**Интерфейс прикладного программирования Win32 API**

Системные вызовы в операционных системах семейства MS Windows, начиная с версии Windows 95, реализованы на основе интерфейса прикладного программирования, получившего название «Win32 API». Под Win32 API понимают совокупность функций, предоставляющих программисту возможность создавать приложения для операционных систем Windows 95/98/ME/NT/2000/XP, базирующихся на использовании 32-х разрядных процессоров Intel, начиная с i386 (и его аналогов). При этом, несмотря на различия между версиями операционной системы, основное множество функций API для них одно и то же. Большинство функций API доступны для вызова из программ на любом исходном языке программирования (в том числе и на ассемблере).

Функции API хранятся в так называемых динамических библиотеках (Dynamic Link Library), которые размещаются в файлах с расширением dll, таких как kernel. dll, user32.dll, gdi32.dll и некоторых других. Эти файлы размещаются в системном каталоге Windows (обычно C:\WINDOWS\SYSTEM).

Фактически функции API для Windows играют ту же самую роль, что и программные прерывания для MS DOS, однако вызов функций API производится более простым и привычным для программиста способом - через символические имена. Например, функция удаления файла вызывается по имени DeleteFile, функция установки системного времени - SetSystemTime и т. д.

При программировании на ассемблере передача параметров функциям Win32 осуществляется не через регистры процессора, а через стек. Результат работы функции API помещается в регистр EAX. Более сложные типы данных возвращаются через адреса памяти (указатели), передаваемые функции в виде входных параметров.

При программировании на языках высокого уровня используются символические имена параметров, а результат передается через возвращаемое функцией значение.

**3.2. Типы данных, применяемые в Win32 API**

Помимо совокупности функций API Windows поддерживает целый ряд специальных типов данных (например, HINSTANCE, HWND, LPSTR и т. п.), не совпадающих со стандартными типами, определенными в основных языках программирования. Использование типов, специально «изобретенных» для Windows, упрощает написание программы, делает ее более ясной и читабельной. Некоторые простейшие типы данных Windows приведены в таблице 2.

Таблица 2. Основные типы данных Windows

|  |  |  |  |
| --- | --- | --- | --- |
| Тип | Определение С | Длина, б | Описание |
| **BOOL** | int | 4 | Логическая переменная, принимающая значения TRUE (ИСТИНА) или FALSE (ЛОЖЬ). |
| **BYTE** | unsigned char | 1 | Байтовое число без знака |
| **DWORD** | unsigned long | 4 | 32-разрядное целое число без знака. |
| **LONG** | signed long | 4 | 32-разрядное целое число со знаком. |
| **UINT** | unsigned int | 4 | 32-разрядное целое число без знака. |
| **LPSTR** | char far \* | 4 | Дальний указатель на строку символов с завершающим нулевым символом |
| **WORD** | unsigned short | 2 | 16-разрядное целое число без знака. |
| **HANDLE** | unsigned long | 4 | Дескриптор объекта (четырехбайтовое целое число) |

Особую роль в Windows играют специальные переменные - дескрипторы (хэндлы). Дескрипторы – это уникальные целые четырехбайтовые числа, применяемые для идентификации объектов, которые создаются и используются в системе. Большинство дескрипторов являются значениями индексов внутренних таблиц, которые Windows использует для доступа и управления своими объектами. Прикладная программа может получить или изменить данные, связанные с каким-либо объектом, только с помощью вызова функции API с указанием дескриптора соответствующего объекта. Для каждого вида объектов используется специальный дескрипторный тип, например – HWND – дескриптор окна, HDC - дескриптор контекста устройства, HFILE – дескриптор открытого файла, HLOCAL - дескриптор локального блока памяти и т. д. Общим для всех дескрипторов является наличие в описании первого символа “H”.

Помимо типов данных, представленных в таблице 2, в Windows существуют и другие, в том числе более сложные типы (структуры). Например, тип MSG представляет структуру, описывающую параметры выводимого на экран сообщения, а тип WNDCLASS – параметры окна приложения. В качестве примера рассмотрим структуру SYSTEMTIME – системное время, описываемую следующим образом:

typedef struct \_SYSTEMTIME {  
WORD wYear; //текущий год  
WORD wMonth; //номер месяца (январь-1, и т. д.)  
WORD wDayOfWeek; //день недели (вск-0, пн-1, …)  
WORD wDay; //день месяца  
WORD wHour; //час  
WORD wMinute; //минуты  
WORD wSecond; //секунды  
WORD wMilliseconds; //миллисекунды

} SYSTEMTIME,  
\*PSYSTEMTIME;

При написании программ на языке C/C++ типы данных Windows и прототипы функций API определяются во включаемых заголовочных файлах Win32, основным из которых является файл windows. h. Помимо типов данных в этом файле определено более 1000 констант. Имена констант стандартизированы: они пишутся заглавными буквами и имеют вид «префикс\_пояснение». Например, IDC\_RESOURCE, CS\_HREDRAW, WM\_QUIT, DRIVE\_UNKNOWN и т. п. Константы также широко применяются при установке значений параметров вызова функций API и при проверке результатов их выполнения.

Одной из особенностей программ, написанных для Windows, является использование так называемой «венгерской нотации» при записи имен переменных. Суть этой системы можно определить следующими правилами:

-  каждое слово в имени переменной пишется с прописной буквы и слитно с другими словами. Например, идентификатор для обозначения какой-то переменной может выглядеть следующим образом - MyVariable, YourVariable, VariableForSavingAnotherVariable и т. п.;

-  каждый идентификатор предваряется несколькими строчными символами, определяющими его тип. Например, целочисленная переменная MyVariable будет выглядеть как nMyVariable (n – общепринятый префикс для целочисленных переменных), символьная (char) переменная YourVariable превращается в cYourVariable. Указатель на строку символов, заканчивающуюся нулевым байтом, следут записать lpszVariableForSavingAnotlierVariable (lpsz - сокращение от Long Point то String with Zero). Как видим, префикс указателя может комбинироваться с другими префиксами. Примеры подобных префиксов приведены в таблице 3.

Таблица 3. Префиксы, применяемые в венгерской нотации

|  |  |  |  |
| --- | --- | --- | --- |
| Префикс | Тип данных | Префикс | Тип данных |
| **by** | BYTE (unsigned char) | **w** | WORD (unsigned int) |
| **c** | char | **dw** | DWORD (unsigned long) |
| **i** | int | **h** | HANDLE |
| **n** | int или short | **s** | string |
| **l** | LONG (long) | **sz** | string с завершающим нулем |
| **lp** | far\* (дальний указатель) | **u** | UINT (unsigned int) |

Применение венгерской нотации не является обязательным, но позволяет упростить процесс чтения и понимания программ, а также делает переменные в некотором смысле самоопределенными - имя переменной определяется ее типом. Отметим также, что параметры функций API представлены в документации «по-венгерски».

**3.3. Использование функций Win32 API при программировании на языке С/С++**

Приложения, разрабатываемые с использованием функций Win32 API, могут быть двух типов: оконные и консольные.

При создании оконного приложения предполагают, что программа будет выполняться в собственном окне с использованием всех возможностей графического пользовательского интерфейса Windows. В этом случае исходный текст программы должен подчиняться определенным требованиям. В частности, главная программа должна иметь имя WinMain() и включать секции подготовки и создания класса окон с заданными характеристиками, создания экземпляра окна только что созданного класса, циклического опроса очереди сообщений приложения и передачи сообщения оконной функции WindowFunction(), обрабатывающей полученное сообщение. Любая, даже самая простая оконная программа, будет состоять как минимум из этих двух функций, и включать несколько десятков строк кода.

Консольное приложение предназначено для поддержки выполнения программ, работающих в текстовом режиме. В этом случае вместо собственного окна приложению выделяется стандартное окно, имитирующее текстовый терминал. Никаких действий по настройке атрибутов консольного окна от программиста не требуется, поэтому консольная программа Windows ничем не отличается по форме от программ, написанных в стиле MS DOS. Например, точкой входа в консольную программу является функция main(). Отметим, что консольному приложению доступны как функции API, так и все стандартные потоки ввода-вывода DOS.

Для применения функций Win32 API и предопределенных типов данных Windows С-программа должна использовать директивы препроцессора, включающие в процесс компиляции файл windows. h (и, возможно, другие файлы):

#include "windows. h"

Рассмотрим пример консольной программы, использующей функцию Win32 API, предназначенную для получения информации о дисковом устройстве: GetVolumeInformation. Прототип функции выглядит следующим образом:

BOOL GetVolumeInformation(

LPCTSTR lpRootPathName,

LPTSTR lpVolumeNameBuffer,

DWORD nVolumeNameSize,

LPDWORD lpVolumeSerialNumber,

LPDWORD lpMaximumComponentLength,

LPDWORD lpFileSystemFlags,

LPTSTR lpFileSystemNameBuffer,

DWORD nFileSystemNameSize);

Как видим, функция возвращает значение типа BOOL и имеет 8 параметров:

lpRootPpathName – входной параметр (in), являющийся указателем на строку символов, содержащую имя корневого каталога диска. Необходимо, чтобы строка завершалась символом / (backslash). Если указать значение NULL, то будет выбран текущий диск.

lpVolumeNameBuffer – выходной параметр (out), являющийся указателем на [буфер](https://pandia.ru/text/category/bufer/), куда будет помещено имя диска.

nVolumeNameSize – входной параметр (in), содержащий длину буфера в символах.

lpVolumeSerialNumber – выходной параметр (out), являющийся указателем на переменную, содержащую серийный номер диска.

lpMaximumComponentLength – выходной параметр (out), являющийся указателем на переменную, содержащую максимальное количество символов в имени файла для данной файловой системы.,

lpFileSystemFlags – выходной параметр (out), являющийся указателем на переменную, содержащую флаги, характеризующие файловую систему. Возможные значения флагов представлены в виде набора констант с префиксом FILE\_, которые в данном примере не рассматриваются.

lpFileSystemNameBuffer – выходной параметр (out), являющийся указателем на буфер, куда будет помещено имя файловой системы (возможные имена: FAT, NTFS и т. п.).

nFileSystemNameSize - входной параметр (in), содержащий длину буфера имени файловой системы в символах.

Возвращаемое значение типа BOOL:

0 - если по какой-либо причине информация о диске не может быть представлена (например, ошибка в исходных данных);

≠0 - в противном случае.

// Пример использования функции GetVolumeInformation

//

#include "windows. h"

#include "iostream. h"

void main()

{

char NameBuffer[MAX\_PATH];

char SysNameBuffer[MAX\_PATH];

DWORD VSNumber;

DWORD MCLength;

DWORD FileSF;

if (GetVolumeInformation("C:\\",NameBuffer,

sizeof(NameBuffer),&VSNumber,&MCLength,&FileSF,

SysNameBuffer, sizeof(SysNameBuffer)))

{

cout << NameBuffer << endl;

cout << SysNameBuffer << endl;

cout << VSNumber << endl;

}

else cout << “Ошибка” << endl;

}

В результате выполнения этой программы в консольном окне появится, например, следующее:

WIN98

FAT

Press any key to continue

Обратите внимание, что указатели типа LPSTR могут объявляться как char, а указатели на числовые переменные при передаче параметров записываются с помощью оператора &

**3.4. Использование функций Win32 API при программировании на Ассемблере**

При написании приложений, использующих вызовы функций Win32 API, на языке ассемблера следует руководствоваться следующими правилами:

·  Функции Win32 API выполняются в защищенном режиме процессора (прерывания DOS - в реальном или виртуальном режиме), поэтому при использовании функций Win32 API программа должна включать директиву.386

·  Модель памяти для Win32 приложений задается директивой. model flat

·  Вызов функций API производится с помощью команды call по адресу, задаваемому именем функции (например, call MessageBoxA). Прототипы функций размещаются в файле Import32.lib, называемом библиотекой импорта. Имена всех используемых системных функций описываются как внешние с помощью директивы extrn

·  Передача параметров осуществляется через стек (а не через регистры, как это было при вызове функций DOS). Перед вызовом функции необходимо записать в стек значения всех указанных параметров используемой функции, начиная с последнего. Каждый параметр всегда представляется 32-битным числом (в котором могут быть задействованы не все биты).

·  Значение каждой функции возвращается в регистре EAX. Если функция возвращает структуру данных, то регистр EAX содержит код возврата, а адрес структуры необходимо передать функции как параметр.

Для пояснения указанных правил, рассмотрим пример программы, выводящей сообщение во всплывающее информационное окно, с помощью функции MessageBoxA, имеющей следующие параметры:

HWND hParentWnd, // дескриптор родительского окна  
LPSTR lpszText, // текст сообщения

LPSTR lpsz Caption, // заголовок окна

DWORD nType // стиль окна

.386 ;тип процессора (i386 или старше) ;использование защищенного режима  
.model flat, STDCALL ; модель памяти для Win32  
;Описание используемых внешних процедур Win32 API:  
extrn MessageBoxA:proc ;функция вывода сообщения  
extrn ExitProcess:proc ;функция завершения процесса

.data ;сегмент данных  
title db 'Изучаем Win32 API',0 ;заголовок окна  
message db 'Это я, [Bill Gates](https://pandia.ru/text/category/bill_gejts/" \o "Билл Гейтс)!',0 ;текст сообщения

.code ;сегмент кода  
start:  
push 40h ;стиль окна - одна кнопка "OK"

;и пиктограмма "i"

push offset title ;адрес строки с заголовком  
push offset message ;адрес строки с сообщением  
push 0 ;дескриптор программы-владельца

;создаваемого окна

call MessageBoxA ;вызов функции вывода сообщения

push 0 ;код завершения программы  
call ExitProcess ;вызов функции завершения программы

end Start ;конец сегмента кода

Анализируя текст приведенного выше примера, обратите внимание на следующие обстоятельства:

·  используемые в приложении функции Win32 API перечисляются как внешние (директива extrn);

·  имена функций Win32 обычно дополняются символом A, например RemoveDirectoryA, GetComputerNameA (это означает, что применяются версии функций, использующие для представления символов кодировку ANSI);

·  названия функций записываются строго с соблюдением регистра (прописные и строчные буквы);

·  нельзя нарушать установленную последовательность записи параметров в стек;

·  типы параметров должны соответствовать указанным в прототипе функции;

·  указатели (адреса) параметров формируются с помощью оператора offset.

**3.5. Рабочее задание**

1.  Для заданного варианта (табл. 4) написать и отладить консольную программу на языке C и программу на языке ассемблера, обеспечивающую реализацию указанной функции Win32 API. В таблице символами [in] помечены параметры, содержащие исходные данные, а символами [out] – результаты работы вызываемой функции. Все входные параметры должны получить значения! Типы LPTSTR и LPCTSTR считать эквивалентными типу LPSTR. Программы представить в виде исполнимых (exe) файлов и продемонстрировать их работу запуском из командной строки.

2.  Программу на языке С организовать таким образом, чтобы ввод данных пользователя осуществлялся через командную строку. B языке C для работы с командной строкой используются параметры функции main():

**main (int argc, char \*argv[]);**

где argc принимает значения числа параметров, записанных в командной строке при запуске программы, включая имя программы;

argv - символьный массив, содержащий параметры командной строки, причем argv[0] содержит имя программы, а argv[1],argv[2] и т. д. - 1-й, 2-й и т. д. параметры.

Предусмотреть программный контроль ситуаций, когда пользователь не ввел требуемые параметры или сделал это неправильно, формируя соответствующие сообщения.

Производить контроль возвращаемого функцией API значения с выдачей соответствующих сообщений.

Настройка параметров среды Borland C++ для создания консольного приложения приведена в приложении.

3. В программе на языке ассемблера исходные данные следует задавать непосредственно в тексте программы. Необходимо производить контроль возвращаемого функцией API значения (регистр EAX) с выдачей соответствующих сообщений с помощью функции MessageBoxA.

При разработке программы на языке ассемблера рекомендуется создать командный файл asmwin. bat следующего вида (вместо слова *путь* нужно указать реальный путь доступа к соответствующим файлам):

del %1.exe  
путь\tasm32 %1.asm /mx/m  
путь\tlink32 %1.obj,,,путь\import32.lib /Tpe /aa  
del %1.obj  
del %1.map

Исходный текст разрабатываемой программы сохраняют в файле с расширением ".asm". В этом случае загрузочный файл (exe) может быть получен путем ввода в командной строке:

asmwin. bat имя\_программы\_без\_расширения

4. Программы должны содержать шапку в виде комментария с указанием номера и названия лабораторной работы, фамилии студента и номера группы, номера варианта и назначения программы, а также описания прототипа вызываемой функции.

5. Отчет по лабораторной работе должен содержать: текст задания, блок-схемы алгоритмов и распечатки текстов программ, результаты работы программ (скрин-шоты) в различных ситуациях.

Таблица 4. Описание функций Win32 API (варианты заданий)

|  |  |  |
| --- | --- | --- |
| № | Функция | Параметры и возвращаемое значение |
| 1. | Создание каталога  BOOL CreateDirectory (  LPCTSTR lpszPath,  LPSECURITY\_ATTRIBUTES lpsa); | lpszPath - [in] имя каталога,  lpsa - [in] атрибуты безопасности (принять равным NULL).  Функция возвращает 0 (FALSE) в случае ошибки. |
| 2. | Удаление каталога  BOOL RemoveDirectory (  LPCTSTR lpszPath); | lpszPath [in] имя каталога  Функция возвращает 0 (FALSE) в случае ошибки. |
| 3. | Получение имени текущего каталога  DWORD GetCurrentDirectory (  DWORD cchCurDir,  LPTSTR lpszCurDir); | cchCurDir – [in] длина буфера (в байтах) для имени каталога  lpszCurDir - [out] буфер для имени каталога. Функция возвращает 0 в случае ошибки, или длину имени каталога в противном случае. |
| 4. | Замена текущего каталога  BOOL SetCurrentDirectory (  LPCTSTR lpszCurDir); | lpszCurDir – [in] имя каталога (относительное или полное). |
| 5. | Удаление файла  BOOL DeleteFile (  LPCTSTR lpFileName); | LpFileName - [in] имя удаляемого файла. Функция возвращает 0 (FALSE) в случае ошибки. |
| 6. | Переименование файла  BOOL MoveFile (  LPCTSTR lpszExisting,  LPCTSTR lpszNew); | lpszExisting – [in] имя существующего файла или каталога  lpszNew - [in] имя нового файла или каталога, который не должен существовать.  Функция возвращает 0 (FALSE) в случае ошибки. |
| 7. | Получение имени временного каталога Windows  DWORD GetTempPath (  DWORD nBufferLength,  LPTSTR lpBuffer); | nBufferLength – [out] длина буфера (в байтах) для имени каталога  lpBuffer – [out] буфер для имени каталога. Функция возвращает 0 в случае ошибки, или длину имени каталога в противном случае. |
| 8. | Получение имени компьютера  BOOL GetComputerName (  LPTSTR lpBuffer, LPDWORD nSize); | nSize – [in] длина буфера;  lpBuffer - [out] буфер для имени.  Функция возвращает 0 (FALSE) в случае ошибки. |
| 9. | Получение имени системного каталога  UINT GetSystemDirectory (  LPTSTR lpBuffer,  UINT uSize); | lpBuffer - [out] буфер для имени; uSize - [out] длина буфера.  Функция возвращает 0 в случае ошибки, или длину имени каталога в противном случае. |
| 10. | Получение имени пользователя  BOOL GetUserNameA ( LPTSTR lpBuffer,  LPDWORD nSize); | lpBuffer - [out] буфера для имени; nSize –[out] указатель на длину буфера  Функция возвращает 0 (FALSE) в случае ошибки. |
| 11. | Получение имени каталога Windows  UINT GetWindowsDirectory ( LPTSTR lpBuffer,  UINT uSize); | lpBuffer -[out] буфер для имени; uSize - [out] длина буфера.  Функция возвращает 0 в случае ошибки, или длину имени каталога в противном случае. |
| 12. | Получение информации о дисковом пространстве  (до 2 Гб)  BOOL GetDiskFreeSpace (  LPCTSTR lpRootPathName,  LPDWORD lpSectorsPerCluster,  LPDWORD lpBytesPerSector,  LPDWORD lpNumberOfFreeClusters,  LPDWORD lpTotalNumberOfClusters); | LpRootPathName - [in] имя корневого каталога (обязательно указать \).  LpSectorsPerCluster - [out] количество секторов в кластере  LpBytesPerSector - [out] количество байт в секторе  LpNumberOfFreeClusters - [out] количество свободных кластеров  LpTotalNumberOfClusters - [out] общее количество кластеров  Функция возвращает 0 (FALSE) в случае ошибки. |
| 13. | Получение заголовка консольного окна  DWORD GetConsoleTitle(  LPTSTR lpConsoleTitle,  DWORD nSize ); | lpConsoleTitle - [out] указатель на буфер для строки заголовка  nSize - [in] размер буфера  Функция возвращает 0 в случае ошибки, или длину заголовка в противном случае. |
| 14. | Копирование файла.  BOOL CopyFile(  LPCTSTR lpExistingFileName,  LPCTSTR lpNewFileName,  BOOL bFailIfExists); | LpExistingFileName - [in] имя исходного файла (не более MAX\_PATH)  lpNewFileName - [in] имя целевого файла  bFailIfExists - [in] TRUE, если требуется блокировать копирование в существующий файл, в противном случае FALSE.  Функция возвращает 0 (FALSE) в случае ошибки. |
| 15. | Задание заголовка консольного окна  BOOL SetConsoleTitle(  LPCTSTR lpConsoleTitle); | lpConsoleTitle - [in] строка заголовка  Функция возвращает 0 (FALSE) в случае ошибки. |
| 16. | Задание метки диска  BOOL SetVolumeLabel(  LPCTSTR lpRootPathName,  LPCTSTR lpVolumeName); | lpRootPathName - [in] имя корневого каталога (обязательно указать \).  lpVolumeName - [in] метка диска  Функция возвращает 0 (FALSE) в случае ошибки. |
| 17. | Получение каталога размещения DLL библиотек  DWORD GetDllDirectory(  DWORD nBufferLength,  LPTSTR lpBuffer); | nBufferLength - [in] размер символьного буфера.  LpBuffer - [out] указатель на буфер для размещения имени каталога  Функция возвращает 0 в случае ошибки, или длину имени каталога в противном случае. |
| 18. | Получение типа исполняемого файла  BOOL GetBinaryType(  LPCTSTR lpApplicationName,  LPDWORD lpBinaryType); | lpApplicationName - [in] полное имя файла  lpBinaryType - [out] указатель на возвращаемый код типа: 0 – Win32, 1 – Win64, 2 – MSDOS, 6 – Win16  Функция возвращает 0 (FALSE) в случае ошибки и если файл не является исполнимым |
| 19. | Получение типа диска  UINT GetDriveType (  LPCTSTR lpRootPathName); | lpRootPathName - [in] имя корневого каталога (обязательно указать \).  Функция возвращает код типа диска (0 – диск не найден, 1 – неверное имя корневого каталога, 2 – сменный диск, 3 - жесткий диск, 4 – внешний (сетевой) диск, 5 - компакт-диск, 6 - віртуальный диск) |
| 20. | Получение атрибутов файла  DWORD GetFileAttributes(  LPCTSTR lpFileName); | lpFileName - [in] имя файла  При успешном завершении функция возвращает код атрибута:  FILE\_ATTRIBUTE\_ARCHIVE – архивный,  FILE\_ATTRIBUTE\_HIDDEN – скрытый,  FILE\_ATTRIBUTE\_READONLY – только чтение,  FILE\_ATTRIBUTE\_SYSTEM - системный  FILE\_ATTRIBUTE\_TEMPORARY - временный.  В случае ошибки возвращаемое значение - INVALID\_FILE\_ATTRIBUTES. |
| 21. | Получить системное время  void GetSystemTime(  LPSYSTEMTIME lpSystemTime); | lpSystemTime [out] - указатель на структуру SYSTEMTIME |
| 22. | Получить версию ОС  DWORD GetVersion(VOID); | Возвращаемое значение:  Ст. бит=0, мл. байт=4 — WinNT;  Ст. бит=0, мл. байт=5 — Win2000/XP;  Ст. бит=1, мл. байт=4 — Win95/98/ME; |
| 23. | Получить информацию о логических дисках в числовом виде  DWORD GetLogicalDrives(VOID); | Возвращает 32-разрядное значение, каждый бит которого соответствует устройству: 1-й бит-A, 2-й бит-В и т. д. |
| 24. | Получить информацию о логических дисках в строковом виде  DWORD GetLogicalDrivesStrings(  DWORD nBufferLength,  LPSTR lpBuffer); | lpBuffer - [in] указатель на буфер для строки имен дисков  nBufferLength - [in] размер буфера  Функция возвращает 0 в случае ошибки, или длину заголовка в противном случае. |

**3.6. Контрольные вопросы**

1.  Как организованы системные вызовы в ОС семейства Windows?

2.  Где размещаются коды системных функций Windows?

3.  Где описаны и с какой целью введены специальные типы данных Windows?

4.  Какие компоненты входят в состав включаемого файла windows. h?

5.  Что означает и для чего используется «венгерская нотация»?

6.  Поясните особенности организации оконных и консольных приложений Windows.

7.  Как организованы вызов функций API, передача параметров и получение результатов при создании приложений на языке С?

8.  Как организованы вызов функций API, передача параметров и получение результатов при создании приложений на языке Ассемблера?