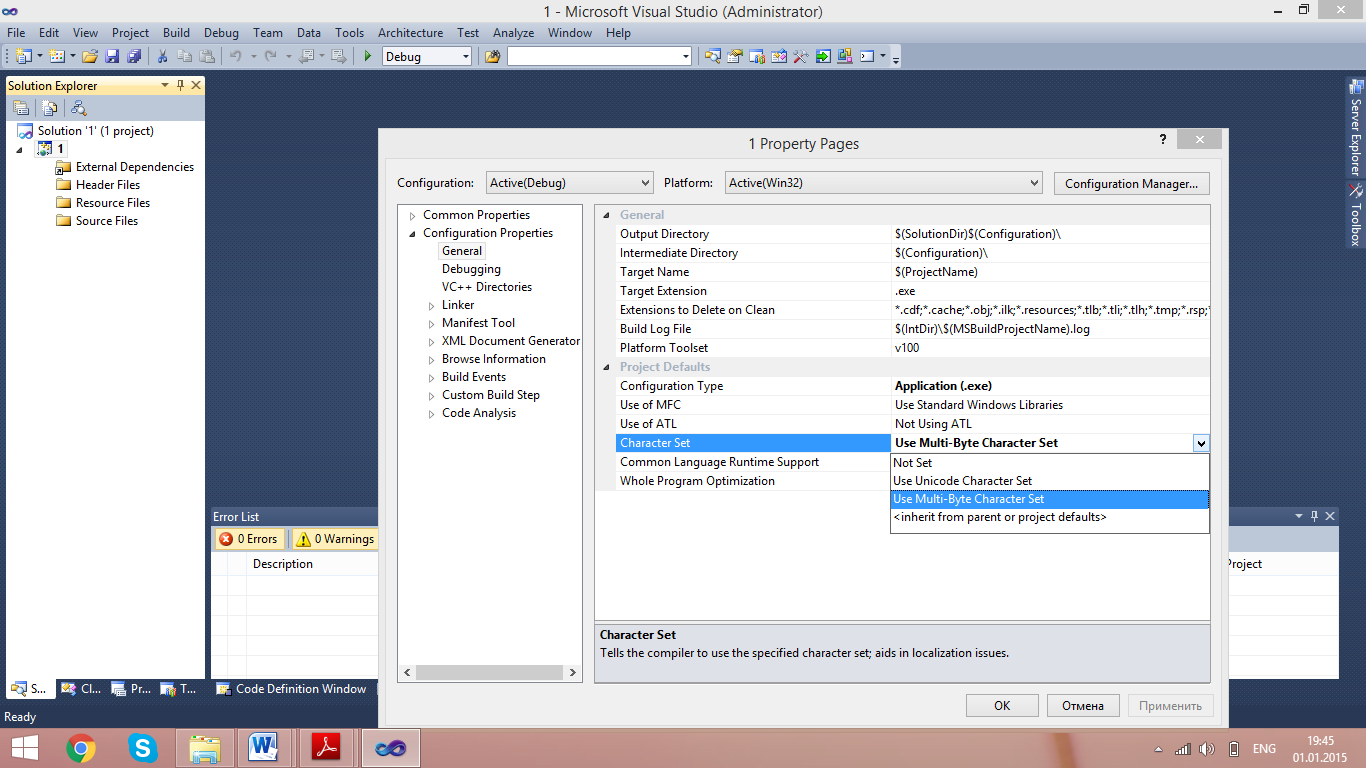
#define \_\_T(x) L ## x

#else

#define \_\_T(x) x

#endif

Макрос преобразуется в оператор "L", который является инструкцией компилятора для образования строки Unicode, если определена константа \_UNICODE; и в "пустой оператор", если константа не определена. Константа \_UNICODE устанавливается в зависимости от установок свойства проекта **Character Set**. Диалоговое окно свойств **Property Pages** доступно сейчас на подложке **Property Manager** пане­ли управления **Solution Explorer**.



**Рис. 1.1.** Страница общих свойств проекта

Таким образом, этот макрос позволяет компилировать проект как в кодировке Unicode, так и в Windows-кодировке. Mногие определения Windows описаны подобным образом.

Далее следует прототип оконной функции:

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

Оконная функция также является функцией обратного вызова, что связано с некоторыми особенностями организации вызовов операционной системы. Эта функция регистрируется в системе, а ее вызов осуществляет операционная система, когда требуется обработать сообщение. Тип возвращаемого значения функции LRESULT эквивалентен long для **Win32**-npoeкта.

На глобальном уровне описывается имя класса окна приложения в виде текстовой строки:

TCHAR WinName[] = \_T("MainFrame");

Тип TCHAR также преобразуется в wchar\_t, если определена константа \_UNICODE, и в char, если константа не определена.

***Примечание***

Тип wchar\_t эквивалентен типу short и служит для хранения строк в кодировке Unicode, где для одного символа выделяется 16 бит.

Имя класса окна используется операционной системой для его идентификации. Имя может быть произвольным, в частности содержать кириллический текст.

Рассмотрим заголовок головной функции:

int WINAPI \_tWinMain(HINSTANCE This, // Дескриптор текущего приложения

HINSTANCE Prev, // В современных системах всегда 0

LPTSTR cmd, // Командная строка

int mode) // Режим отображения окна

Для Windows-приложений с Unicode она носит имя wWinMain(), а в 8-битной кодировке — WinMain(), выбор варианта определяется префиксом \_t, что также является стандартным приемом в библиотеке API-функций. Функция имеет четыре параметра, устанавливаемых при загрузке приложения:

This — дескриптор, присваиваемый операционной системой при загрузке приложения;

Prev — параметр предназначен для хранения дескриптора предыдущего экземпляра приложения, уже загруженного системой. Сейчас он потерял свою актуальность и сохранен лишь для совместимости со старыми приложениями (начиная с Windows 95, параметр устанавливается в нулевое значение);

cmd — указатель командной строки, но без имени запускаемой программы. Тип LPTSTR эквивалентен TCHAR\*;

mode — режим отображения окна.

***Примечание***

Windows-тип данных — *дескриптор* (описатель), используется для описания объектов операционной системы. Дескриптор напоминает ин­декс хеш-таблицы и позволяет отслеживать состояние объекта в памяти при его переме­щении по инициативе операционной системы. Предусмотрено много типов дескрипторов HINSTANCE, HWND и др., но все они являются 32-разрядными целыми числами.

Внутри головной функции описаны три переменные:

hwnd — предназначена для хранения дескриптора главного окна программы;

msg — это структура, в которой хранится информация о сообщении, передаваемом операционной системой окну приложения:

struct MSG

{

HWND hWnd; // Дескриптор окна

UINT message; // Номер сообщения

WPARAM wParam; // 32-разрядные целые содержат

LPARAM lParam; // дополнительные параметры сообщения

DWORD time; // Время посылки сообщения в миллисекундах

POINT pt; // Координаты курсора (х,у)

};

struct POINT

{

LONG х,у;

};

***Примечание***

Тип WPARAM — "короткий параметр" был предназначен для передачи 16-разрядного зна­чения в 16-разрядной операционной системе, в Win32 это такое же 32-разрядное значе­ние, что и LPARAM.

* 1. wc — структура, содержащая информацию по настройке окна. Требуется заполнить следующие поля:

wc.hInstance = This; // Дескриптор текущего приложения

wc.lpszClassName = WinName; // Имя класса окна

wc.lpfnWndProc = WndProc; // Имя оконной функции для обработки сообщений

wc.style = CS\_HREDRAW | CS\_VREDRAW; // Такой стиль определяет автоматическую

// перерисовку окна при изменении его ширины или

// высоты.

CS\_BYTEALIGNCLIENT — внутренняя область окна (client area) должна быть выравнена по границе байта видеопамяти. Иногда используется для ускорения процесса вывода изображенияж;

CS\_BYTEALIGNWINDOW — все окно (не только внутренняя область окна) должно быть выравнено по границе байта видеопамяти;

CS\_CLASSDC — необходимо создать единый контекст отображения, который будет использоваться всеми окнами, создаваемыми на базе данного класса;

CS\_DBLCLKS — функция окна будет получать сообщения при двойном щелчке клавишей мыши (double click);

CS\_GLOBALCLASS — данный класс является глобальным и доступным другим приложениям. Другие приложения могут создавать окна на базе этого класса;

CS\_HREDRAW — внутренняя область окна должна быть перерисована при изменении ширины окна;

CS\_NOCLOSE — в системном меню окна необходимо запретить выбор функции закрытия окна (строка Close будет отображаться серым цветом, и ее нельзя выбрать);

CS\_OWNDC — для каждого окна, определяемого на базе данного класса, будет создаваться отдельный контекст отображения;

CS\_PARENTDC — окно будет пользоваться родительским контекстом отображения, а не своим собственным. Родительский контекст - это контекст окна, создавшего другое окно;

CS\_SAVEBITS — для данного окна Windows должна сохранять изображение в виде битового образа (bitmap). Если такое окно будет перекрыто другим окном, то после уничтожения перекрывшего окна изображение первого окна будет восстановлено Windows на основании сохраненного ранее образа;

CS\_VREDRAW — внутренняя область окна должна быть перерисована при изменении высоты окна;

wc.hIcon = LoadIcon(NULL, IDI\_APPLICATION); // Дескриптор пиктограммы (иконки)

// приложения

Функция LoadIcon() обеспечивает ее загрузку. Если первый параметр NULL, используется системная пиктограмма, которая выбирается по второму параметру из следующего набора:

1. IDI\_APPLICATION — стандартная иконка;
2. IDI\_ASTERISK — звездочка;

IDI\_EXCLAMATION — восклицательный знак;

IDI\_HAND — ладонь;

IDI\_QUESTION — вопросительный знак;

IDI\_WINLOGO — логотип Windows;

wc.hCursor = LoadCursor(NULL, IDC\_ARROW);

1. Аналогичная функция LoadCursor() обеспечивает загрузку графи­ческого образа курсора, где нулевой первый параметр также означа­ет использование системного курсора, вид которого можно выбрать из списка:
2. IDC\_ARROW — стандартный курсор;
3. IDC\_APPSTARTING — стандартный курсор и маленькие песочные часы;
4. IDC\_CROSS — перекрестие;
   1. IDC\_IBEAM — текстовый курсор;
   2. IDC\_NO — перечеркнутый круг;
5. IDC\_SIZEALL — четырехлепестковая стрелка;
6. IDC\_SIZENESW — двухлепестковая стрелка, северо-восток и юго-запад;
7. IDC\_SIZENWSE — двухлепестковая стрелка, северо-запад и юго-восток;
8. IDC\_SIZENS — двухлепестковая стрелка, север и юг;
9. IDC\_SIZEWE — двухлепестковая стрелка, запад и восток;
10. IDC\_UPARROW — стрелка вверх;
11. IDC\_WAIT — песочные часы;

wc.lpszMenuName = NULL; // Ссылка на строку главного меню, при его отсутствии NULL

wc.cbClsExtra = 0; // Дополнительные параметры класса окна

wc.cbWndExtra = 0; // Дополнительные параметры окна

wc.hbrBackground = (HBRUSH) (COLOR\_WINDOW+1); // Дескриптор кисти, которая

/\* используется для заполнения окна. Стандартная конструкция, создает системную кисть белого цвета WHITE\_BRUSH. Требуется явное преобразование типа — HBRUSH \*/

После того как определены основные характеристики окна, можно это окно создать при помощи **API**-функции Createwindow(), где также нужно задать параметры:

1. WinName — имя, которое присвоено классу окна.
2. \_T("Каркас Windows-приложения") — заголовок окна в виде строки Unicode либо C-строки.
3. WS\_OVERLAPPEDWINDOW — макрос, определяющий стиль отображения стандартного окна, имеющего системное меню, заголовок, рамку для изменения размеров, а также кнопки минимизации, развертывания и закрытия. Это наиболее общий стиль окна, он определен так:
   * 1. #define WS\_OVERLAPPEDWINDOW (WS\_OVERLAPPED|WS\_CAPTION|WS\_SYSMENU| WS\_THICKFRAME|WS\_MINIMIZEBOX|WS\_MAXIMIZEBOX)
4. Можно создать другой стиль, используя комбинацию стилевых макросов при помощи операции логического сложения, вот некоторые из них:
5. WS\_OVERLAPPED — стандартное окно с рамкой;
6. WS\_CAPTION — окно с заголовком;
7. WS\_THICKFRAME — окно с рамкой;
8. WS\_MAXIMIZEBOX — кнопка распахивания окна;
9. WS\_MINIMIZEBOX — кнопка минимизации;
10. WS\_SYSMENU — системное меню;
11. WS\_HSCROLL — горизонтальная панель прокрутки;
12. WS\_VSCROLL — вертикальная панель прокрутки;
13. WS\_VISIBLE — окно отображается;
14. WS\_CHILD — дочернее окно;
15. WS\_POPUP — всплывающее окно;
16. Следующие два параметра определяют координаты левого верхнего угла окна (x, y). еще два параметра: Width — ширину и Height — высоту окна в пикселах. Задание параметра CW\_USEDEFAULT означает, что система сама выберет для отображения окна наиболее (с ее точки зрения) удобное место и размер.
17. Следующий параметр — указатель на структуру меню, или NULL, при его отсутствии.
18. Далее требуется указать дескриптор приложения, владельца окна — This.
19. И, наконец, указатель на дополнительную информацию, в нашем случае — NULL.
20. Окно создано, и с ним можно работать, но пока оно не отображается. Для того чтобы окно увидеть, необходимо его отобразить с помощью функции ShowWindow(hWnd, mode), которая принимает два параметра: hWnd — дескриптор окна и mode — режим отображения. В нашем случае мы используем значение, полученное при открытии приложения через параметр головной функции.
21. Далее, заключительная часть головной функции — цикл обработки сообщений. Он задается оператором while, аргументом которого является функция GetMessage(&msg, NULL, 0, 0). Такой цикл является обязательным для всех Windows-приложений, его цель — получение и обработка сообщений, передаваемых операционной системой. Операционная система ставит сообщения в очередь, откуда они извлекаются функцией GetMessage() по мере готовности приложения:
22. первым параметром функции является &msg — указатель на структуру MSG, где и хранятся сообщения;
23. второй параметр hwnd — определяет окно, для которого предназначено сообщение, если же необходимо перехватить сообщения всех окон данного приложения, он должен быть NULL;
24. остальные два параметра определяют [min, max] диапазон получаемых сообщений. Чаще всего необходимо обработать все сообщения, тогда эти параметры должны быть равны 0.
25. ***Примечание***
26. Сообщения определяются их номерами, символические имена для них определены и файле включений winuser.h. Префикс всех системных сообщений WM\_.
27. Внутри цикла расположены две функции:
28. TranslateMessage(&msg);
29. DispatchMessage(&msg);
30. Первая из них транслирует код нажатой клавиши в клавиатурные сообщения WM\_CHAR. При этом в переменную wParam структуры msg помещается код нажа­той клавиши в Windows-кодировке CP-1251, в младшее слово lParam — количество повторений этого сообщения в результате удержания клавиши в нажатом состоянии, а в старшее слово — битовая карта со значениями (клавиша отпущена или нажата, нажата клавиша <Alt>, нажата функциональная клавиша, Scan-код клавиши).

Использование этой функции не обязательно и нужно только для обработки сооб­щений от клавиатуры.

Вторая функция, DispatchMessage(&msg), обеспечивает возврат преобразо­ванного сообщения обратно операционной системе и инициирует вызов оконной функции данного приложения для его обработки.

Данным циклом и заканчивается головная функция.

Нам осталось лишь описать оконную функцию WndProc(), и построение каркаса Windows-приложения будет закончено.

Основной компонент этой функции — переключатель switch, обеспечивающий выбор соответствующего обработчика сообщений по его номеру message. В нашем случае мы предусмотрели обработку лишь одного сообщения WM\_DESTROY. Это сообщение посылается, когда пользователь завершает программу. Получив его, оконная функция вызывает функцию PostQuitMessage(0), которая завершает приложение и передает операционной системе код возврата — 0. Если говорить точ­нее, генерируется сообщение WM\_QUIT, получив которое функция GetMessage() возвращает нулевое значение. В результате цикл обработки сообщений прекращает­ся и происходит завершение работы приложения.

Все остальные сообщения обрабатываются по умолчанию функцией DefWindowProc() имеющей такой же список параметров и аналогичное возвращаемое значение, поэ­тому ее вызов помещается после оператора return.

**Стандартная заготовка Windows-приложения**

Мастер Visual Studio позволяет автоматически генерировать стандартную заготовку Windows-приложения. Для этого в стартовом окне построителя Win32 приложения (см. рис. 1.3) достаточно выбрать кнопку Finish. Проект состоит из набора файлов, изучение которых, студенты осуществляют самостоятельно.

**О цветах**

Цветные дисплеи требуют более одного бита для хранения информации о пикселе. Больше битов — больше цветов. Число уникальных цветов равно 2 в степени, равной числу битов. 16-цветные видеоадаптеры требуют 4 бита на пиксель. Эти биты организованы в цветовые плоскости — красная плоскость, зеленая плоскость, голубая плоскость и плоскость интенсивности. Адаптеры с 8, 16 или 24 битами на пиксель имеют одну цветовую плоскость, в которой набор смежных битов представляет цвет каждого пикселя.

Windows использует без знаковое 32-разрядное длинное целое для представления цвета. Тип данных для цвета называется COLORREF. Младшие три байта задают красную, зеленую и голубую составляющие, величина которых находится в интервале от 0 до 255.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 | … | 24 | 23 | … | 16 | 15 | … | 8 | 7 | … | 0 |

0 blue green red

**Рис.** 32-разрядное представление цвета

Таким образом, палитра может иметь (примерно 16 миллионов) цветов. Это без знаковое длинное целое часто называют "RGB-цвет". Заголовочный файл Windows содержит несколько макросов для работы со значениями RGB. Макрос RGB получает три аргумента, представляющих красную, зеленую и голубую составляющие, и конструирует из них без знаковое длинное целое:

#define RGB(r, g, b) ((COLORREF)(((BYTE)(r) | ((WORD)(g) << 8)) | ((DWORD)(BYTE)(b)) << 16)))

Таким образом, величина:

RGB(255, 0, 255)

равна 0x00FF00FF, значению RGB для пурпурного цвета (magenta). Когда все три аргумента равны 0, цвет — черный; когда все три равны 255 — белый.