```cpp
// computerInfo.cpp : Defines the entry point for the console application.
// This sample is a couple years old now.
#include "stdafx.h"
#include <cmath>
#include <iostream>
#include "computerInfo.h"
#include <string>
#include <windows.h>
#include "Intrin.h"
using std::string;
LPFN_ISWOW64PROCESS fnIsWow64Process;
/* This is a system info program. I did a lot of googling to learn what I needed to write this.
   All MS sample code used is licensed under the MICROSOFT LIMITED PUBLIC LICENSE and copyright to MS (according to the usage
docs)*/
// I must point out that you need "Multi-Byte Character Set" strings enabled in the configuration to build this program. Also,
// end users need the Visual Studio 2010 C++ runtime.
int _tmain(int argc, _TCHAR* argv[]){
    // Get the OS version. tutorial: msdn.microsoft.com/en-us/library/ms724451(VS.85).aspx
    OSVERSIONINFOEX osviex;
    ZeroMemory(&osviex, sizeof(OSVERSIONINFOEX));
    osviex.dwOSVersionInfoSize = sizeof(OSVERSIONINFO);
    GetVersionEx ((LPOSVERSIONINFO)&osviex);
    printf("OS:");
    // This does not include server OS's and all I can test is Win 7 and XP (in XP mode).
    if(osviex.dwMajorVersion == 5){
        if(osviex.dwMinorVersion == 0){
            std::cout << " Windows 2000";
        }else if(osviex.dwMinorVersion == 1){
            std::cout << " Windows XP";
        }
        else if(osviex.dwMinorVersion == 2){
            std::cout << " Windows XP 64-bit";
        }
    }else if(osviex.dwMajorVersion == 6){
        if(osviex.dwMinorVersion == 0){
            std::cout << " Windows Vista";
        }else if(osviex.dwMinorVersion == 1){
            std::cout << " Windows 7";
        }
    }else{
        std::cout << "  Other OS";
    }
    // Print the edition which for me will always be pro.
```

```
    if(osviex.wSuiteMask == VER_SUITE_EMBEDDEDNT){
        std::cout << "  Embedded \n";
    }
    else if(osviex.wSuiteMask == VER_SUITE_PERSONAL){
       std::cout << "  Home \n";
    }
    else{
       std::cout << "  Professional \n";
       // assuming pro by default
    }

    // Get the CPU speed. (This should be in the registry.)
    HKEY hkey;
    DWORD speed;
    DWORD keys;
    DWORD dataSize =  sizeof(LPBYTE);
    TCHAR * keyName = TEXT("Hardware\\Description\\System\\CentralProcessor\\0");
    TCHAR * buffer = TEXT("");
    TCHAR * result = EnsureAlignedString(keyName, buffer, sizeof(keyName));
    PBOOL sixtyFourBit = false;
    if(IsWow64()){
        keys = KEY_QUERY_VALUE  ^ KEY_WOW64_32KEY;
    }else{
        keys = KEY_QUERY_VALUE;
    }
    long errorCode = RegOpenKeyEx(HKEY_LOCAL_MACHINE, result,
                                  0/* reserved, must be 0 */, keys, &hkey);
    if (errorCode == ERROR_SUCCESS){
        long innerresult =
        RegQueryValueEx(hkey, TEXT("~MHz"), NULL/* reserved, must be null */,NULL, (LPBYTE)&speed, &dataSize);
        if(innerresult != ERROR_SUCCESS){
           std::cout << "\n something went wrong reading the key; error code:" << innerresult << "\n";
           system("pause");
           exit(1);
        }
        // else it works and "speed" has the speed.
    }
    else{
        std::cout << "\n something went wrong with the reg key grab; error code:" << errorCode << "\n" ;
        system("pause");
        exit(1);
    }
```

```cpp
// Get the CPU name (the compiled version of this should be platform dependent).
// Got this from http://msdn.microsoft.com/en-us/library/hskdteyh%28v=vs.80%29.aspx (code sample at the bottom).
// Using a low level way of getting CPU info. This is done by design as a contrast to the high level approuch used above.
int infoReturned[4];
char CPUBrandString[0x40];
memset(CPUBrandString, 0, sizeof(CPUBrandString));
__cpuid(infoReturned, 0x80000000);
unsigned  nExIds = infoReturned[0];
// Calling __cpuid with 0x80000000 as the InfoType argument
// gets the number of valid extended IDs.
for (unsigned i=0x80000000; i<=nExIds; ++i){
    __cpuid(infoReturned, i);
    if  (i == 0x80000002){
        memcpy(CPUBrandString, infoReturned, sizeof(infoReturned));
    }
    else if  (i == 0x80000003){
        memcpy(CPUBrandString + 16, infoReturned, sizeof(infoReturned));
    }
    else if  (i == 0x80000004){
        memcpy(CPUBrandString + 32, infoReturned, sizeof(infoReturned));
    }
}
std::cout << "CPU:" << speed / 1000 << "GHz " << CPUBrandString << "\n";

// how much total ram do I have; http://msdn.microsoft.com/en-us/library/windows/desktop/aa366589%28v=vs.85%29.aspx
MEMORYSTATUSEX statex;
statex.dwLength = sizeof (statex);
GlobalMemoryStatusEx (&statex);
double totalRam = (double) statex.ullTotalPhys;
totalRam = totalRam / 1073741824 ;
printf("Total Ram: ~ %1.1f GB \n", totalRam);

// get free hard disk space
char var[105]; // max size of the string
char* ldBuffer = var;
DWORD debug = GetLogicalDriveStrings(101, (LPSTR) ldBuffer);
string driveStrings[26];// Drives in Windows go from A - Z; I don't know about UNIX.
int driveStringCounter = 0;
string tempStr = "";
for(int counter = 0; counter < 105; counter++){
    if(ldBuffer[counter] == '\0' && ldBuffer[counter -1 ] == '\0'){
         //system("pause"); // remove when done debugging this part
```

```
            break; // end of drive letter data.
        }
        if(ldBuffer[counter] == '\0'){
            driveStrings[driveStringCounter] = tempStr;
            tempStr.clear();
            driveStringCounter++;
        }
        else{
            tempStr += ldBuffer[counter];
        }
    }
    ULARGE_INTEGER* lpFreeBytesAvailable = new ULARGE_INTEGER();
    ULARGE_INTEGER* lpTotalNumberOfBytes = new ULARGE_INTEGER();;
    ULARGE_INTEGER* lpTotalNumberOfFreeBytes = new ULARGE_INTEGER();// this variable gives the grand total;
                                                                    //the other two values incude possible quotas.
    for(int index = 0; index < driveStringCounter; index++){
        LPCTSTR driveStringLP = (LPCTSTR) driveStrings[index].c_str();
        if(GetDriveType(driveStringLP) != DRIVE_FIXED){
            continue;
        }
        if(GetDiskFreeSpaceEx(driveStringLP, lpFreeBytesAvailable, lpTotalNumberOfBytes, lpTotalNumberOfFreeBytes)){
            ULONGLONG QuadPart = lpTotalNumberOfFreeBytes->QuadPart;
            double rounded =(double) QuadPart / 1073741824;
            printf("~ Free Disk space on  hard drive %s: %1.1f GB \n", driveStrings[index].c_str(), rounded);

        }else{
            std::cout << "\n something went wrong trying to get free disk space!\n";
            std::cerr << "error: " << GetLastError() << "!\n";
            system("pause");
            exit(1);
        }
    }
    // clean up.
    delete lpFreeBytesAvailable, lpTotalNumberOfBytes, lpTotalNumberOfFreeBytes, tempStr;


    // Get user name.
    char var2[20]; // I've heard Windows user names have a max of 20 chars.
    ldBuffer = var;
    DWORD value = 20;
    LPDWORD size = &value;
    if (GetUserName((LPSTR)ldBuffer, size)){
    std::cout << "User name: " << ldBuffer << "\n";
```

```
    }else{
        std::cout << "\n something went wrong trying to get your user name.\n";
        std::cerr << "error: " << GetLastError() << "!\n";
        system("pause");
        exit(1);
    }

    system ("pause");
    return 0;
}
/* Thanks to Raymond Chen - MSFT - http://msdn.microsoft.com/en-us/library/windows/desktop/ms724911%28v=vs.85%29.aspx
   Published as an article contributor; May be used and modified freely (according to the contributor agreement).
*/
TCHAR *EnsureAlignedString(TCHAR *in_Str, TCHAR *in_Buf, INT in_MaxLen)
{
    // registry calls need unicode strings to be aligned on 16-bit boundaries
    if ((INT)in_Str & 1)
    {
        // alignment is needed, copy to aligned buffer
        strncpy_s(in_Str, sizeof(in_Buf),in_Str, in_MaxLen);

        // use the buffer
        return in_Buf;
    }
    else
    {
        // already aligned
        return in_Str;
    }
}
// from http://msdn.microsoft.com/en-us/library/windows/desktop/ms684139%28v=vs.85%29.aspx
// This is from a Microsoft sample.
BOOL IsWow64()
{
    BOOL bIsWow64 = FALSE;

    //IsWow64Process is not available on all supported versions of Windows.
    //Use GetModuleHandle to get a handle to the DLL that contains the function
    //and GetProcAddress to get a pointer to the function if available.

    fnIsWow64Process = (LPFN_ISWOW64PROCESS) GetProcAddress(
        GetModuleHandle(TEXT("kernel32")),"IsWow64Process");
```

```
    if(NULL != fnIsWow64Process)
    {
        if (!fnIsWow64Process(GetCurrentProcess(),&bIsWow64))
        {
            std::cout << "\n something went wrong trying to determine your computer's bit rate (32-bit or 64)";
            system("pause");
            exit(1);
        }
    }
    return bIsWow64;
}
```