Основы разработки приложений Windows

Простейшая программа с главным окном

группа операторов препроцессора

раздел прототипов используемых в программе прикладных функций

главная функции WinMain()

оконная функция главного окна

1. Группа операторов препроцессора

#include <windows.h>

#include <windowsx.h>

– заголовочный файл **WINDOWS.H** обеспечивает понимание компилятором смысла многочисленных типов данных Windows и подключение этого файла к исходному тексту программы является обязательным.

– Часть определений, используемых в программах содержится в файле **WINDOWSX.H**, который также необходимо включать практически во все приложения Windows.

например,

макрос GetStockBrush() и другие с ним сходные

макрос HANDLE_MSG

2. Прототипы и шаблоны функций

- Вслед за операторами препроцессора в нашем примере идет раздел прототипов, где определяется прототип единственной в данной программе функции пользователя **WndProc()**.

- В программе обязательно должны быть указаны прототипы всех используемых функций:

  - прикладных
  - системных

Вызовов системных функций Windows у нас довольно много:

- **RegisterClass(),**
- **CreateWindow(),**
- **GetMessage()** и др.

Однако прототипы всех этих функций уже определены в заголовочных файлах системы программирования.

Прототип функции **WinMain()** описан в файле **WINBASE.H**:

Прототипы остальных использованных в программе функций Windows определены в файле **WINUSER.H**.

Таким образом, о прототипах функций Windows можно не заботиться, кроме оконной функции

Оконная функция WndProc()

- Это прикладная функция, ее имя может быть каким угодно, и системе программирования это имя неизвестно.
- При наличии в приложении нескольких окон (а практически всегда так и бывает) в программе описывается несколько оконных функций, по одной для каждого класса окон.
- Для всех используемых в программе оконных функций необходимо указывать их прототипы.

формат оконной функции

- количество входных параметров функции
- типы входных параметров функции
- тип возвращаемого ею значения

– Параметры определены системой Windows и не могут быть произвольно изменены.
– Оконная функция вызывается из Windows при поступлении в приложение сообщения.
– При ее вызове Windows передает ей вполне определенный список параметров, и функция должна иметь возможность эти параметры принять и работать с ними.

формат оконной функции

В интерактивном справочнике системы программирования дается **шаблон (template)** оконной функции, который по своему виду очень **похож на прототип**, однако является не прототипом конкретной функции, а заготовкой для прикладного программиста:

```
LRESULT CALLBACK WindowProc(
    HWND      hwnd,              //Дескриптор окна
    UINT  uMsg,              //Код сообщения
    WPARAM wParam,           //Первый параметр сообщения
    LPARAM   lParam           //Второй параметр сообщения
    ) ;
```

Наша оконная функция

/*Прототип используемой в программе функции пользователя*/

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

Оконная функция имея другое имя, в точности соответствует приведенному выше шаблону:

- принимает 4 параметра указанных типов и
- возвращает (в Windows) результат типа **LRESULT.**
- Кроме того, она объявлена с описателем **CALLBACK**.

Что обозначает этот описатель?

Что обозначает описатель CALLBACK?

- В файле **WINDEF.H** символическое обозначение **CALLBACK** объявляется равнозначным ключевому слову языка C++ **_stdcall**, которое определяет правила взаимодействия функций с вызывающими процедурами.
- Win32 практически для всех функций действует так называемое соглашение стандартного вызова.
- Это соглашение определяет, что при вызове функции ее

параметры помещаются в стек в таком порядке, что в глубине стека оказывается последний параметр, а на вершине - первый.

- Сама функция, разумеется, знает о таком расположении ее параметров и выбирает их из стека в правильном порядке.

- Для 16-разрядных функций Windows действует соглашение языка Паскаль, при котором порядок помещения параметров в стек обратный.

3. Главная функция WinMain()

– Главная функция приложения **WinMain()** начинается в нашем примере с определения переменных, которые будут использоваться в программе.
– В программах на языке C++ объявления переменных могут быть разбросаны по телу программы (хотя в загрузочном модуле они будут собраны вместе).

3.1. В программе описаны 4 переменные:

char szClassName[]="MainWindow"; //Произвольное имя класса главного окна

char szTitle[]="Программа MainWindow"; //Произвольный заголовок окна

MSG Msg; //Структура Msg типа MSG для получения сообщений Windows

WNDCLASS wc; //Структура wc типа WNDCLASS для задания характеристик окна

- две символьные строки с именем класса главного окна и его заголовком и
- две структурные переменные типов MSG и WNDCLASS.

Почему для строковых переменных выбраны такие странные имена?

Программы для Windows, даже относительно простые, содержат большое количество переменных, и желательно, чтобы в их именах был какой-то порядок.

Естественно стараться давать переменным содержательные имена, и, если смысл переменной передается двумя или тремя словами, лучше каждое слово начинать с прописной буквы.

С этой точки зрения достаточно разумными выглядят имена

– **Title или ClassName**.

– Однако в программах для Windows часто идут еще дальше, включая в имя каждой переменной еще и информацию о ее *типе*. Это делается с помощью так называемой венгерской нотации.

Венгерская нотация

Суть венгерской нотации заключается в том, что имя переменной или функции предваряется одной или несколькими буквами - префиксом, говорящим о типе этой переменной.

Своё название венгерская нотация получила благодаря программисту компании Майкрософт венгерского происхождения Чарльзу Симони (венг. Simonyi Károly), предложившего её ещё во времена разработки первых версий MS-DOS. Эта система стала внутренним стандартом Майкрософт.

префиксы венгерской нотации

(для 32-разрядных приложений)

Префикс	*Расшифровка*	*Значение*
b	Bool	Логическая (булева) переменная, 32 бита
c	Character	Символ, 1 байт
dw	DoubleWord	Двойное слово без знака, 32 бита
f	Function	Функция
h	Handle	Дескриптор объекта
1	Long	Длинное целое со знаком, 32 бита
lp	LongPointer	Дальний указатель, 32 бита
lpsz	LongPointer StringZero	Д. указ на симв. строку, заканч. Двоич. нулем, 32 бита
n	iNt	Целое со знаком, 32 бита
p	Pointer	Ближний указатель, 32 бита
pt	PoinT	X- и y-координаты точки, упакованные в 64 бита
rgb	RedGreenBlue	Цвет из красной, зеленой и синей сост.- 32 бита
sz	StringZero	Символьная строка, заканчивающаяся двоичным нулем
u	Uint	Целое без знака, 32 бита
w	Word	Слово без знака, 16 бита

- ***// Простейшая программа с главный окном***
- **/*0нераторы препроцессора*/**

```
#include <windows.h>                    //Два файла с определениями, макросами
#include <windowsx.h>                   //и прототипами функций Windows

/*Прототип используемой в программе функции пользователя*/
LRESULT CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);//Оконная функция

/*Главная функция WinMain*/
int WINAPI WinMain(HINSTANCE hInst,HINSTANCE,LPSTR,int)
{
   char szClassHame[]="MainWindow";             //Произвольное имя класса главного окна
   char szTitle[]="Программа MainWindow";       //Произвольный заголовок окна
   MSG Msg;                         //Структура Msg типа MSG для получения сообщений Windows
   WNDCLASS wc;                     //Структура we типа NNDCLASS для задания характеристик окна

/*Зарегистрируем класс главного окна*/
   memset(&wc,0,sizeof(wc));        //Обнуление всех членов структуры wc
   wc.lpfnWndProc=WndProc;          //OnpenenneM оконную процедуру для главного окна
   wc.hInstance=hInst;              //Дескриптор приложения
   wc.hIcon=LoadIcon(NULL,IDI_APPLICATION);     //Стандартная пиктограмма
   wc.hCursor=LoadCursor(NULL,IDC_ARROW);       //Стандартный курсор ыыши
   wc.hbrBackground=GetStockBrush(LTGRAY_BRUSH);//Светло-серый фона окна
   wc.lpszClassName=szClassName;                //Имя класса окна
   RegisterClass(&wc);              //Вызов функции Windows регистрации класса окна

/*Создадим главное окно и сделаем его видимым*/

HWND hwnd=CreateWindow(szClassName,szTitle,             //Класс и заголовок окна
   WS_OVERLAPPEDWINDOW,10,10,300,100,       //Стиль окна,координаты/размеры
   HWND_DESKTOP,NULL,hInst,NULL);           //Родитель, меню, другие параметры
ShowWindow(hwnd,SW_SHOWNORMAL);             //Вызов функции Windows показа окна

/*Организуем цикл обработки сообщении*/

while(GetMessage(&Msg,NULL,0,0))            //Цикл обработки сообщений:
            DispatchMessage(&Msg);          //получить сообщение, вызвать WndProc
return 0;                                   //После выхода из цикла вернуться в Windows
}                                   //Конец функции WinMain

/*0конная функция WndProc главного окна*/

LRESULT CALLBACK WndProc(HWND hwnd, UINT msg,WPARAM wParam,LPARAM lParam)
{
   switch (msg)
   {                                // Переход ло зна чению msg - номеру сообщения
            case WM_DESTROY:        //При завершении приложения пользователем
                  PostQuitMessage(0);  //Вызов функции Windows завершения приложение
                  return 0;    //Возврат в Windows
default:                            //В случае всех остальных сообщений Windows обработка
   return(DefWindowProc(hwnd,msg,wParam,lParam));    //их по умолчанию
   }                                //Конец оператора switch
}                                   //Конец функции MndProc
```

3.2. Параметры функции WinMain()

- Запуская приложение (из среды разработки или с помощью кнопки "Пуск"), мы фактически передаем управление программам Windows, которые загружают в память нашу программу и вызывают ее главную функцию, которая должна иметь имя WinMain и описатель WINAPI.
- Напомним, что прототип функции WinMain() описан в файле WINBASE.H следующим образом:

```
int WINAPI WinMain (
    HINSTANCE       hInstarice,     //Дескриптор экземпляра приложения
          //Дескриптор текущего экземпляра                    //приложения
    HINSTANCE       hPrevInstance,      //В Win32 Не используется.
                    //Дескриптор предыдущего экземпляра
    //приложени
    LPSTR           lpszCmdLine,//Адрес параметров командной строки .
      //Указатель на параметры командной                      //строки
    int             nCmdShow); //Режим запуска Константа,
    //характеризующая начальный вид окна
```

3.2. Параметры функции WinMain()

Вызывая функцию WinMain(), Windows передает ей 4 параметра .

int WINAPI WinMain(HINSTANCE hInst, HINSTANCE, LPSTR, int)

- Легко увидеть, что заголовок функции WinMain() в нашей программе в *точности* соответствует приведенному выше прототипу. Иначе и быть не может. Достаточно изменить хотя бы немного характеристики нашей функции WinMain(), как программа либо не пройдет этапа компиляции, либо не будет загружаться для выполнения.

Первый параметр типа HINSTANCE

Первый параметр типа HINSTANCE представляет собой дескриптор данного экземпляра приложения, поступающий у нас в программе в локальную переменную, которой для краткости дано имя **hInst.**

- Он назначается приложению при его запуске системой Windows и служит для его идентификации.
- Многие функции Windows используют этот дескриптор в качестве входного параметра, поэтому в дальнейшем мы будем сохранять его в глобальной переменной, чтобы сделать доступным всем функциям приложения.
- В данной программе сохранение hInst не предусмотрено, так как в программе нет никаких функций, кроме главной.

Второй параметр

- **Второй параметр** того же типа, который в документации назван hPrevInstance,

- в 16-разрядных приложениях являлся дескриптором предыдущего экземпляра этого же приложения и принимал ненулевое значение, если приложение запускалось в нескольких экземплярах.
- Анализируя значение параметра hPrevInstance, можно было определить, является ли данный экземпляр приложения первым.

- В Win32 этот параметр всегда равен нулю и не имеет смысла. Соответственно нет необходимости предусматривать для него локальную переменную, поэтому в заголовке функции WinMain() указан только тип второго параметра (HINSTANCE), но нет обозначения переменной.

Третий параметр lpszCmdLine

- **Третий параметр,** lpszCmdLine, представляет собой указатель на строку, содержащую параметры командной строки запуска приложения, если при его запуске были указаны параметры.
- Поскольку мы не предполагаем запускать наше приложение с параметрами и этот аргумент функции WinMain() нам не нужен, в заголовке функции указан только его тип **LPSTR**, а сам аргумент опущен.

Четвертый параметр nCmdShow

- **Четвертый параметр,** nCmdShow, характеризует режим запуска.

- Режим запуска любого приложения можно установить, если, создав ярлык для приложения, открыть для него окно "Свойства", перейти на вкладку "Ярлык" и раскрыть список "Окно".
- Если, далее, выбрать в этом списке пункт **"Свернутое в значок",** Windows, запуская приложение, будет свертывать его в пиктограмму. В этом случае из Windows в WinMain() поступает значение параметра nCmdShow, равное символической константе SW_SHOWMINNOACTIVE=7.
- Если же включен режим **Стандартный размер**, окно приложения на экране развертывается до размера, заданного в самом приложении, а в WinMain() поступает константа SW_SHOWNORMAL=1. Полученную через параметр nCmdShow константу можно затем использовать в качестве параметра при вызове функции Windows ShowWindow().
- Мы поступили проще, указав при вызове ShowWindow() явным образом константу **SW_SHOWNORMAL**.
- Разумеется, внутренние, действующие в программе имена для параметров функции WinMain, как и для любой другой функции, пользователь может выбирать по своему усмотрению.

3.3. Состав функции WinMain()

В типичном приложении Windows главная функция WinMain() должна выполнить по меньшей мере 3 важные процедуры:

- **Зарегистрировать в системе Windows класс главного окна**. Если помимо главного окна планируется выводить на экран внутренние, порожденные окна, то их классы также необходимо зарегистрировать. Windows выводит на экран и обслуживает только зарегистрированные окна.
- **Создать главное окно и показать его на экране**. Порожденные окна также необходимо создать, хотя это можно сделать и позже, и не обязательно в функции WinMain().
- **Организовать цикл обработки сообщений, поступающих в приложение**.
  - Вся дальнейшая жизнь приложения будет фактически состоять в бесконечном выполнении этого цикла и в обработке поступающих в приложение сообщений.
  - Запущенное приложение Windows обычно функционирует до тех пор, пока пользователь не подаст команду его завершения с помощью системного меню или вводом команды Alt+F4.
  - Эти действия приводят к завершению главной функции и удалению приложения из списка действующих задач.

3.3.1. Класс окна и его характеристики

Для вывода на экран любого окна, в частности главного окна приложения, необходимо прежде всего зарегистрировать *класс окна*, в котором задаются наиболее общие характеристики всех окон данного класса.

Это действие выполняется в первой, инициализирующей части функции WinMain().

Действия по регистрации класса окна заключаются

- в заполнении структуры типа WNDCLASS, служащей для описания характеристик класса регистрируемого окна,
- вызове затем функции Windows **RegisterClass(),** которая и выполняет регистрацию данного класса.

Структура **WNDCLASS**, определенная в файле **WINUSER.H**, содержит ряд членов, задающих наиболее общие характеристики окна:

Структура **WNDCLASS**

```
typedef   struct  tagWNDCLASS (
    UINT        style;          //Стиль класса окна
    WNDPROC     lpfnWndProc;    //имя оконной функции
    int         cbClsExtra;     //Число байтов дополнительной информации о классе
    int         cbWndExtra;     //Число байтов дополнительной информации об окне
    HINSTANCE   hInstance;      //Дескриптор приложения
    HICON       hIcon;          //Дескриптор пиктограммы приложения
    HCURSOR     hCursor;        //Дескриптор курсора приложения
    HBRUSH      hbrBackground;  //Дескриптор кисти для фона окна
    LPCSTR      lpszMenuName;   //Указатель на строку с именем меню окна
    LPCSTR      lpszClassName;  //Указатель на строку с именем класса окна
    )WNDCLASS;                  //Новое имя типа данной структуры
```

В большинстве случаев нет необходимости определять все члены этой структуры;

- для упрощения дела мы сначала обнуляем всю структуру вызовом функции C++ **memset()**
  memset(&wc, 0, sizeof(wc));
  которая записывает, начиная с адреса &wc, нули в количестве sizeof(wc) штук,
- затем задаем значения только интересующим нас членам.

- При заполнении структурной переменной типа WNDCLASS необходимо обеспечить нулевое значение всех элементов, которым мы *не присваиваем* конкретные значения.
- Нулевое значение элемента структуры обозначает для Windows, что характеристики этого элемента должны устанавливаться по умолчанию. Это правило, кстати, характерно и для других структур, описывающих те или иные объекты Windows,

- например
  - для структуры OPENFILENAME, служащей для вывода на экран стандартного диалога Windows "Открытие файла",
  - или структуры LOGFONT, позволяющей создать шрифт требуемого начертания.

Предварительное обнуление всей структурной переменной

- Предварительное обнуление всей структурной переменной перед ее инициализацией можно осуществить с помощью "классической" функции C++ **memset(),** как это сделано в нашем примере,
- можно использовать функцию Win32 **ZeroMemory():**
  ZeroMemory(&wc,sizeof(wc)) ;

которая делает то же самое, но, возможно, несколько быстрее.

- Можно, наконец, воспользоваться тем обстоятельством, что глобальные переменные обнуляются автоматически.
  - Если переменную **wc** расположить *перед* функцией WinMain(), то при загрузке программы в память она будет заполнена нулями и в вызове функций memset(), или ZeroMemory() не будет необходимости.

Какое-то средство обнуления использовать необходимо, так как иначе структура при ее создании в памяти будет заполнена "мусором", что при запуске программы скорее всего приведет к драматическим последствиям.

Поля структуры

- Наиболее важными для дальнейшего функционирования программы являются три поля:
  - hInstance, где хранится дескриптор данного приложения,
  - lpszClassName, куда заносится произвольное имя, присвоенное нами окнам данного класса (у нас это окно единственное) и
  - lpfnWndProc — имя оконной функции.
- Именно с помощью структуры WNDCLASS Windows, обслуживая нашу программу, определяет адрес оконной функции, которую она должна вызывать при поступлении в окно сообщений.
- Поле lpszClassName мы заполняем из переменной szClassName, определенной нами в начале программы,
- значение дескриптора приложения переносим из параметра hInst

Поля структуры

- Менее важными в принципиальном плане, но существенными для разумного поведения приложения являются поля
  - hIcon,
  - hCursor и
  - hbrBackground,
- куда следует записать дескрипторы пиктограммы, курсора и цвета фона окна приложения (точнее, цвета кисти, которая используется для закраски фоновой области окна).

Курсор

- Курсор относится к *ресурсам* Windows;
- ресурсы обычно загружаются из специально созданного файла ресурсов с помощью соответствующих функций Windows, в частности LoadCursor().
  - В качестве первого аргумента этих функций указывается дескриптор приложения, в котором хранится требуемый ресурс,
  - а в качестве второго - имя ресурса.
- Однако можно обойтись и встроенными ресурсами Windows.
- Для этого в качестве
  - первого аргумента этих функций указывается NULL (обычно NULL на месте дескриптора приложения обозначает саму систему Windows),

Встроенные курсоры

Имя курсора	*Вид курсора*
IDC_ARROW	
IDC_CROSS	
IDC_SIZE	
IDC_IBEAM	
IDC_SIZENS	
IDC_WAIT	

- Очевидно, что для главного окна приложения целесообразно выбирать стандартный курсор IDC_ARROW.
- Остальные курсоры используются в специальных случаях,
- как правило, загружаясь динамически лишь в определенных ситуациях и на некоторое время.

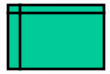
Например,

- курсор в форме песочных часов традиционно говорит о том, что в системе протекает какой-то процесс, не позволяющий приложению работать обычным образом, - загрузка файла, вычисления и пр.

пиктограмма

- Другим ресурсом Windows, указываемым в классе окна, является пиктограмма.
- Как и курсор, пиктограмма может быть разработана программистом специально для данного приложения и храниться в файле ресурсов приложения или в отдельном файле с расширением .ICO;
- для учебных задач проще использовать одну из стандартных пиктограмм Windows.

Встроенные пиктограммы

Имя пиктограммы	*Вид пиктограммы*
IDI_APPLICATION	
IDI_HAND	
IDI_ASTERISK	
IDI_QUESTION	

- Для того чтобы придать приложению требуемую пиктограмму, надо при заполнении структурной переменной **wc** типа WNDCLASS загрузить дескриптор пиктограммы в элемент **wc.hIcon**.
- Для получения дескриптора следует воспользоваться функцией LoadIcon();

Кисти

- Цвет фона окна определяется дескриптором кисти, записанным в структуру WNDCLASS.
- В принципе кисти можно придать любой цвет и даже фактуру, но проще всего воспользоваться одной из встроенных кистей, хранящихся "на складе" Windows.
- Для получения встроенной кисти следует воспользоваться макросом GetStockBrush(), в качестве единственного аргумента которого указывается константа, характеризующая цвет кисти.

Предопределенные кисти Windows

Имя кисти	*Цвет*
BLACK_BRUSH	Черный
LTGRAY _BRUSH	Светло-серый
DKGRAY _BRUSH	Темно-серый
NULL_ BRUSH	Прозрачный
GRAY_BRUSH	Серый
WHITE _BRUSH	Белый

Кисти произвольного цвета

- Таких кистей нет "на складе", и их придется создавать заново.
- Создание кисти - функция **CreateSolidBrush(),**
  - в качестве единственного параметра которой указывается требуемый цвет в виде трех чисел, характеризующих интенсивность красной, зеленой и синей составляющих цвета.
  - Для упаковки этих трех чисел в одно 4-байтовое слово служит макрос RGB().

- Пример: создать бирюзовый фон

wc.hbrBackgrouhd=CreateSolidBrush(RGB(0,255,255));

стиль класса окна

Стиль представляет собой целое число (32 бита), отдельные биты которого закреплены за теми или иными общими характеристиками всех окон, принадлежащих регистрируемому классу; каждому биту соответствует своя символическая константа.

- 0 бит (0x00000001, константа CS_VREDRAW)
- 1 бит (0x00000002, константа CS_HREDRAW) заставляет Windows перерисовывать окно заново при каждом изменении его размеров по горизонтали и вертикали;
- 3 бит (0x00000008, константа CS_DBLCLKS) позволяет программе реагировать на двойные щелчки мышью в области окна;
  …….
- 9 бит (0x00000200, константа CS_NOCLOSE) запрещает закрытие окна пользователем и т. д.

Функция RegisterClass()

- Функция RegisterClass()
  - аргумент функции: *адрес* структурной переменной типа WNDCLASS
  - вызов функции : **RegisterClass(&wc) ;**

- **Пример:**
- **memset(&wc,0,sizeof(wc));** //Обнуление всех членов структуры wc
  wc.lpfnWndProc=WndProc; //Onределим оконную процедуру для гл. окна
  wc.hInstance=hInst; //Дескриптор приложения
  wc.hIcon=LoadIcon(NULL,IDI_APPLICATION); //Стандартная пиктограмма
  wc.hCursor=LoadCursor(NULL,IDC_ARROW); //Стандартный курсор мыши
  wc.hbrBackground=GetStockBrush(LTGRAY_BRUSH);//Светло-серый фона окна
  wc.lpszClassName=szClassName; //Имя класса окна
  RegisterClass(&wc); //регистрация класса окна

- Зарегистрировав класс окна, можно приступить
  - к созданию
  - и показу окна.

3.3.2. Создание и показ окна

- Для создания окна,, используется функция Windows CreateWindow()

Прототип функции:

```
HWND CreateWindow(
    LPCSTR  lpClassName,//Адрес строки с именем класса
    LPCSTR. lpWindowName,   //Адрес строки с заголовком окна
    DWORD  dwStyle,        //Стиль окна
    int          x,          //Гориз. позиция окна на Рабочем столе
    int          y,          //Вертик. позиция окна на Рабочем столе
    int          nWidth,        //Ширина окна
    int          nHeight,       //Высота окна
    HWND    hWndParent,//Дескриптор родительского окна
    HMENU   hMenu,         //Дескриптор меню
    HINSTANCE   hInstance,     //Дескриптор приложения
    LPVOID   lpParam       //Адрес дополнительных данных
    );
```

- Существенными элементами прототипа являются
  - порядок
  - типы параметров
- Указание в справочнике имен параметров следует воспринимать как необязательную рекомендацию.

Параметр lpClassName

- **Параметр lpClassName** - адрес строки с именем регистрируемого класса окна.
- Передача функции CreateWindow() имени класса позволяет использовать при создании окна те общие характеристики окна, которые были определены в структуре WNDCLASS, например
  - форму курсора
  - цвет фона.
- Вызывая функцию CreateWindow() многократно, можно создать много окон данного класса, различающихся, например,
  - размерами
  - местоположением на экране.
- Однако пока мы создаем главное окно приложения, которое, очевидно, должно быть в одном экземпляре.

Параметр lpWindowName

- **Параметр lpWindowName** определяет адрес строки с заголовком, который появится в верхней части окна.

Параметр dwStyle

- **Параметр dwStyle** определяет стиль окна
  - вид окружающей его рамки,
  - наличие или отсутствие строки заголовка
  - ….
- Стиль представляет собой целое число (32 бита)
  - отдельные биты закреплены за элементами оформления или свойствами окна
  - каждому биту соответствует своя символическая константа
- Например:
  - 18 бит (0x00040000, константа WS_THICKFRAME) придает окну толстую рамку;
  - 19 бит (0x00080000, WS_SYSMENU) снабжает окно системным меню;
  - 21 бит (0x00200000, WS_VSCROLL) отвечает за появление линейки вертикальной прокрутки
  - ……..

Параметр dwStyle

- Обычно главное окно описывается константой **WS_OVERLAPPEDWINDOW** (OxOOCFOOOO),

- в константу входят элементы стиля

  – WS_OVERLAPPED (перекрывающееся окно),
  – WS_CAPTION (строка заголовка),
  – WS_SYSMENU (системное меню),
  – WS_THICKFRAME (толстая рамка),
  – WS_MINIMIZEBOX (кнопка минимизации, т. е. свертывания окна в пиктограмму) и
  – WS_MAXIMIZEBOX (кнопка максимизации, т. е. развертывания окна на весь экран).

Параметр dwStyle

- Операция побитового ИЛИ (знак |) позволяет "набрать" требуемый комплект свойств.
  - весь необходимый набор можно указать явным образом.

  Пример:
  - для создания главного окна без кнопки максимизации константу стиля надо задать следующим образом:

  WS_THICKFRAME | WS_SYSMENU | WS_MINIMIZEBOX
  (строка заголовка появляется в главном окне в любом случае, а константа WS_OVERLAPPED равна нулю, и ее указание не влияет на стиль).

- С другой стороны, воспользовавшись операторами побитовых преобразований И (&) и НЕ (~), можно не набирать заново всю комбинацию констант, а просто исключить ненужный элемент WS_MAXIMIZEBOX из полного набора:

  - **WS_OVERLAPPEDWINDOW & ~WS_MAXIMIZEBOX**

Координаты окна

- **4 параметра** определяют x- и y-координаты левого верхнего угла окна относительно начала экрана и размеры окна по горизонтали и вертикали (в пикселах).

Параметр hWndParent

- **В качестве параметра hWndParent** указывается дескриптор родительского окна.
- Для главного окна, у которого нет родителя, используется константа HWND_DESKTOP.

Параметр hMenu

- **Параметр hMenu** позволяет задать меню окна.
- Если меню нет (как в нашем случае) или используется меню класса, заданное в структуре WNDCLASS, этот параметр должен быть равен нулю (NULL).

Параметр hInstance

- **Параметр hInstance** идентифицирует экземпляр приложения.
- Значение дескриптора приложения было получено нами через аргумент hInst функции WinMain().

Параметр lpParam

- **Параметр lpParam** является адресом дополнительных данных, которые обычно не требуются; соответственно мы указали "пустой" адрес (NULL).
  - Символическое обозначение NULL (нуль-указатель) используется в тех случаях, когда для некоторой функции надо указать нулевое значение параметра, являющегося *указателем.*
  - В Borland C++ для Win32 NULL определяется как 0, так что формально во всех случаях вместо NULL можно использовать просто 0; однако обозначение NULL напоминает, что данный аргумент представляет собой адрес.

Функция **CreateWindow()**

- Функция **CreateWindow()** возвращает (при успешном выполнении) дескриптор созданного окна.
- Этот дескриптор (локальная переменная hwnd) передается в функцию **ShowWindow(),** которая организует вывод созданного окна на экран.
- /*Создадим главное окно и сделаем его видимым*/

HWND hwnd=CreateWindow(szClassName,szTitle,//Класс и заголовок окна
WS_OVERLAPPEDWINDOW,10,10,300,100,//Стиль окна, координаты /размеры
HWND_DESKTOP,NULL,hInst,NULL); //Родитель, меню, другие параметры

ShowWindow(hwnd,SW_SHOWNORMAL); //Вызов функции Windows показа окна

Теперь для правильного функционирования приложения необходимо организовать цикл обработки сообщений.

3.3.3. Цикл обработки сообщений

- **Назначение цикла обработки сообщений -**
  - получение сообщений, поступающих в приложение
  - вызов в ответ на каждое сообщение оконной процедуры для обработки этого сообщения.
- В простейшем виде *цикл обработки сообщений* состоит из одного предложения языка

```
while(GetMessage(&Msg,NULL,0,0))       //Цикл обработки сообщений:
    DispatchMessage(&Msg);             //получить сообщение, вызвать WndProc
```

В этом бесконечном (если его не разорвать изнутри) цикле:

- вызывается функция Windows GetMessage(),
- если она возвращает ненулевое значение, вызывается функция DispatchMessage().

- Если сообщения в приложение не поступают, цикл обработки сообщений "крутится" вхолостую.
- Как только в очереди обнаруживается сообщение,
  - функция GetMessage() забирает его из очереди и
  - переносит в структурную переменную, предназначенную для приема сообщений (Msg).
  - функция **DispatchMessage()** вызывает **оконную процедуру** того окна, которому предназначено данное сообщение, и передает ей (через ее параметры) содержимое сообщения из структуры Msg.

- **Задача оконной процедуры** - выполнить требуемую обработку поступившего сообщения.

- Когда оконная процедура завершит обработку полученного сообщения, управление вернется в цикл обработки сообщений, который продолжит свое циклическое выполнение в ожидании нового сообщения.

- Для того чтобы разобраться в деталях этого очень важного механизма, нам надо познакомиться поближе с самим понятием сообщений.