CS 240
Fall 2019

Project 4
Hexdump

Assigned: Monday, September 23, 2019
Due: 10:00 p.m., Sunday, October 6, 2019

Introduction. Consider the following C++ program:

```
// File:        DosRocks.cc
//
// Author:      Professor Bailey
//
// Date:        September 22, 2019
//
// Purpose:     A program outputs a Pro-DOS greeting
//

#include <iostream>

using namespace std;

int main() {
  cout << "DOS Rocks the bitzone!" << endl;
}
```

This is an ASCII file. What that means is that the bytes representing this file on disk have been encoded in ASCII. To see this, we can look at the raw data using the Unix `hexdump` utility:

```
hexdump -C DosRocks.cc | more
```

(I've piped this through the program `more` as well to page long output). The result is:

```
00000000  2f 2f 20 46 69 6c 65 3a  20 09 44 6f 73 52 6f 63  |// File: .DosRoc|
00000010  6b 73 2e 63 63 0a 2f 2f  0a 2f 2f 20 41 75 74 68  |ks.cc.//.// Auth|
00000020  6f 72 3a 09 50 72 6f 66  65 73 73 6f 72 20 42 61  |or:.Professor Ba|
00000030  69 6c 65 79 0a 2f 2f 0a  2f 2f 20 44 61 74 65 3a  |iley.//.// Date:|
00000040  09 53 65 70 74 65 6d 62  65 72 20 32 32 2c 20 32  |.September 22, 2|
00000050  30 31 39 0a 2f 2f 20 0a  2f 2f 20 50 75 72 70 6f  |019.// .// Purpo|
00000060  73 65 3a 09 41 20 70 72  6f 67 72 61 6d 20 6f 75  |se:.A program ou|
00000070  74 70 75 74 73 20 61 20  50 72 6f 2d 44 4f 53 20  |tputs a Pro-DOS |
00000080  67 72 65 65 74 69 6e 67  0a 2f 2f 0a 0a 23 69 6e  |greeting.//..#in|
00000090  63 6c 75 64 65 20 3c 69  6f 73 74 72 65 61 6d 3e  |clude <iostream>|
000000a0  0a 0a 75 73 69 6e 67 20  6e 61 6d 65 73 70 61 63  |..using namespac|
000000b0  65 20 73 74 64 3b 0a 0a  69 6e 74 20 6d 61 69 6e  |e std;..int main|
000000c0  28 29 20 7b 0a 20 20 63  6f 75 74 20 3c 3c 20 22  |() {.  cout << "|
000000d0  44 4f 53 20 52 6f 63 6b  73 20 74 68 65 20 62 69  |DOS Rocks the bi|
000000e0  74 7a 6f 6e 65 21 22 20  3c 3c 20 65 6e 64 6c 3b  |tzone!" << endl;|
000000f0  0a 7d 0a                                          |.}.|
000000f3
```

As you can see, hexdump shows every byte in the file by giving both its value (in hex) and its printed character representation (when it has one). Up to 16 bytes of the file are displayed per line; the first column shows the offset of each line from the beginning of the file. The last number shows the total number of bytes (0F3h, or 243 bytes). When a character has no printed representation, a period (.) is used. So, why would you use hexdump? Consider the result of compiling this program (using `c++ -o DosRocks DosRocks.cc`). The output is considerably longer (2360h bytes). An excerpt of the output of hexdump on the executable file DosRocks is:

```
000008a0  41 5e 41 5f c3 90 66 2e  0f 1f 84 00 00 00 00 00  |A^A_..f.........|
000008b0  f3 c3 00 00 48 83 ec 08  48 83 c4 08 c3 00 00 00  |....H...H.......|
000008c0  01 00 02 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000008d0  44 4f 53 20 52 6f 63 6b  73 20 74 68 65 20 62 69  |DOS Rocks the bi|
000008e0  74 7a 6f 6e 65 21 00 00  01 1b 03 3b 44 00 00 00  |tzone!.....;D...|
000008f0  07 00 00 00 58 fd ff ff  90 00 00 00 e8 fd ff ff  |....X...........|
00000900  60 00 00 00 d5 fe ff ff  b8 00 00 00 fc fe ff ff  |`...............|
00000910  d8 00 00 00 39 ff ff ff  f8 00 00 00 58 ff ff ff  |....9.......X...|
00000920  18 01 00 00 c8 ff ff ff  60 01 00 00 00 00 00 00  |........`.......|
```

This excerpt shows where the string `"DOS Rocks the bitzone!"` is stored in the file (along with a bunch of other binary data). You can even see the zero termination in the string.

What to do. Your task is to write a hexdump utility for MS-DOS that works in the same way as this command (same output, etc.). In particular, your utility should take the file name from the command line, and should print out identical format to `hexdump -C` (DOS has a more command so you can pipe it to that when you want. You are only to use DOS routines for this assignment. *You may not use any routines provided by the class link library.*

One idea is to design a procedure ProcessByte that accepts a single byte and does the processing necessary to produce the output desired, including actually producing that output every 16 bytes processed. There are many other good approaches, just make sure you pick one of them.

Your output must match the format shown, and you need to be able to handle files of arbitrary length up to $2^{32} - 1$ bytes. (That means you can't simply create a big buffer for the data and load it all before you start processing.) Also note that you will have to manage a DWORD (32-bit) offset, but using 16-bit registers.

Comment your design neatly. For each procedure you write, tell exactly what parameters it's expecting in which registers, describe the effects, and tell what registers will be modified, and how. Also, don't forget to comment your variables.

Write a comment near your main routine, in paragraph format, giving a description of the big picture: what was your approach, general algorithm, and so on. If there are any known problems, briefly describe them there too.

Make wise choices about data types and memory usage. You may use global variables, i.e. declarations in your data segment, in order to keep track of program state, but use them sparingly.

How to submit. Submit the file hexdump.asm using the standard course submission procedure below.

1. At the DOS prompt, remove the flash drive.
2. Reboot the computer into Windows (or your operating system)
3. Reinsert the flash drive.

4. Transfer the files to `gemini.cs.hamilton.edu` (if you don't know how to do this, you'll need to research it).
5. Log in to `gemini.cs.hamilton.edu`
6. `[user@gemini ~]$ cs240`
7. `[user@gemini ~]$ submit`

Submit will not be open until 24 hours before the assignment is due. You may submit as many times as you want up to the deadline. Your final submission will be used in grading.