CSC 3350 Operating Systems Programming
Lab 1 -- HEX File Dump
Due: Friday, April 8, 2016 in class

Write a **C** console (command line application) program that will open and read in a file (of arbitrary data, e.g., text, binary, etc.) and print out a hexidecimal representation of the input file contents in the form shown below. Each line of output displays (in hex) the offset position in the file of the first byte on that line, the hex values of the next 16 bytes in the file, and the normal ASCII character representation for those 16 bytes:

C:-> hexdump lab1test.txt

```
HEX FILE DISPLAY of  lab1test.txt       Written by M. Tindall
              H E X    F I L E    D I S P L A Y
    Addr:  0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   ------ASCII-----
00000000: 54 68 69 73 20 69 73 20 61 20 73 61 6D 70 6C 65   This is a sample
00000010: 20 66 69 6C 65 20 66 6F 72 20 6C 61 62 31 2E 0D    file for lab1.
00000020: 0A 49 74 20 73 68 6F 77 73 20 75 70 70 65 72 63    It shows upperc
00000030: 61 73 65 20 41 42 43 2C 20 6C 6F 77 65 72 63 61   ase ABC, lowerca
00000040: 73 65 20 61 62 63 2C 20 0D 0A 6E 75 6D 62 65 72   se abc,   number
00000050: 73 20 30 31 32 33 34 35 36 37 38 39 2C 20 61 6E   s 0123456789, an
00000060: 64 20 73 70 65 63 69 61 6C 20 0D 0A 63 68 61 72   d special   char
00000070: 61 63 74 65 72 73 20 20 2F 3F 40 23 24 25 5E 26   acters  /?@#$%^&
00000080: 2A 28 29 2E 0D 0A 49 74 20 69 73 20 69 6D 70 6F   *().  It is impo
00000090: 72 74 61 6E 74 20 74 68 61 74 20 61 20 68 65 78   rtant that a hex
000000A0: 20 64 75 6D 70 20 70 72 6F 67 72 61 6D 20 72 65    dump program re
000000B0: 70 72 65 73 65 6E 74 20 6F 66 66 73 65 74 20 61   present offset a
000000C0: 64 64 72 65 73 73 65 73 20 64 69 73 70 6C 61 79   ddresses display
000000D0: 65 64 0D 0A 69 6E 20 68 65 78 69 64 65 63 69 6D   ed  in hexidecim
000000E0: 61 6C 20 70 72 6F 70 65 72 6C 79 20 66 6F 72 20   al properly for
000000F0: 6C 61 72 67 65 72 20 66 69 6C 65 73 20 74 68 61   larger files tha
00000100: 74 20 72 65 71 75 69 72 65 20 6D 6F 72 65 20 74   t require more t
00000110: 68 61 6E 20 31 36 20 6C 69 6E 65 73 20 6F 66 20   han 16 lines of
00000120: 6F 75 74 70 75 74 2E 0D 0A 1A                     output.
Final character count = 298 (0x12A)
```

Note: this program must be coded in the "C" programming language (not C++). Therefore, character string data uses the c-string ASCIIZ array of chars format, and I/O uses the <stdio.h> conventions, e.g.,

```
#include <stdio.h>
printf( ); putchar( ); FILE *f;
errno_t fopen_s(&f, "filename", "filemode" ); int ch = fgetc(f);  (ch == EOF) for end-of-file detection; fclose(f);
```

Your program must obtain the input filename from a command-line parameter, e.g. use main( ) prototype

int main(int argc, char *argv[]) // **argc**: count of cmdline args; **argv[i]**: i^{th} (char*) argument

To execute the program after a successful "Build", start a commandline window (Start/Run/cmd) and change into the subdirectory that contains the executable using the "cd" command (e.g., hexdump.exe. It is usually in the Debug subdirectory of the project). Then run the program by typing the executable name with any commandline arguments on the same line. Note that it is also possible to execute the program from within the Visual Studio project environment using the Debugger, allowing breakpoints, single-stepping, and variable interrogation.

If the user specifies instead the command-line switch **/?**, e.g., argc>1 and strcmp(argv[1], "/?") == 0

C:-> hexdump /?

print out a short message describing the function of this program and the command-line syntax, and then exit.

Note that the output shows **control characters** (e.g., 0D, 0A, 1A, etc.), which means the file must be opened for reading in the 'binary' filemode, e.g., "rb". The file can contain any type of data (e.g., text, or general binary). After opening the file, read in the first 16 bytes from the file into a small array buffer. Generate the output display for the values in the buffer, then loop around and refill the buffer with the next 16 bytes of data. Repeat until the end of the file is reached. Don't forget to output the last (possibly partial) buffer.

Normally, displaying a data value in hexidecimal in the C language is done using the "%x" printf formatter, such as
printf("Here are values 10 and 30 and 300 in hex: %02X, %02x, %04X\n", 10, 30, 300); // 0A, 1e, 012C

However, for this lab, DO NOT use printf(%x) or any other built-in conversion for any hex output.
Instead, convert a byte value (which is in the range 0 .. 255) into two 'hex' values (in the range 0 .. 15): the High 'nibble' and the Low 'nibble'. This can be done using the bitwise '&' and '>>' operators in C. Then the two nibbles can be individually converted into a character form ('0' - '9', 'a' - 'f') and output.

The Address values displayed at the start of each line represent the byte offset (in hex) into the file of the first byte on that line. As you read the input file, count the incoming characters using a 32-bit integer (int) variable. At the start of each new line, using an algorithm similar to converting byte values to hex, convert the integer counter value into a sequence of eight nibbles and output each nibble in hex character form.

When displaying the normal ASCII printing character for each byte in the buffer (on the right side of the output), output the blank character if the byte will not print correctly (printing characters are in the range 32 - 127).

Turn in:

1. **Program listing, documented**.
   Include a heading comment block with your name and a description of this lab. Each procedure/function should be commented as to purpose/parameters/results/etc. All variable declarations should be commented. Comments should be included in the code anytime they would improve the understanding, particularly for any unusual or obscure operation.

2. **Sample output demonstrating correct operation of your program.**
   Include a dump of a file containing **at least 256 characters.** Make sure that your sample file contains several lines of data, upper/lower case letters, digits, some punctuation and other special symbols, etc.

3. There are several ways to produce the printed output file.
   One of the simplest is to run your program from a CMD commandline window. After building your application, copy the executable (e.g., "lab1.exe" from the Debug directory) and the sample data file (e.g., "lab1test.txt") to an obvious location (such as the root directory of the disk drive). Then type in the commandline to run the program and redirect the standard console output to a text output file:

   C:-> lab1 lab1test.txt > lab1dump.txt

   Then you can use Textpad or Notepad to print the "lab1dump.txt" generated output file.

Program development hint:
For initial program implementation and testing, you might want to code using the printf(" %x ") formatted output capabilities. However, once the program is functionally operational, you must then replace this hex-output coding with your own 'do it yourself" hex conversion and outputting, as described in the lab requirements.