* 1. **«Новые» типы данных**

Каждый тип данных является синонимом уже существующего типа языка С++. Например:

|  |
| --- |
| typedef int BOOL |

Таблица типов данных Windows:

| **Тип данных** | **Описание** |
| --- | --- |
| BOOL | Булевский тип данных. Может принимать одно из двух значений TRUE или FALSE. Занимает 4 байта. |
| BYTE | 1-байтное целое без знака. |
| COLORREF | Тип данных, используемый для работы с цветом. Занимает 4 байта. |
| DWORD | 4-х байтное целое или адрес. |
| HANDLE | 4-х байтное целое, используемое в качестве дескриптора. |
| HBITMAP | Дескриптор растрового изображения. |
| HBRUSH | Дескриптор кисти. |
| HCURSOR | Дескриптор курсора. |
| HDC | Дескриптор устройства. |
| HFONT | Дескриптор шрифта. |
| HICON | Дескриптор иконки. |
| HINSTANCE | Дескриптор экземпляра приложения. |
| HMENU | Дескриптор меню. |
| HWND | Дескриптор окна. |
| INT | 4-х байтное целое со знаком. |
| LONG | 4-х байтное целое со знаком. |
| LPARAM | Переменные этого типа передаются в качестве дополнительного параметра в функцию - обработчик какого-либо сообщения. В них обычно содержатся информация специфическая для данного события. Занимает 4 байта. |
| LPCSTR | 4-х байтный указатель на константную строку символов. Указатели с приставкой LP обычно называют длинными указателями. |
| LPCWSTR | 4-х байтный указатель на константную Unicode-строку. |
| LPSTR | 4-х байтный указатель строку символов. |
| LPWSTR | 4-х байтный указатель на Unicode-строку. |
| LRESULT | Значение типа LONG, возвращаемое оконной процедурой |
| UINT | 4-х байтное целое без знака. |
| WORD | 2-х байтное целое без знака. |
| WPARAM | Переменные этого типа передаются в качестве дополнительного параметра в функцию - обработчик какого-либо сообщения. В них обычно содержатся информация специфическая для данного события. Занимает 4 байта. |

**4. Венгерская нотация**

Было бы удобно по имени переменной определить её назначение в программе, а также тип данных. Для решения этой проблемы программисты Microsoft предложили так называемую **венгерскую нотацию.** Она названа так потому, что ее в Microsoft популяризировал венгерский программист Чарльз Шимоньи (Charles Simonyi). В венгерской нотации переменным даются описательные имена, начинающиеся с заглавных букв. Например, Counter, Flag, BookTitle, AuthorName. Если имя состоит из нескольких слов, каждое слово начинается с заглавной буквы. Затем перед описательным именем добавляются буквы, чтобы указать тип переменной. Например, uCounter для типа unsigned int и bFlag для типа bool, szBookTitle для символьного массива (sz – string zero).

Большинство функций WinAPI используют венгерскую нотацию, поэтому, по меньшей мере, знать о ней необходимо. В венгерской нотации предлагаются следующие префиксы, приведя следующую таблицу:

| **Префикс** | **Тип переменной** |
| --- | --- |
| b | Логический тип (bool или BOOL) |
| i | Целое число (индекс) |
| n | Целое число (количество чего-либо) |
| u | Целое число без знака |
| d | Число с двойной точностью |
| sz | Строковая переменная, ограниченная нулем |
| p | Указатель |
| lp | Длинный указатель |
| a | Массив |
| lpfn | Длинный указатель на функцию |
| h | Дескриптор |
| cb | Счетчик байтов |
| C | Класс |

1. **Минимальное Win32-приложение**

Минимальное Win32-приложение должно содержать как минимум две функции:

* **WinMain** — главную функцию, в которой создается основное окно программы и запускается цикл обработки сообщений;
* **WndProc** — оконную процедуру, обеспечивающую обработку сообщений для основного окна программы.

WinMain является точкой входа в программу и выполняет следующие действия:

* определение класса окна;
* регистрация класса окна;
* создание окна;
* отображение окна;
* запуск цикла обработки сообщений.

|  |
| --- |
| // Файл WINDOWS.H содержит определения, макросы, и структуры  // которые используются при написании приложений под Windows.  #include <windows.h>  #include <tchar.h>  LRESULT CALLBACK WindowProc(HWND, UINT, WPARAM, LPARAM);  TCHAR szClassWindow[] = TEXT("Каркасное приложение");  INT WINAPI WinMain(HINSTANCE hInst, HINSTANCE hPrevInst, LPSTR lpszCmdLine, int nCmdShow)  {  HWND hWnd;  MSG lpMsg;  WNDCLASSEX wcl;  wcl.cbSize = sizeof (wcl);  wcl.style = CS\_HREDRAW | CS\_VREDRAW;  wcl.lpfnWndProc = WindowProc;  wcl.cbClsExtra = 0;  wcl.cbWndExtra = 0;  wcl.hInstance = hInst;  wcl.hIcon = LoadIcon(NULL, IDI\_APPLICATION);  wcl.hCursor = LoadCursor(NULL, IDC\_ARROW);  wcl.hbrBackground = (HBRUSH) GetStockObject(WHITE\_BRUSH);  wcl.lpszMenuName = NULL;  wcl.lpszClassName = szClassWindow;  wcl.hIconSm = NULL;  if (!RegisterClassEx(&wcl))  return 0;    hWnd=CreateWindowEx(  0,  szClassWindow,  TEXT("Каркас Windows приложения"),  WS\_OVERLAPPEDWINDOW,  CW\_USEDEFAULT,  CW\_USEDEFAULT,  CW\_USEDEFAULT,  CW\_USEDEFAULT,  NULL,  NULL,  hInst,  NULL);  ShowWindow(hWnd, nCmdShow);  UpdateWindow(hWnd);    while(GetMessage(&lpMsg, NULL, 0, 0))  {  TranslateMessage(&lpMsg);  DispatchMessage(&lpMsg);  }  return lpMsg.wParam;  }  LRESULT CALLBACK WindowProc(HWND hWnd, UINT uMessage, WPARAM wParam,  LPARAM lParam)  {  switch(uMessage)  {  case WM\_DESTROY:  PostQuitMessage(0);  break;  default:  return DefWindowProc(hWnd, message, wParam, lParam);  }  return 0;  } |

Заголовочный файл **Windows.h** содержит определения, макросы, и структуры, которые используются при написании приложений под Windows. Таким образом, при программировании Windows-приложений с использованием Win API данный файл следует подключать всегда.

Как отмечалось выше, функция **WinMain** является точкой входа в приложение и имеет следующий прототип:

|  |
| --- |
| INT WINAPI WinMain(  HINSTANCE hInst, // дескриптор экземпляра приложения  HINSTANCE hPrevInst, //равен 0 и необходим для совместимости  LPSTR lpszCmdLine, // указатель на строку, в которую копируются   //аргументы приложения, если оно запущено в режиме командной строки  int nCmdShow // способ визуализации окна при запуске программы  ); |

Спецификатор **WINAPI** определяет соглашение о вызове функции.

Наиболее распространены два протокола вызова функции:

1. **\_\_ cdecl** – по данному протоколу вызывающая функция сама очищает стек после вызываемой функции. При этом передача параметров функции в стек происходит в порядке справа налево. По данному протоколу происходит вызов функции только в языке С/С++. Это связано с тем, что в языке С имеется семейство функций с произвольным количеством параметров (**printf**).
2. **\_\_ stdcall (WINAPI)** – согласно этому протоколу вызываемая функция сама за собой очищает стек. При этом передача параметров функции в стек происходит в порядке справа налево. Такой вызов используется в других языках программирования (например, Паскаль, Фортран).
   1. **6.1. Определение класса окна**

Для определения класса окна в функции **WinMain** заполняются поля структуры **WNDCLASSEX:**

|  |
| --- |
| typedef struct tagWNDCLASSEX {  UINT cbSize; // размер данной структуры в байтах  UINT style; // стиль класса окна  WNDPROC lpfnWndProc; // указатель на функцию окна (оконную процедуру)  int cbClsExtra; // число дополнительных байтов, которые должны  //быть распределены в конце структуры класса  int cbWndExtra; // число дополнительных байтов, которые должны  //быть распределены вслед за экземпляром окна  HINSTANCE hInstance; // дескриптор экземпляра приложения, в котором  //находится оконная процедура для этого класса  HICON hIcon; // дескриптор иконки  HCURSOR hCursor; // дескриптор курсора  HBRUSH hbrBackground; //дескриптор кисти, используемой для закраски фона окна  LPCTSTR lpszMenuName; // указатель на строку, содержащую имя меню,  //применяемого по умолчанию для этого класса  LPCTSTR lpszClassName; // указатель на строку, содержащую имя класса окна  HICON hIconSm; // дескриптор малой иконки  } WNDCLASSEX; |

В рассматриваемом приложении в поле **style** структуры **WNDCLASSEX** указана комбинация стилей **CS\_HREDRAW | CS\_VREDRAW**. Это означает, что окно будет перерисовано, если изменён размер по горизонтали или по вертикали.

В поле **hIcon** устанавливается дескриптор иконки, возвращаемый функцией API **LoadIcon***:*

|  |
| --- |
| HICON LoadIcon (  HINSTANCE hInst, //дескриптор экземпляра приложения, содержащего иконку  LPCSTR lpszName //строка, содержащая имя иконки  ); |

Для того чтобы использовать встроенные типы иконок Windows, первый параметр должен быть равен NULL, a в качестве второго параметра должен использоваться один из следующих макросов:

| **Макрос** | **Форма иконки** |
| --- | --- |
| IDI\_APPLICATION | Стандартная иконка для приложения |
| IDI\_ASTERISK | Иконка "информация" |
| IDI\_EXCLAMATION | Иконка "восклицательный знак" |
| IDI\_HAND | Иконка "знак Стоп" |
| IDI\_QUESTION | Иконка "вопросительный знак" |

В поле **hCursor** устанавливается дескриптор курсора, возвращаемый функцией API **LoadCursor***:*

|  |
| --- |
| HCURSOR LoadCursor (  HINSTANCE hInst, //дескриптор экземпляра приложения, содержащего курсор  LPCSTR lpszName //строка, содержащая имя курсора  ); |

Для того чтобы использовать встроенный тип курсора Windows, первый параметр должен быть равен NULL, a в качестве второго параметра должен использоваться один из следующих макросов:

| **Макрос** | **Форма иконки** |
| --- | --- |
| IDC\_ARROW | Стандартный курсор - стрелка |
| IDC\_CROSS | Перекрестье |
| IDC\_IBEAM | Текстовый двутавр |
| IDC\_WAIT | "Песочные часы" |
| IDC\_HELP | Стрелка и вопросительный знак |
| IDC\_SIZEALL | Четырехконечная стрелка |

Поле **hbrBackground** инициализируется дескриптором кисти, используемым для закраски фона окна. Кисть (brush) — это графический объект, который представляет собой шаблон пикселов различных цветов, используемый для закрашивания области. В Windows имеется несколько стандартных или предопределенных кистей. Вызов функции API **GetStockObject** с аргументом **WHITE\_BRUSH** возвращает дескриптор белой кисти. Так как возвращаемое значение имеет тип **HGDIOBJ**, то его необходимо преобразовать к типу **HBRUSH**.

|  |
| --- |
| HGDIOBJ GetStockObject(  int object //предопределённый объект GDI  *);* |

* 1. **6.2. Регистрация класса окна**

Когда класс окна полностью определен, он должен быть зарегистрирован в системе. Для этого используется функция API **RegisterClassEx,** возвращающая значение, идентифицирующее зарегистрированный класс окна:

|  |
| --- |
| ATOM RegisterClassEx(  CONST WNDCLASS \* lpWClass *//* адрес структуры WNDCLASSEX  ); |

* 1. **6.3. Создание окна. Стили окна.**

После того как класс окна определен и зарегистрирован, можно создавать окна этого класса, используя функцию API **CreateWindowEx:**

|  |
| --- |
| HWND CreateWindowEx(  DWORD dwExStyle*, //* расширенныйстиль окна  LPCSTR lpClassName*, //* имя класса окна  LPCSTR lpWinName*, //* заголовок окна  DWORD dwStyle*, //* стиль окна  int x*,* int у, // координаты верхнего левого угла  int Width*,* int Height*, //* размеры окна  HWND hParent*, //* дескриптор родительского окна  HMENU hMenu*, //* дескриптор главного меню  HINSTAHCE hThisInst*, //* дескриптор приложения  LPVOID lpszAdditional // указатель на дополнительную информацию  ); |

Первый параметр **dwExStyle** задает расширенный стиль окна, применяемый совместно со стилем, определенным в параметре **dwStyle**.

Например, в качестве расширенного стиля можно задать один или несколько флагов, приведенных в следующей таблице.

| **Стиль** | **Описание** |
| --- | --- |
| WS\_EX\_ACCEPTFILES | Создать окно, которое принимает перетаскиваемые файлы |
| WS\_EX\_CLIENTEDGE | Рамка окна имеет утопленный край |
| WS\_EX\_CONTROLPARENT | Разрешить пользователю перемещаться по дочерним окнам с помощью клавиши Tab |
| WS\_EX\_MDICHILD | Создать дочернее окно многодокументного интерфейса |
| WS\_EX\_STATICEDGE | Создать окно с трехмерной рамкой. Этот стиль предназначен для элементов, которые не принимают ввод от пользователя |
| WS\_EX\_TOOLWINDOW | Создать окно с инструментами, предназначенное для реализации плавающих панелей инструментов |
| WS\_EX\_TRANSPARENT | Создать прозрачное окно. Любые окна того же уровня, накрываемые этим окном, получат сообщение WM\_PAINT в первую очередь |
| WS\_EX\_WINDOWEDGE | Создать окно, имеющее рамку с активизированным краем |

В нашем приложении в качестве основного стиля (параметр **dwStyle**) используется макрос **WS\_OVERLAPPEDWINDOW,** который определяет стандартное окно, имеющее системное меню, заголовок, рамку для изменения размеров, а также кнопки минимизации, развертки и закрытия. Используемый стиль окна является наиболее общим. Допускается создавать окна с другими стилями, некоторые из которых приведены в следующей таблице.

| **Стиль** | **Описание** |
| --- | --- |
| WS\_OVERLAPPED | Стандартное окно с рамкой |
| WS\_MAXIMIZEBOX | Наличие кнопки развертки |
| WS\_MINIMIZEBOX | Наличие кнопки минимизации |
| WS\_SYSMENU | Наличие системного меню |
| WS\_HSCROLL | Наличие горизонтальной панели прокрутки |
| WS\_VSCROLL | Наличие вертикальной панели прокрутки |

В нашем приложении для параметров **x, у, Width** и **Height** функции **CreateWindowEx** используется макрос **CW\_USEDEFAULT**, что позволяет системе самостоятельно выбирать координаты и размеры окна. Если окно не имеет родительского окна, как в случае нашего приложения, то параметр **hParent** должен быть равен **HWND\_DESKTOP** (или **NULL**, - это тоже допускается).

* 1. **6.4. Отображение окна**

Для отображения на экране созданного окна вызывается функция **ShowWindow**, имеющая следующий прототип:

|  |
| --- |
| BOOL ShowWindow(  HWND hWnd, //дескриптор окна  int nCmdShow //способ отображения окна  ); |

При начальном отображении главного окна рекомендуется присваивать второму параметру то значение, которое передается приложению через параметр **nCmdShow** функции **WinMain**. При последующих отображениях можно использовать любое из значений, приведенных в следующей таблице.

| **Макрос** | **Эффект** |
| --- | --- |
| SW\_HIDE | Скрыть окно |
| SW\_MAXIMIZE | Развернуть окно |
| SW\_MINIMIZE | Свернуть окно |
| SW\_SHOW | Активизировать окно и показать в его текущих размерах и позиции |
| SW\_RESTORE | Отобразить окно в нормальном представлении |

Рекомендуется после вызова функции **ShowWindow** вызвать функцию **UpdateWindow**, которая посылает оконной процедуре сообщение **WM\_PAINT**, заставляющее окно перерисовать свою клиентскую область.

|  |
| --- |
| BOOL UpdateWindow( HWND hWnd ); |

* 1. **6.5. Цикл обработки сообщений**

Последней частью функции **WinMain** является **цикл обработки сообщений***.* Его целью является получение и обработка сообщений, передаваемых операционной системой. Эти сообщения ставятся в очередь сообщений приложения, откуда они затем (по мере готовности программы) выбираются функцией API **GetMessage:**

|  |
| --- |
| BOOL GetMessage(  LPMSG msg, //адрес структуры MSG, в которую помещается выбранное сообщение  HWND hwnd*,* // дескриптор окна, принимающего сообщение  /\* Обычно значение этого параметра равно NULL, что позволяет выбрать сообщения для любого окна приложения. \*/  UINT min, // минимальный номер принимаемого сообщения  UINT max // максимальный номер принимаемого сообщения  /\* Если оба последних параметра равны нулю, то функция выбирает из очереди любое очередное сообщение. \*/  ); |
| typedef struct tagMSG {  HWND hwnd; - дескриптор окна, которому адресовано сообщение  UINT message; - идентификатор сообщения  WPARAM wParam; - дополнительная информация  LPARAM lParam; - дополнительная информация  DWORD time; - время отправки сообщения  POINT pt; - экранные координаты курсора мыши в момент отправки сообщения  } MSG; |
| typedef struct tagPOINT {  LONG x; //координата X точки  LONG y; //координата Y точки  } POINT, \*PPOINT; |

Функция **GetMessage** возвращает значение **TRUE** при извлечении любого сообщения, кроме одного — **WM\_QUIT**. Получив сообщение **WM\_QUIT**, функция возвращает значение **FALSE**. В результате этого происходит немедленный выход из цикла, и приложение завершает работу, возвращая операционной системе код возврата **msg.wParam.**

Вызов **TranslateMessage** нужен только в тех приложениях, которые должны обрабатывать ввод данных с клавиатуры. Дело в том, что для обеспечения независимости от аппаратных платформ и различных национальных раскладок клавиатуры в Windows реализована двухуровневая схема обработки сообщений от символьных клавиш. Сначала система генерирует сообщения о так называемых виртуальных клавишах, например: сообщение **WM\_KEYDOWN** — когда клавиша нажимается, и сообщение **WM\_KEYUP** — когда клавиша отпускается. В сообщении **WM\_KEYDOWN** содержится также информация о так называемом скан-коде нажатой клавиши.

Функция API **TranslateMessage** преобразует пару аппаратных сообщений, **WM\_KEYDOWN** и **WM\_KEYUP**, в символьное сообщение **WM\_CHAR**, которое содержит ASCII-код символа (wParam). Сообщение **WM\_CHAR** помещается в очередь, а на следующей итерации цикла функция **GetMessage** извлекает его для последующей обработки.

Функция API **DispatchMessage** передает структуру MSG обратно в Windows. Windows отправляет сообщение для его обработки соответствующей оконной процедуре, вызывая ее как **функцию обратного вызова.**

* 1. **6.6. Оконная процедура**

Оконная процедура вызывается операционной системой и получает в качестве параметров сообщения из очереди сообщений данного приложения.

|  |
| --- |
| LRESULT CALLBACK WindowProc(HWND hWnd, UINT uMessage, WPARAM wp, LPARAM lp); |

Четыре параметра оконной процедуры идентичны первым четырем полям структуры **MSG**. Первый параметр функции содержит дескриптор окна, получающего сообщение. Во втором параметре указывается идентификатор сообщения. Для системных сообщений зарезервированы номера от 0 до 1024. Третий и четвертый параметры содержат дополнительную информацию, которая распознается системой в зависимости от типа полученного сообщения.

Обычно в оконной процедуре используют оператор **switch** для определения того, какое сообщение получено и как его обрабатывать. Если сообщение обрабатывается, то оконная процедура обязана вернуть нулевое значение. Все сообщения, не обрабатываемые оконной процедурой, должны передаваться системной функции **DefWindowProc**. В этом случае оконная процедура должна вернуть то значение, которое возвращает **DefWindowProc**.

При обработке сообщения **WM\_DESTROY** оконная процедура вызывает функцию API **PostQuitMessage**. Значение параметра этой функции будет использовано как код возврата программы. Вызов **PostQuitMessage** приводит к посылке приложению сообщения **WM\_QUIT**, получив которое, функция **GetMessage** возвращает нулевое значение и завершает тем самым цикл обработки сообщений и, следовательно, приложение.