EECS 4980 / 5980 Programming Assignment
Due Saturday, April 29, 2017 at 11:59:59 PM.

> ***NOTE***: Under no circumstances are you allowed to use *any* code from *any* source other than what you individually create from scratch, or are provided by the instructor. You may, of course, use any information in the text and the lecture slides, including the FIPS links provided on SHA. Source code for SHA is readily available from the web; *don't use any of it, even as a reference (except as explicitly allowed). If I believe you didn't write your code, I'll ask you to explain it, potentially line-by-line, to prove to me that it's yours and yours alone.*

Your assignment is to write a program to create the SHA-512 hash of a file. Use Visual Studio 2015 to code your **Win32 Console Application** in C/C++. Call your program "SHA512".

The command-line syntax for your program will be:

```
SHA512 <file>
```

Obviously, the filename parameter is required, and you should check to make sure you can open the file. If the user omits the file parameter, specifies too many parameters, or the file can't be opened for input, you are to display a message showing what the proper command-line usage is.

The output of your program should be the 512-bit hash of the file, in hexadecimal, grouped 8 bytes at a time (all on one line, space-delimited), followed (on the next line) by the elapsed time it took to run.

Example:

```
SHA512.exe \shakespeare.txt
F0A1DC33CA071D53 232B0971A4638DAE A25EC22E47C7636C 9DD0AE5863EAF44D
3B0BBD7186EB02BE 1D093AA93E84D90B 17D0625907F4956B D71C7A8542BFC342
Elapsed Time:  0.030 Seconds
```

Note: in the example above, the output line is too long to fit, but on your console window, it should all be on one line. If a group of 16 hex digits begins with a zero, you need to display the zero, rather than having it be a 15-hex-digit value.

The Stallings text has sample values of variables A through H at the beginning of rounds 1, 2, and 3, for a three-byte file containing "abc". If your program correctly handles the first three rounds, it should be able to do all of them.

To compare your results with a known-to-be-good hash (other than the Shakespeare hash provided above), you may want to consider a package that creates SHA-512 hashes. One such program is HashCalc, by SlavaSoft (http://www.slavasoft.com/hashcalc/index.htm). It's freeware, so help yourself.

One thing that is a bit different about this assignment (compared to, say, DES and AES) is that there is no on-line reference that shows what the intermediate values should be (a la the Orlin Grabbe article for DES or the "stick figure" illustrations for AES. For this one, you will just have to be careful, and pay attention to detail.

Note: The input MUST be padded, and a 128-bit length field must be appended to the final block. If your data nearly fills up the last block (not leaving room for the padding and a length field), then you will need to output an additional block.

The length field is supposed to be 128 bits long, supporting files / streams up to 2^{128} bits long. Note that 2^{128} bits is 2^{125} *bytes*. Considering that a terabyte is only 2^{40} bytes, you may safely assume that the upper 64 bits of the length field will always be zero, and just use an `unsigned long long` for the lower 64 bits of the length in your padding scheme (i.e., you don't really have to generate a 128-bit integer).

Submit your code as a ZIP or 7-Zip archive (bzip, gzip, tar, rar, etc. are not acceptable) of the entire project tree (including all source and binary subdirectories). Post your submission to Blackboard. Make sure your code is commented, and that the comments include your name!

Make sure your project folder has your name in it. Ideally, it should be called "4980-<LastName><comma><FirstName>".

Your code should be well-designed (appropriately modularized) and well-documented. There is no reason for shoddy documentation; in fact, extra documentation will probably help you write the code. Resist the temptation to code it all with minimal / no comments and then add the documentation later.

The comments in your code should demonstrate to me that you understand how the code works.

If you have any questions, or run into difficulties, e-mail me as soon as possible. Do not delay in starting your work; this one can appear deceptively easy. I spent a LOT longer debugging it than I did writing it, and there were very few problems to fix. None of it is particularly difficult, but as they say, "the devil is in the details." Debugging crypto code is particularly challenging, because, even when it works correctly, the output looks like garbage, and that applies to cryptographic hashes every bit as much as it does to encryption algorithms.

DO NOT use bitsets or any other STL type or extension to C/C++. It should go without saying that you may not use a hashing library. You may use `_rotr64()` to do the rotations, if you like. Check the Visual Studio documentation for its use. As we discussed in class, you may find a `union` helpful, but it is certainly *not* required.

Think in terms of efficiency. Your completed code should run quickly (after all, this IS code that would be built-in to system utilities; we don't want to wait while we number-crunch our files just to hash them!). You should already have the 5 MB test file, SHAKESPEARE.TXT, which contains the complete works of Shakespeare's plays. Your program should hash this file in less than 1 second, depending on your CPU and speed. Compile your code as x64 release code, with optimizations for speed.