

# ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ

на **C++**

## Win32 API-приложения

- *Графический интерфейс Windows-приложения*
- *Элементы управления*
- *Создание дочерних и всплывающих окон*
- *Растровая графика*
- *Библиотеки динамической компоновки (DLL)*
- *Процессы и потоки*

УДК 681.3.068+800C++(075.8)  
ББК 32.973.26-018.1я73  
Л64

**Литвиненко Н. А.**

Л64 Технология программирования на C++. Win32 API-приложения. —  
СПб.: БХВ-Петербург, 2010. — 288 с.: ил. — (Учебное пособие)  
ISBN 978-5-9775-0600-7

Изложен начальный курс низкоуровневого программирования на C++ для Windows с использованием библиотеки Win32 API. Рассмотрены графический интерфейс Windows-приложения, стандартные диалоговые окна, элементы управления, растровая графика, DLL-библиотеки, процессы и потоки. Материал иллюстрирован многочисленными примерами, выполненными в Visual Studio 2010 под управлением Windows 7.

*Для студентов и преподавателей технических вузов и самообразования*

УДК 681.3.068+800C++(075.8)  
ББК 32.973.26-018.1я73

**Группа подготовки издания:**

Главный редактор	<i>Екатерина Кондукова</i>
Зам. главного редактора	<i>Евгений Рыбаков</i>
Зав. редакцией	<i>Григорий Добин</i>
Редактор	<i>Юрий Рожко</i>
Компьютерная верстка	<i>Натальи Караваевой</i>
Корректор	<i>Наталья Першакова</i>
Дизайн серии	<i>Инны Тачиной</i>
Оформление обложки	<i>Елены Беляевой</i>
Зав. производством	<i>Николай Тверских</i>

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 29.06.10.

Формат 70×100<sup>1/16</sup>. Печать офсетная. Усл. печ. л. 23,22.

Тираж 1000 экз. Заказ №

"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию  
№ 77.99.60.953.Д.005770.05.09 от 26.05.2009 г. выдано Федеральной службой  
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов  
в ГУП "Типография "Наука"  
199034, Санкт-Петербург, 9 линия, 12.

# Оглавление

<b>Введение.....</b>	<b>1</b>
<b>Глава 1. Интерфейс Windows-приложения.....</b>	<b>3</b>
Каркас Windows-приложения .....	4
Исследование каркаса Windows-приложения .....	9
Стандартная заготовка Windows-приложения .....	15
Обработка сообщений .....	21
Нажатие клавиши .....	21
Сообщение мыши .....	25
Создание окна .....	27
Таймер .....	27
Рисование в окне .....	29
Рисование линии.....	29
Прямоугольники, регионы и пути .....	46
Прямоугольники .....	46
Регионы .....	47
Пути .....	50
Области отсечения .....	52
Вывод текста.....	53
Цвет текста и фона .....	53
Получение метрики текста .....	54
Определение длины строки .....	55
Системные шрифты .....	56
Определение произвольных шрифтов.....	57
Диалог с пользователем .....	59
Окно сообщений .....	60
Меню .....	61
Пример интерактивной графики .....	63
Вопросы к главе .....	68
Задания для самостоятельной работы.....	69

<b>Глава 2. Работа с файлами .....</b>	<b>71</b>
Диалог выбора файлов .....	71
Простой просмотрщик файлов .....	72
Организация скроллинга .....	78
Панель инструментов .....	85
Выбор шрифтов.....	89
Чтение и запись файлов в библиотеке Win32 API.....	96
Вопросы к главе .....	100
Задания для самостоятельной работы.....	100
<b>Глава 3. Окна и элементы управления .....</b>	<b>103</b>
Дочерние окна .....	104
Всплывающие окна.....	109
Диалоговые окна .....	116
Тестирование элементов управления .....	118
Общие элементы управления .....	128
Окно редактирования .....	134
Строка состояния.....	140
Простой текстовый редактор на элементе управления Edit Box Control .....	141
Немодальные окна .....	148
Стандартное диалоговое окно выбора цвета .....	152
Вопросы к главе .....	155
Задания для самостоятельной работы.....	156
<b>Глава 4. Растровая графика .....</b>	<b>157</b>
Функция <i>BitBlt()</i> .....	157
Вывод изображения в заданный прямоугольник .....	160
Загрузка изображения из файла .....	161
Растровые операции .....	164
Анимация .....	167
Функция <i>PlgBlt()</i> .....	172
Функция <i>MaskBlt()</i> .....	177
Вращение графического образа.....	180
Виртуальное окно .....	183
Метафайлы .....	187
Создание дискового файла .....	190
Растровое изображение в метафайле.....	190
Расширенные метафайлы .....	192
Вопросы к главе .....	196
Задания для самостоятельной работы.....	196

<b>Глава 5. Библиотеки динамической компоновки DLL .....</b>	<b>197</b>
Создание DLL.....	197
Использование DLL.....	199
Неявное связывание .....	199
DLL общего использования .....	202
Явная загрузка DLL.....	204
Загрузка ресурсов из DLL.....	207
Вопросы к главе .....	210
Задания для самостоятельной работы.....	210
<b>Глава 6. Процессы и потоки .....</b>	<b>211</b>
Создание процесса .....	211
Создание потока.....	216
Функции C++ для создания и завершения потока .....	219
Измерение времени работы потока .....	220
Высокоточное измерение времени .....	223
Приоритеты потоков.....	225
Синхронизация потоков в пользовательском режиме.....	228
Interlocked-функции .....	228
Критические секции (critical section).....	230
Синхронизация с использованием объектов ядра .....	232
Семафоры.....	233
События.....	238
Мьютексы.....	241
Ожидаемые таймеры.....	242
Обмен данными между процессами.....	247
Разделяемая память для нескольких экземпляров exe-файла .....	247
Файлы, проецируемые в память.....	249
Совместный доступ к данным нескольких процессов.....	256
Передача данных через сообщение .....	260
Вопросы к главе .....	264
Задания для самостоятельной работы.....	265
<b>Приложение. Поиск окна.....</b>	<b>267</b>
Поиск всех окон, зарегистрированных в системе.....	267
Поиск главного окна созданного процесса .....	269
<b>Литература .....</b>	<b>273</b>
<b>Дополнительная литература .....</b>	<b>273</b>
<b>Предметный указатель .....</b>	<b>275</b>



## Глава 1

# Интерфейс Windows-приложения

Стиль программирования Windows-приложений принципиально отличается от того, который сложился в операционных системах раннего поколения. В MS-DOS программа монополюно владеет всеми ресурсами системы и является инициатором взаимодействия с операционной системой. Совсем иначе дело обстоит в операционной системе Windows, которая строилась как многозадачная, и именно операционная система является инициатором обращения к программе. Все ресурсы Windows являются разделяемыми, и программа, в дальнейшем будем называть ее *приложением*, не может владеть ими монополюно. В связи с такой идеологией построения операционной системы приложение должно ждать посылки сообщения операционной системы и лишь после его получения выполнить определенные действия, затем вновь перейти в режим ожидания очередного сообщения. На рис. 1.1 схематично изображена диаграмма типичного Windows-приложения.

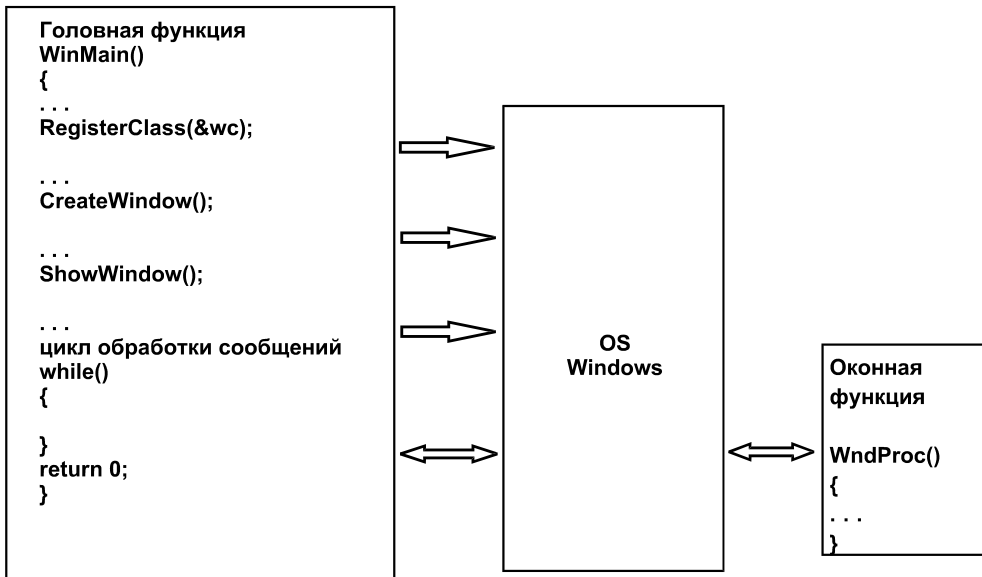


Рис. 1.1. Структура Windows-приложения

Windows генерирует множество различных сообщений, которые направляются приложению, например, щелчок кнопки мыши или нажатие клавиши на клавиатуре. Если приложение не обрабатывает какие-то сообщения, реакция на них осуществляется операционной системой стандартным способом, так что задачей программиста является обработка лишь тех сообщений, которые необходимы приложению.

Разработчиками операционной системы Windows была создана библиотека функций, при помощи которых и происходит взаимодействие приложения с операционной системой, так называемые функции *Программного интерфейса приложений* (*Application Program Interface, API*).

Подмножество этих функций, предназначенных для графического вывода на дисплей, графопостроитель и принтер, представляет собой *Интерфейс графических устройств* (*Graphics Device Interface, GDI*).

Библиотека API-функций разрабатывалась в расчете на то, что ее можно использовать для любого языка программирования, а поскольку разные языки имеют различные типы данных, то были созданы собственные Windows-типы, которые приводятся к типам данных языков программирования. Отметим только, что в Windows нет логического типа `bool`, но есть Windows-тип `BOOL`, который эквивалентен целому типу `int`. Будем рассматривать типы данных Windows по мере необходимости.

Еще одной особенностью API-функций является использование обратного, по отношению к принятому в языке C, порядка передачи параметров, как это реализовано в языке Pascal. В C для идентификации таких функций использовалось служебное слово `pascal`, в Windows введены его синонимы `CALLBACK`, `APIENTRY` или `WINAPI`. По умолчанию C-функции передают параметры, начиная с конца списка так, что первый параметр всегда находится на вершине стека. Именно это позволяет использовать в языке C функции с переменным числом параметров, что в API-функциях невозможно.

## 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.

### **ПРИМЕЧАНИЕ**

Термин *интерфейс* здесь следует понимать как способ взаимодействия пользователя и приложения. *Класс окна* — структура, определяющая его свойства.

Рассмотрим сначала, 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** выберем путь к рабочей папке, а в поле **Name** имя проекта (рис. 1.2). В следующем диалоговом окне, приведенном на рис. 1.3, нажимаем кнопку **Next**, а в окне опций проекта (рис. 1.4) выберем флажок **Empty project** (Пустой проект) и нажмем кнопку **Finish** — получим пустой проект, в котором нет ни одного файла.

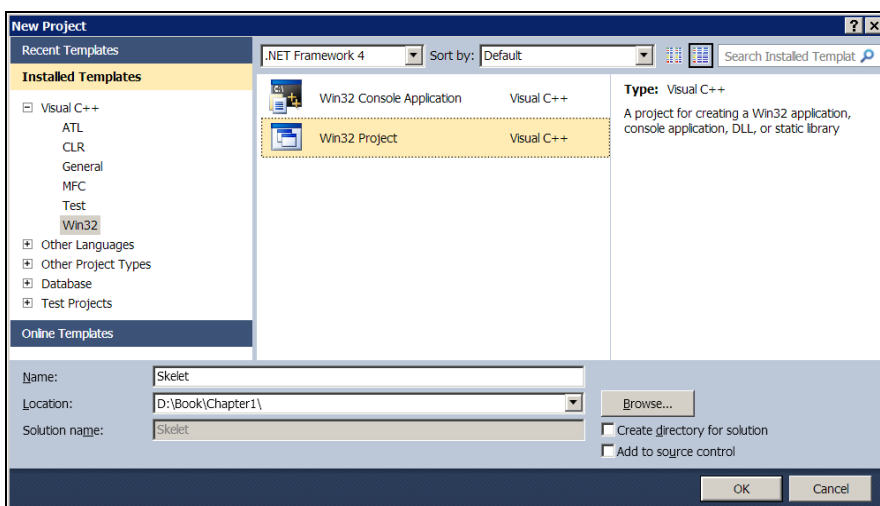


Рис. 1.2. Выбор типа проекта

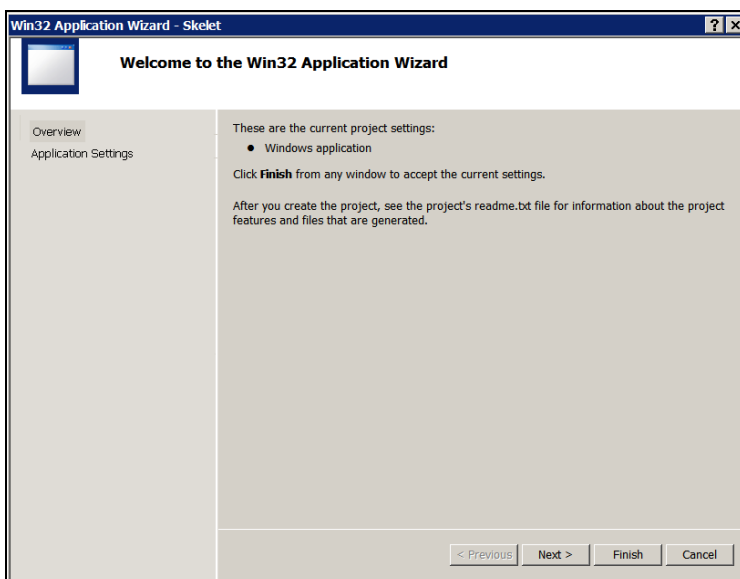


Рис. 1.3. Стартовое окно построителя приложения



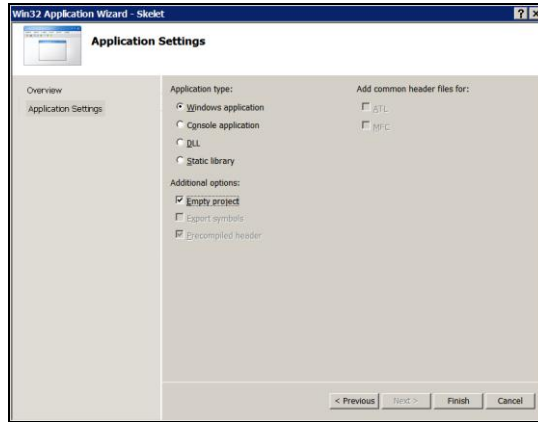


Рис. 1.4. Окно опций проекта

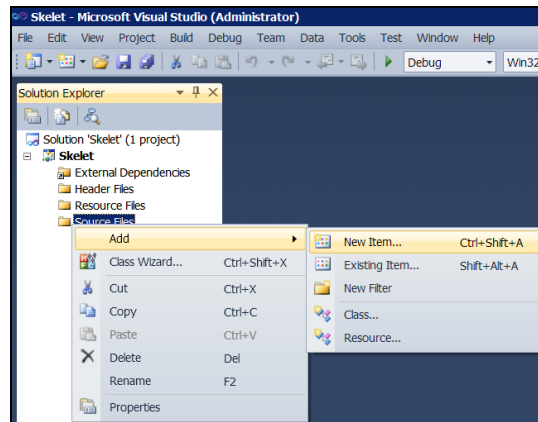


Рис. 1.5. Добавление к проекту нового объекта с помощью контекстного меню

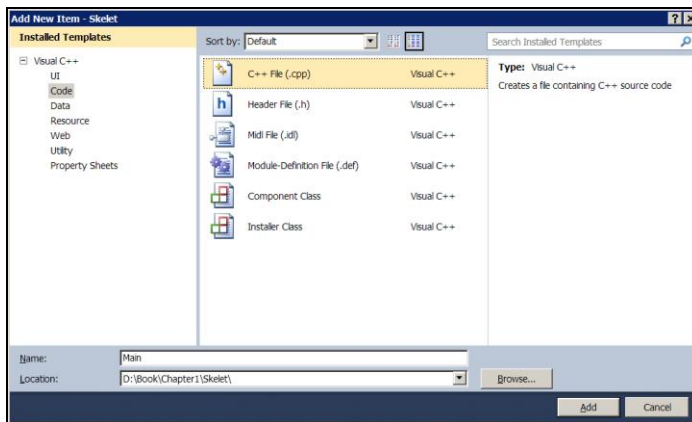


Рис. 1.6. Выбор шаблона объекта

С помощью контекстного меню (рис. 1.5) добавим файл для кода приложения, имя файла введем в ходе диалога выбора шаблона объекта на рис. 1.6. (Тот же самый диалог мы могли бы получить по команде меню **Project | Add New Item....**)

### ПРИМЕЧАНИЕ

Пока мы создаем простые решения, состоящие из одного проекта, можно убрать флажок **Create directory for solution**. Это упростит структуру каталога.

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 APIENTRY _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
    // Заполнение окна белым цветом
    wc.hbrBackground = (HBRUSH) (COLOR_WINDOW+1);
    if(!RegisterClass(&wc)) return 0; // Регистрация класса окна

    // 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)
{
    // Обработчик сообщений
    switch(message)
    {
        case WM_DESTROY : PostQuitMessage(0);
            break; // Завершение программы
        // Обработка сообщения по умолчанию
        default : return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}
```

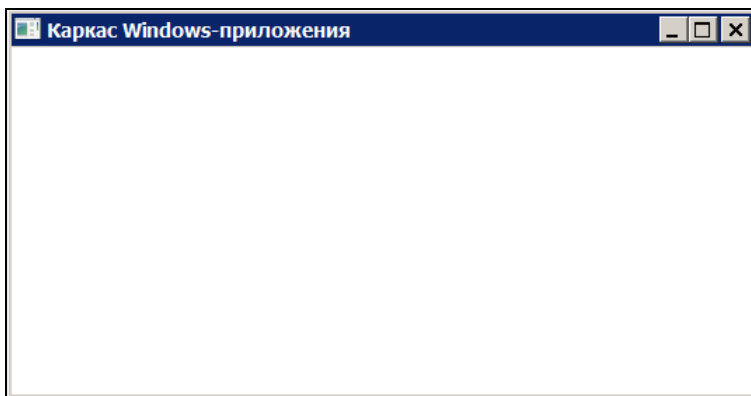


Рис. 1.7. Окно первой Windows-программы

Программа не делает ничего полезного, поэтому, запустив ее на выполнение кнопкой **▶ (Start Debugging)**, мы получим изображенное на рис. 1.7 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: `windows.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 _T(x)        __T(x)
#ifdef _UNICODE
#define __T(x)      L ## x
#else
#define __T(x)      x
#endif
```

Макрос преобразуется в оператор "L", который является инструкцией компилятору 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** (рис. 1.8). Диалоговое окно свойств **Property Pages** доступно сейчас на подложке **Property Manager** панели управления **Solution Explorer**.

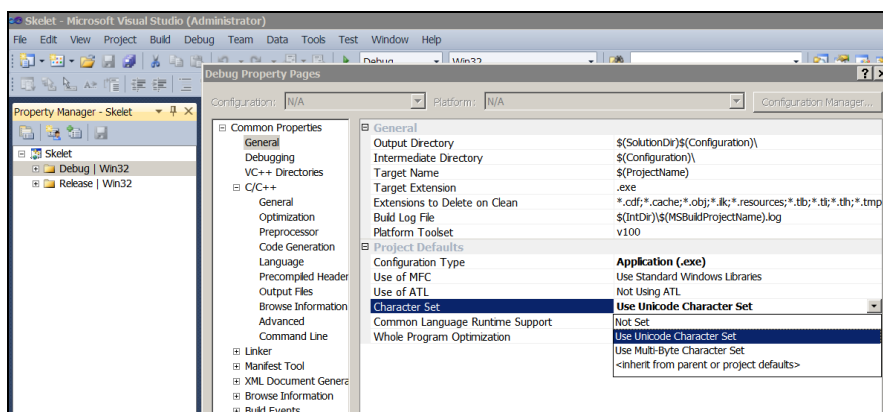


Рис. 1.8. Страница общих свойств проекта

Таким образом, этот макрос позволяет компилировать проект как в кодировке Unicode, так и в Windows-кодировке. Мы подробно рассмотрели данный макрос потому, что многие 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-проекта.

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 APIENTRY _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**

Здесь впервые появляется Windows-тип данных — *дескриптор* (описатель), который 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 отправки сообщения в миллисекундах
    POINT pt;           // Координаты курсора (x, y)
};
struct POINT
{
    LONG x, y;
};
```

**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`.

- `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.

- `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:

◇ `IDI_APPLICATION` — стандартная иконка;

◇ `IDI_ASTERISK` — звездочка;

- ◇ `IDI_EXCLAMATION` — восклицательный знак;
- ◇ `IDI_HAND` — ладонь;
- ◇ `IDI_QUESTION` — вопросительный знак;
- ◇ `IDI_WINLOGO` — логотип Windows;
- `wc.hCursor = LoadCursor(NULL, IDC_ARROW);`

Аналогичная функция `LoadCursor()` обеспечивает загрузку графического образа курсора, где нулевой первый параметр также означает использование системного курсора, вид которого можно выбрать из списка:

- ◇ `IDC_ARROW` — стандартный курсор;
- ◇ `IDC_APPSTARTING` — стандартный курсор и маленькие песочные часы;
- ◇ `IDC_CROSS` — перекрестие;
- ◇ `IDC_IBEAM` — текстовый курсор;
- ◇ `IDC_NO` — перечеркнутый круг;
- ◇ `IDC_SIZEALL` — четырехлепестковая стрелка;
- ◇ `IDC_SIZENESW` — двухлепестковая стрелка, северо-восток и юго-запад;
- ◇ `IDC_SIZENWSE` — двухлепестковая стрелка, северо-запад и юго-восток;
- ◇ `IDC_SIZENS` — двухлепестковая стрелка, север и юг;
- ◇ `IDC_SIZENE` — двухлепестковая стрелка, запад и восток;
- ◇ `IDC_UPARROW` — стрелка вверх;
- ◇ `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, разворачивания и закрытия. Это наиболее общий стиль окна, он определен так:

```
#define WS_OVERLAPPEDWINDOW (WS_OVERLAPPED|WS_CAPTION|WS_SYSMENU|  
WS_THICKFRAME|WS_MINIMIZEBOX|WS_MAXIMIZEBOX)
```

Можно создать другой стиль, используя комбинацию стилевых макросов при помощи операции логического сложения, вот некоторые из них:

- `WS_OVERLAPPED` — стандартное окно с рамкой;
  - `WS_CAPTION` — окно с заголовком;
  - `WS_THICKFRAME` — окно с рамкой;
  - `WS_MAXIMIZEBOX` — кнопка распахивания окна;
  - `WS_MINIMIZEBOX` — кнопка минимизации;
  - `WS_SYSMENU` — системное меню;
  - `WS_HSCROLL` — горизонтальная панель прокрутки;
  - `WS_VSCROLL` — вертикальная панель прокрутки;
  - `WS_VISIBLE` — окно отображается;
  - `WS_CHILD` — дочернее окно;
  - `WS_POPUP` — всплывающее окно;
4. 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.
5. Следующий параметр — указатель на структуру меню, или `NULL`, при его отсутствии.
6. Далее требуется указать дескриптор приложения, владельца окна — `This`.
7. И, наконец, указатель на дополнительную информацию, в нашем случае — `NULL`.

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.

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:

- первым параметром функции является `&msg` — указатель на структуру `MSG`, где и хранятся сообщения;



- второй параметр `hWnd` — определяет окно, для которого предназначено сообщение, если же необходимо перехватить сообщения всех окон данного приложения, он должен быть `NULL`;
- остальные два параметра определяют `[min, max]` диапазон получаемых сообщений. Чаще всего необходимо обработать все сообщения, тогда эти параметры должны быть равны 0.

### ПРИМЕЧАНИЕ

Сообщения определяются их номерами, символические имена для них определены в файле включений `winuser.h`. Префикс всех системных сообщений `WM_`.

Внутри цикла расположены две функции:

```
TranslateMessage (&msg);
```

```
DispatchMessage (&msg);
```

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, приведенными в табл. 1.1.

**Таблица 1.1.** Битовая карта клавиатуры, `HIWORD (lParam)`

Бит	Значение
15	1, если клавиша отпущена, 0 — если нажата
14	1, если клавиша была нажата перед посылкой сообщения
13	1, если нажата клавиша <Alt>
12—9	Резерв
8	1, если нажата функциональная клавиша
7—0	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,



```

#include "targetver.h"
#define WIN32_LEAN_AND_MEAN //Отключает некоторые редко используемые
//возможности компилятора, для ускорения компиляции
#include <windows.h> //Стандартный набор файлов включений
#include <stdlib.h> //для Win32-проекта
#include <malloc.h>
#include <memory.h>
#include <tchar.h>

//Standard.h////////////////////////////////////
#pragma once
#include "resource.h"

//targetver.h////////////////////////////////////
#pragma once
#include <SDKDDKVer.h>

//resource.h////////////////////////////////////
#define IDS_APP_TITLE 103
#define IDR_MAINFRAME 128
#define IDD_STANDARD_DIALOG 102
#define IDD_ABOUTBOX 103
#define IDM_ABOUT 104
#define IDM_EXIT 105
#define IDI_STANDARD 107
#define IDI_SMALL 108
#define IDC_STANDARD 109
#define IDC_MYICON 2
#ifndef IDC_STATIC
#define IDC_STATIC -1
#endif
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NO_MFC 130
#define _APS_NEXT_RESOURCE_VALUE 129
#define _APS_NEXT_COMMAND_VALUE 32771
#define _APS_NEXT_CONTROL_VALUE 1000
#define _APS_NEXT_SYMED_VALUE 110
#endif
#endif
#endif

//stdafx.cpp //////////////////////////////////////
#include "stdafx.h"

//Standard.cpp //////////////////////////////////////

```

```
#include "stdafx.h"
#include "Standard.h"
#define MAX_LOADSTRING 100
HINSTANCE hInst;
TCHAR szTitle[MAX_LOADSTRING];
TCHAR szWindowClass[MAX_LOADSTRING];

ATOM MyRegisterClass(HINSTANCE hInstance);
BOOL InitInstance(HINSTANCE, int);
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
INT_PTR CALLBACK About(HWND, UINT, WPARAM, LPARAM);

int APIENTRY _tWinMain(HINSTANCE hInstance,
                      HINSTANCE hPrevInstance,
                      LPTSTR lpCmdLine,
                      int nCmdShow)
{
    UNREFERENCED_PARAMETER(hPrevInstance);
    UNREFERENCED_PARAMETER(lpCmdLine);

    MSG msg;
    HACCEL hAccelTable;
    LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
    LoadString(hInstance, IDC_STANDARD, szWindowClass, MAX_LOADSTRING);
    MyRegisterClass(hInstance);
    if (!InitInstance (hInstance, nCmdShow))
    {
        return FALSE;
    }
    hAccelTable=LoadAccelerators(hInstance,MAKEINTRESOURCE(IDC_STANDARD));
    while (GetMessage(&msg, NULL, 0, 0))
    {
        if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
        {
            TranslateMessage (&msg);
            DispatchMessage (&msg);
        }
    }
    return (int)msg.wParam;
}

ATOM MyRegisterClass(HINSTANCE hInstance)
{
    WNDCLASSEX wcex;
    wcex.cbSize = sizeof(WNDCLASSEX);
    wcex.style = CS_HREDRAW | CS_VREDRAW;
```