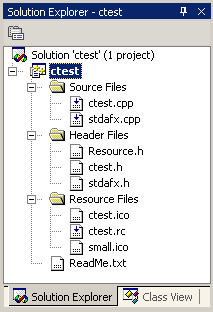
## Простейшая программа на С++

В первых частях книги мы достаточно часто будем использовать шаблон **Win32 Application**, который создает нам Visual C++. Поэтому сейчас необходимо подробнее рассмотреть некоторые вещи, которые нам пригодятся уже в ближайшем будущем. В специализированной литературе по Visual C++ вы сможете найти более детальное описание, а сейчас мы поговорим о самом необходимом.

В следующих главах мы еще много раз будем говорить о том, что сгенерировал нам мастер, и постепенно вы сможете разобраться со всем происходящим. Объяснять все сразу нет необходимости, потому что вы не сможете запомнить слишком много информации, которой нельзя найти применение.

Итак, откройте проект ctest и обратите внимание на панель **Solution Explorer**. Здесь расположено дерево, в котором по разделам сгруппировано все, что входит в проект. Мое дерево вы можете увидеть на рис. 1.10, и у вас должно быть что-то подобное.



**Рис. 1.10.** Дерево проекта

В нашем проекте есть следующие разделы:

* **Source Files** (Файлы исходного кода) — здесь хранятся файлы с исходным кодом программы. Основной код расположен в файле с именем как   
  у проекта и расширением cpp. В данном случае это ctest.cpp;
* **Header Files** (Заголовочные файлы) — здесь хранятся файлы описания, содержащие различную вспомогательную информацию, подключаемые модули, описания классов;
* **Resource Files** (Файлы ресурсов) — здесь сейчас хранятся файлы иконок и файл ресурсов ctest.rc.

### Ресурсы проекта

Давайте посмотрим содержание ресурсов. Щелкните дважды по имени файла ctest.rc, и в этой же панели появится вкладка **Resource View**. В ней   
в виде дерева представлены ресурсы, разбитые по разделам.

В разделе **Accelerator** находится список горячих клавиш для программы. Там может быть несколько ресурсов, но мастер пока создал один с именем IDC\_CTEST. Мы не будем использовать горячие клавиши, поэтому из данного раздела можно все удалить. Для этого щелкните правой кнопкой по ресурсу и в появившемся меню выберите пункт **Delete**. С одной стороны,   
в программе ничего лишнего не должно быть, а с другой — много места мы не выиграем.

Раздел **Dialog** содержит диалоговые окна. Если программа будет невидима, то и никакие окна ей не нужны. В нашем шаблоне по умолчанию находится окно IDD\_ABOUTBOX — окно с информацией о программе. Для удале-  
ния окна нужно также щелкнуть правой кнопкой и выбрать в меню пункт **Delete**.

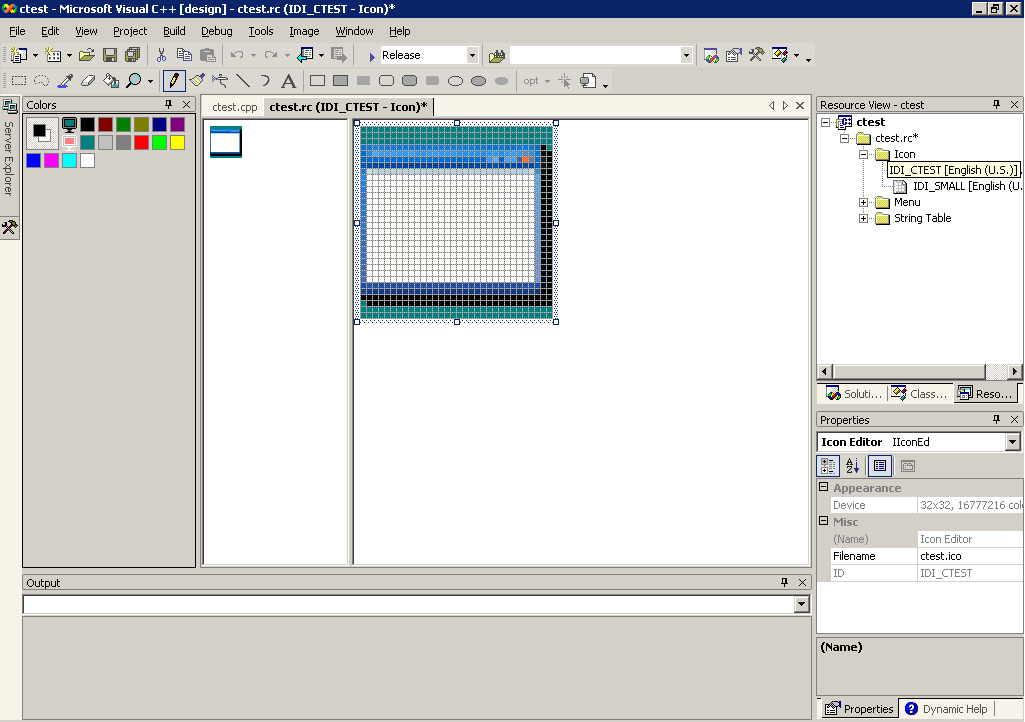
В разделе **Icon** находятся иконки программы. Иконок может быть несколько, неодинакового размера и c разным количеством цветов. Очень часто они нужны для шуточных программ, но не должны вызывать подозрения у пользователя при их запуске. Если вы сами будете просить запустить какую-то программу, то это будет не так впечатляюще, а вот когда он все сделает сам, то эффект неожиданности придаст вашей шутке большую остроту.

Чтобы пользователь запустил программу, иконка должна быть знакома и не вызывать подозрений. Например, для шуточной программы можно назначить иконку программы MS Word. Если в системе у пользователя прячутся расширения для стандартных файлов (используется по умолчанию), то он подумает, что перед ним Word-документ. А если еще и название файла заманчивое, то он обязательно его запустит.

Чтобы изменить иконку и нарисовать нечто оригинальное, вы должны дважды щелкнуть на ее названии, и в главном окне откроется простой графический редактор (рис. 1.11), в котором можно изменить изображение. Ничего серьезного в нем нарисовать невозможно, поэтому лучше вставить уже готовое изображение (например, через буфер обмена). Что-то подходящее всегда можно найти в Интернете.

В разделе **Menu** можно создавать меню для программы. С этим мы познакомимся на практике немного позже, и для демонстрации примеров мы будем очень часто использовать меню.

В разделе **String Table** можно хранить строки в виде констант. По умолчанию там находятся названия заголовков окон. Эти строки много места не занимают, поэтому их можно и оставить.



**Рис. 1.11.** Редактор иконки

### Код программы

Теперь познакомимся с кодом программы, который нам сгенерировал мастер. Он находится в файле ctest.cpp, и вы можете его увидеть в листинге 1.1. Весь код мы, конечно же, рассматривать не будем. Данная книга не является самоучителем по языку С++, хотя необходимые вещи рассматриваются очень подробно. Если вы уже знакомы с этим языком программирования, то для вас код файла должен быть понятен. Если нет, то достаточно того, что мы сейчас рассмотрим.

Листинг 1.1. Исходный код файла ctest.cpp

#include "stdafx.h"

#include "ctest.h"

#define MAX\_LOADSTRING 100

// Global Variables

// (Глобальные переменные):

HINSTANCE hInst; // current instance

// (текущий интерфейс)

TCHAR szTitle[MAX\_LOADSTRING]; // The title bar text

//(Заголовок окна)

TCHAR szWindowClass[MAX\_LOADSTRING]; // The main window class name

// (Имя класса главного окна)

// Forward declarations of functions included in this code module:

// Описание процедур, используемых в этом модуле:

ATOM MyRegisterClass(HINSTANCE hInstance);

BOOL InitInstance(HINSTANCE, int);

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

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

int APIENTRY \_tWinMain(HINSTANCE hInstance,

HINSTANCE hPrevInstance,

LPTSTR lpCmdLine,

int nCmdShow)

{

// TODO: Place code here

//(Поместите свой код здесь)

MSG msg;

HACCEL hAccelTable;

// Initialize global strings

// (Инициализация глобальных строк)

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

LoadString(hInstance, IDC\_CTEST, szWindowClass, MAX\_LOADSTRING);

MyRegisterClass(hInstance);

// Perform application initialization:

// (Инициализация приложения:)

if (!InitInstance (hInstance, nCmdShow))

{

return FALSE;

}

hAccelTable = LoadAccelerators(hInstance, (LPCTSTR)IDC\_CTEST);

// Main message loop:

//(Главный цикл обработки сообщений:)

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

{

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

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

return (int) msg.wParam;

}

// FUNCTION (Функция): MyRegisterClass()

// PURPOSE (Предназначение): Registers the window class

// (Регистрация класса окна)

// COMMENTS (Комментарии):

// This function and its usage are only necessary if you want this code

// to be compatible with Win32 systems prior to the 'RegisterClassEx'

// function that was added to Windows 95.

// It is important to call this function so that the application

// will get 'well formed' small icons associated with it.

// (Эта функция и ее использование необходимы, только если вы хотите,

// чтобы этот код был совместим с системой Win32 до функции

// 'RegisterClassEx', которая была добавлена в Windows 95.

// Это важно, вызвать эту функцию так, чтобы приложение получило

// 'хорошо отфарматированную' маленькую иконку, ассоциированную с ним.)

ATOM MyRegisterClass(HINSTANCE hInstance)

{

WNDCLASSEX wcex;

wcex.cbSize = sizeof(WNDCLASSEX);

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

wcex.lpfnWndProc = (WNDPROC)WndProc;

wcex.cbClsExtra = 0;

wcex.cbWndExtra = 0;

wcex.hInstance = hInstance;

wcex.hIcon = LoadIcon(hInstance, (LPCTSTR)IDI\_CTEST);

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

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

wcex.lpszMenuName = (LPCTSTR)IDC\_CTEST;

wcex.lpszClassName = szWindowClass;

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

return RegisterClassEx(&wcex);

}

// FUNCTION (Функция): InitInstance(HANDLE, int)

// PURPOSE (Предназначение): Saves instance handle and creates main

// window

// (Функция сохраняет указатель экземпляра и создает окно)

// COMMENTS (Комментарии):

// In this function, we save the instance handle in a global variable // and create and display the main program window.

// (В этой функции мы сохраняем указатель экземпляра в глобальной

// переменной, создаем и отобажаем главное окно.)

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)

{

HWND hWnd;

hInst = hInstance; // Store instance handle in our global variable

// (Сохраняем указатель экземпляра в глобальной переменной)

hWnd = CreateWindow(szWindowClass, szTitle, WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, 0, CW\_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);

if (!hWnd)

{

return FALSE;

}

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

return TRUE;

}

// FUNCTION (Функция): WndProc(HWND, unsigned, WORD, LONG)

// PURPOSE (Предназначение): Processes messages for the main window

// (Обработка сообщений главного окна)

// WM\_COMMAND — process the application menu

// (обработка меню приложения)

// WM\_PAINT — Paint the main window

// (Прорисовка окна)

// WM\_DESTROY — post a quit message and return

// (отправка сообщения о выходе из программы))

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

{

int wmId, wmEvent;

PAINTSTRUCT ps;

HDC hdc;

switch (message)

{

case WM\_COMMAND:

wmId = LOWORD(wParam);

wmEvent = HIWORD(wParam);

// Parse the menu selections:

// Проверка выбранного меню:

switch (wmId)

{

case IDM\_ABOUT:

DialogBox(hInst, (LPCTSTR)IDD\_ABOUTBOX, hWnd, (DLGPROC)About);

break;

case IDM\_EXIT:

DestroyWindow(hWnd);

break;

default:

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

}

break;

case WM\_PAINT:

hdc = BeginPaint(hWnd, &ps);

// TODO: Add any drawing code here...

EndPaint(hWnd, &ps);

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

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

}

return 0;

}

// Message handler for about box

// (Обработчик сообщения для окна "О программе")

// Мы окно о программе удалили, поэтому следующий код можно удалять

LRESULT CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)

{

switch (message)

{

case WM\_INITDIALOG:

return TRUE;

case WM\_COMMAND:

if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)

{

EndDialog(hDlg, LOWORD(wParam));

return TRUE;

}

break;

}

return FALSE;

}

Мы уже знаем, что у нашей программы не должно быть никаких окон. Из проекта в разделе **Dialog** окно **О программе** мы уже удалили *(см. разд. 1.3.1)*. Но в коде еще остались ссылки на него, поэтому вы не сможете выполнить программу. Чтобы проект запустился, удалите все, что находится после следующей строки:

// Мы окно о программе удалили, поэтому следующий код можно удалять

Этот код отображает окно **О программе**, и его можно удалять полностью.

Теперь перейдите в процедуру WndProc и удалите здесь вызов процедуры About. Для этого нужно найти и удалить следующие три строчки:

case IDM\_ABOUT:

DialogBox(hInst, (LPCTSTR)IDD\_ABOUTBOX, hWnd, (DLGPROC)About);

break;

Это обработчик события для пункта меню **Help/About** нашей программы. Все обработчики находятся в функции WndProc и имеют следующую структуру:

case Идентификатор

Действия

break;

Здесь Идентификатор — это константа, которая назначена элементу управления (например, пункту меню). Оператор case проверяет, если пришло   
событие от элемента управления с указанным идентификатором, то выполняется последующий код до оператора break. Чуть позже мы познакомимся с событиями на практике.

Вот теперь мы удалили все, что нам не понадобится. Можно убрать еще меню или полностью окно, когда у вас невидимое приложение, но в данной книге все это нам может понадобиться для наглядной иллюстрации происходящего, поэтому остановимся на таком шаблоне.

Но когда мы будем создавать невидимые приложения, достаточно только найти в коде и удалить следующие две строки:

ShowWindow(hWnd, nCmdShow);

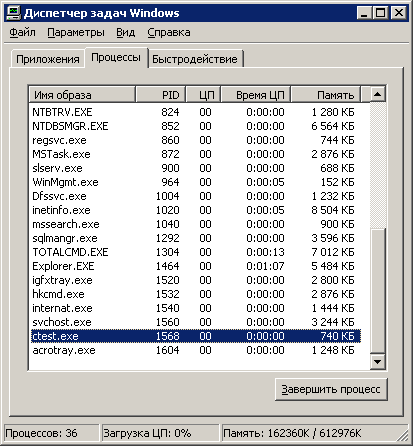
UpdateWindow(hWnd);

Эти строки находятся в процедуре InitInstance, которая предназначена для создания и отображения окна на рабочем столе. Создание окна можно и не убирать, а вот для того, чтобы программа стала невидимой, отображать ничего не надо.

Первая строка кода показывает созданное в данной программе окно, а вторая — обновляет его содержимое. Вы можете закомментировать строки, поставив перед ними две косые черты (//). Попробуйте теперь скомпилировать и запустить программу. Вы ничего не увидите ни на экране, ни по нажатию клавиш <Ctrl>+<Alt>+<Del>. Если у вас Windows 2000/XP, то только на вкладке **Процессы** окна **Диспетчер задач Windows** вы сможете найти в списке свою программу (рис. 1.12).

Если вы не имели опыта программирования на Visual C++ и сейчас чего-то не поняли, не расстраивайтесь. Постепенно все встанет на свои места.   
В дальнейшем мы рассмотрим достаточно много из того, что вы видите   
в исходном коде.

Рассмотрим подробнее некоторые части представленного кода. Программа начинает выполнение с функции \_tWinMain (листинг 1.2).



**Рис. 1.12.** Программа ctest среди процессов

Листинг 1.2. Стартовая функция

int APIENTRY \_tWinMain(HINSTANCE hInstance,

HINSTANCE hPrevInstance,

LPTSTR lpCmdLine,

int nCmdShow)

{

// Дальше идет объявление двух переменных

MSG msg;

HACCEL hAccelTable;

// Инициализация строковых переменных

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

LoadString(hInstance, IDC\_CTEST, szWindowClass, MAX\_LOADSTRING);

MyRegisterClass(hInstance);

// Выполнение инициализации приложения

if (!InitInstance (hInstance, nCmdShow))

{

return FALSE;

}

hAccelTable = LoadAccelerators(hInstance, (LPCTSTR)IDC\_CTEST);

// Главный цикл обработки сообщений Windows

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

{

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

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

return (int) msg.wParam;

}

Мы уже много раз использовали слово "функция", но не дали ей определение. Если у вас есть опыт программирования в С++, то вы должны быть знакомы с этим понятием. Некоторые вещи выходят за рамки данной книги, и я их буду опускать. Более подробную информации о языке С++ вы можете узнать в специализированной литературе. И все же мы рассмотрим максимальное количество информации, необходимой для понимания примеров.

Все функции в С++ объявляются следующим образом:

Тип Имя (Параметры)

{

}

* Тип— тип возвращаемого значения. Если используется int, это указывает на число целого типа.
* Имя— может быть любым, но для главной функции, с которой начинается выполнение программы, оно предопределено.
* Параметры— переменные и различные значения, которые передаются   
  в функцию для использования внутри нее.

У нашей главной функции после возвращаемого типа стоит ключевое слово APIENTRY, которое указывает на точку входа программы.

Теперь посмотрите на листинг 1.1. Здесь я расставил комментарии, чтобы вы понимали, что происходит. Как мы уже знаем, комментарии начинаются с двойной косой черты (//). Текст, который стоит после этих черточек, не влияет на работу программы, а только поясняет код.

В самом начале функции объявляются две переменные:

MSG msg;

HACCEL hAccelTable;

При объявлении указывают Тип и Имя. По типу программа определяет количество памяти, которое надо выделить под хранение значения переменной. К этой памяти можно обратиться с помощью указанного имени.   
В С++ достаточно много типов переменных, и с основными из них мы познакомимся в процессе изучения примеров.

Работа с простыми переменными (строка, число, структура) никаких дополнительных действий не требует. Но если это объект или указатель, то им нужно выделить память. Объекты используются при программировании   
с использованием MFC, а указатели — это переменные, которые указывают на определенную область памяти, выделенную для хранения данных.

Зачем нужны указатели? Многие не понимают их мощи или просто боятся использовать из-за их незащищенности и возможности вылета за пределы выделенной области памяти. Представьте себе, что у вас в память загружена картинка размером в мегабайт, и вы должны дать возможность какой-то функции читать ее данные. В этом случае нужно переслать в функцию целый мегабайт, что отнимет не только много времени, но и лишнюю память. Вместо этого можно показать функции, куда загружена картинка, т. е. передать указатель на память с данными изображения.

Второй способ — использование глобальных переменных, что не рекомендуется делать. Глобальные переменные видны в любой функции. Их принято определять в заголовочном файле (файл с расширением h) или до описания функций, в самом начале файла.

Локальные переменные объявляются внутри функции, и к ним можно обратиться только в ней. Такие переменные автоматически создаются при запуске функции в специальной области памяти (стеке) и автоматически уничтожаются при выходе из нее. Автоматическое создание/удаление относится только к простым переменным, но не к указателям, которые желательно освобождать вручную.

После объявления переменных идут следующие две строки:

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

LoadString(hInstance, IDC\_CTEST, szWindowClass, MAX\_LOADSTRING);

Эти две функции с именем LoadString загружают текст из ресурсов строк. Функция — это часть кода, которая имеет имя и может вызываться из других мест программы. В данном случае выполнится код загрузки ресурса. Чтобы функция узнала, какой ресурс нам нужен, и куда его загрузить, существуют параметры, которые передаются в скобках. Параметры перечисляются через запятую. В данном случае их четыре.

* HInstance — указывает на экземпляр нашей программы, потому что нужны ресурсы из нашего проекта.
* IDS\_APP\_TITLE — имя ресурса, который надо загрузить. Если вы сейчас дважды щелкните в ресурсах на разделе String Table, то перед вами откроется таблица строк, в которой в одной колонке будет имя строки, а во второй текст. Вот именно это имя и нужно указывать в этом параметре.
* szTitle — переменная, в которую должно поместиться значение. В начале файла у нас объявлено две переменные с именами szTitle и szWindowClass:

TCHAR szTitle[MAX\_LOADSTRING]; // Текст заголовка окна

TCHAR szWindowClass[MAX\_LOADSTRING]; // Имя класса главного окна

Как мы уже знаем, при объявлении переменных вначале идет тип.   
В данном случае указан TCHAR, что означает строку. Далее идет имя переменной. А для строк еще надо указать в квадратных скобках размер (максимальную длину в символах). В качестве размера указано MAX\_LOADSTRING. Это константа, которая равна максимальному размеру загружаемых символов. Можно было бы указать в квадратных скобках и реальное число, но если есть возможность, то лучше использовать предопределенные константы.

* MAX\_LOADSTRING — последний параметр, который указывает максимальное количество загружаемых символов. Тут опять применяется константа, которая является и длиной строк, в которые мы загружаем текст из ресурса. Получается, что размер загружаемой строки равен размеру строки в переменной, и мы никогда не сможем загрузить из ресурсов в переменную больше, чем выделено памяти.

После этого идет вызов функции MyRegisterClass(hInstance). В ней происходит заполнение структуры WNDCLASSEX. Что такое структура? Это особая переменная, которая хранит в себе набор переменных любого типа. Например, структура может хранить одну переменную с именем Age числового типа и одну строкового — с именем Name. Чтобы прочитать или изменить значение этих переменных, нужно написать Структура.Переменная. Структура — это имя структурной переменной, а Переменная — это имя переменной.

WNDCLASSEX — это структура, которая используется при создании нового класса окна. Для минимального приложения нам понадобится заполнить следующие поля (основные):

* style — стиль окна;
* Lpfnwndproc — указатель на процедуру, которая будет вызываться на все пользовательские или системные события;
* Hinstance — манипулятор, который мы получили при запуске программы в процедуре \_tWinMain;
* HbrBackground — цвет фона (в принципе, он необязателен, но по умолчанию используется цвет окна);
* LpszClassName — имя создаваемого класса;
* Hcursor — курсор. Сюда загружается стандартный курсор-стрелка.

Все, структура готова, и мы можем зарегистрировать новый класс будущего окна. Для этого вызывается функция WinAPI RegisterClassEx(&wcex). После этого в системе есть описание вашего будущего окна. Почему будущего?   
Да потому, что само окно мы еще не создали. Для этого нужно еще вызвать функцию CreateWindow (это происходит в функции InitInstance, которая в свою очередь вызывается в \_tWinMain после вызова MyRegisterClass):

hWnd = CreateWindow(szWindowClass, szTitle, WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, 0, CW\_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);

У нее достаточно много параметров, и давайте посмотрим на них внимательнее.

* Имя класса. Мы зарегистрировали класс и сохранили имя в переменной szWindowClass, значит и здесь мы должны указать именно этот класс.
* Имя окна. Это всего лишь заголовок, который будет выводиться в окне. Его мы уже загрузили с помощью функции LoadString и сохранили   
  в переменной szTitle.
* Стиль окна. Нас интересует простейшее WS\_OVERLAPPEDWINDOW окно.
* Следующие четыре параметра — это левая и правая позиции, ширина и высота окна. Если указать все параметры равными нулю или CW\_USEDEFAULT, то значения будут выбраны по умолчанию.
* Главное окно по отношению к создаваемому. Наше окно само по себе главное, поэтому указываем NULL, что соответствует нулю.

Остальные параметры нас пока не интересуют. После создания окна его   
надо отобразить. Делается это с помощью вызова процедуры ShowWindow,   
о которой мы уже немного говорили. У этой процедуры использованы два параметра:

* созданное окно;
* параметры отображения окна.

Здесь указано nCmdShow, значение, которое передается программе в зависимости от параметров, указанных в свойстве ярлыка, вызывающего программу. Остальные значения параметра можно посмотреть в файле справки по WinAPI-функциям.

И последняя подготовительная функция — UpdateWindow. Это просто отрисовка созданного окна.

Теперь разберемся с циклом обработки сообщений. Функция GetMessage ожидает пользовательского или системного сообщения, и как только оно наступает, возвращает true (истина). Полученное сообщение преобразуется в необходимый вид с помощью TranslateMessage и отправляется обработчику сообщений с помощью вызова функции DispatchMessage.

В каждой программе должна быть процедура обработки сообщений. Какая именно? Мы указали ее при создании класса окна в свойстве WindowClass.Lpfnwndproc. В Visual C++ принято называть ее WndProc — стандартное имя, используемое по умолчанию. Сама же процедура должна выглядеть приблизительно как в листинге 1.1.

В процедуре-обработчике событий желательно всегда делать вызов функции defwindowproc. Эта функция ищет в системе обработчик полученного сообщения, установленный по умолчанию. Это очень важно, тогда вам не придется без особой необходимости самому писать то, что может сделать ОС. Обработка полученного сообщения происходит с помощью сравнения параметра message со стандартными событиями. Например, если message равен wm\_destroy, то это значит, что программа хочет уничтожиться, и тогда в обработчике можно освободить выделенную под программу память.

Вот и все, с шаблоном мы разобрались. Если вы сейчас запустите созданную программу, то перед вами появится пустое окно. Чтобы его закрыть, просто нажмите <Alt>+<F4> или кнопку закрытия окна.

Если вы захотите сделать это окно невидимым, то просто уберите из кода функцию ShowWindow, которая отображает окно на экране. Ваша программа сразу же станет невидимой в системе. Второй способ — изменить второй параметр этой процедуры на SW\_HIDE (внешне равносильно отсутствию вызова процедуры). Функцию ShowWindow используют с параметром SW\_HIDE, когда нужно спрятать окно в процессе выполнения программы без его уничтожения из памяти компьютера.

Чуть позже мы еще встретимся с процедурой ShowWindow, и не один раз.

Для компиляции проекта выберите команду меню **Build/Build Solution**.   
Таким образом, вы соберете проект и создадите запускаемый файл. Чтобы запустить программу, выберите команду меню **Debug/Start**.

PRIM

На компакт-диске в папке \Demo\Chapter1\empt\вы можете увидеть исходный код этого примера.