COMP 2511

Assignment 2

1 Introduction

The purpose of this assignment is to write a program that "dumps" the content of a file in different formats — octal, hexadecimal, etc. By dumping a file, we mean that the encoding (ASCII encoding in the platforms we use in our labs) of each byte in the file is printed in an appropriate format (e.g. in hexadecimal).

We'll call our program `dump`. It is modelled after, but is much simpler than, the `hexdump` utility found in most Unix systems

2 Description

The following description of the `dump` program is based on the documentation of the `hexdump` utility:

NAME

dump - ascii, hexadecimal, octal dump

SYNOPSIS

dump [**-bcC**] [**-n**`length`] [**-s**`offset`] [`file`]

DESCRIPTION

The dump utility is a filter which displays the specified file, or the standard input, if no file is specified, in a user specified format.

The options are as follows:

-b One-byte octal display. Display the input offset in hexadecimal, followed by sixteen space-separated, three column, zero-filled, bytes of input data, in octal, per line.

-c One-byte character display. Display the input offset in hexadecimal, followed by sixteen space-separated, three column, space-filled, characters of input data per line.

-C Canonical hex+ASCII display. Display the input offset in hexadecimal, followed by sixteen space-separated, two column, hexadecimal bytes, followed by the same sixteen bytes in "character" format enclosed in '|' characters.

-n`length` Interpret only `length` bytes of input. Note that there is no space between **-n** and `length`.

-s`offset` Skip `offset` bytes from the beginning of the input. Note that there is no space between **-s** and `offset`.

The default is to interpret `length` & `offset` as decimal numbers. If they start with 0x (zero x) or 0X (zero X), they are interpreted as hexadecimal numbers; otherwise, if they start with 0 (zero), they are interpreted as octal numbers.

The default format, when neither **-b**, **-c** nor **-C** is specified, is to use one-byte hexadecimal display — the input offset is displayed in hexadecimal, followed by sixteen space-separated, two column, zero-filled, bytes of input data, in hexadecimal, per line.

3 Sample Output

To illustrate the different formats, we'll look at some sample output. As the output is fairly substantial, we'll omit part of it.

The following illustrates the default format:

```
$ dump hello.o
0000000 7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
0000010 01 00 03 00 01 00 00 00 00 00 00 00 00 00 00 00
0000020 c0 00 00 00 00 00 00 00 34 00 00 00 00 00 28 00
0000030 0a 00 07 00 55 89 e5 83 ec 08 83 e4 f0 b8 00 00
0000040 00 00 29 c4 83 ec 0c 68 00 00 00 00 e8 fc ff ff
... (40 lines omitted) ...
00002d0 0e 00 00 00 00 00 00 00 00 00 00 00 10 00 00 00
00002e0 00 68 65 6c 6c 6f 2e 63 00 6d 61 69 6e 00 70 72
00002f0 69 6e 74 66 00 00 00 00 14 00 00 00 01 05 00 00
0000300 19 00 00 00 02 08 00 00
0000308
```

In the default format, the offsets (the first "column" of numbers) are printed in hexadecimal in a minimum field width of 7 & padded by zeros. The offset is followed by 16 bytes (except for the last 2 lines), each printed as a 2-digit hexadecimal padded by zero. The columns of numbers are separated by columns of space characters of width 1. The penultimate line may have fewer than 16 bytes & the last line consists only of the offset. Note that hexadecimals are printed with lowercase 'a'–'f'. Note also that the last two lines are not padded by spaces at the end — a newline follows the last `00` on the penultimate line & the offset on the last line.

The following shows the **-b** option:

```
$ dump -b hello.o
0000000 177 105 114 106 001 001 001 000 000 000 000 000 000 000 000 000
0000010 001 000 003 000 001 000 000 000 000 000 000 000 000 000 000 000
0000020 300 000 000 000 000 000 000 000 064 000 000 000 000 000 050 000
0000030 012 000 007 000 125 211 345 203 354 010 203 344 360 270 000 000
0000040 000 000 051 304 203 354 014 150 000 000 000 000 350 374 377 377
... (40 lines omitted) ...
00002d0 016 000 000 000 000 000 000 000 000 000 000 000 020 000 000 000
00002e0 000 150 145 154 154 157 056 143 000 155 141 151 156 000 160 162
00002f0 151 156 164 146 000 000 000 000 024 000 000 000 001 005 000 000
0000300 031 000 000 000 002 010 000 000
0000308
```

The output format is similar to the default format except that each byte is printed as a 3-digit octal padded by zeros.

The following shows the **-c** option:

```
$ dump -c hello.o
0000000 177   E   L   F 001 001 001  \0  \0  \0  \0  \0  \0  \0  \0  \0
0000010 001  \0 003  \0 001  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
0000020 300  \0  \0  \0  \0  \0  \0  \0   4  \0  \0  \0  \0  \0   (  \0
0000030  \n  \0  \a  \0   U 211 345 203 354  \b 203 344 360 270  \0  \0
0000040  \0  \0   ) 304 203 354  \f   h  \0  \0  \0  \0 350 374 377 377
... (40 lines omitted) ...
00002d0 016  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0 020  \0  \0  \0
00002e0  \0   h   e   l   l   o   .   c  \0   m   a   i   n  \0   p   r
00002f0   i   n   t   f  \0  \0  \0  \0 024  \0  \0  \0 001 005  \0  \0
```

```
0000300 031  \0  \0  \0 002  \b  \0  \0
0000308
```

The offset is the same as in the default format. Each byte is printed in a width of 3. Printable characters (as determined by the C `isprint` function) are printed as themselves padded by spaces. The 8 escape characters, '\0', '\a', '\b', '\f', '\n', '\r', '\t', and '\v', are each printed as 2 characters — a backslash followed by the appropriate character — padded by a space. All other characters are printed as 3-digit octals padded by zeros. As before, the columns are separated by columns of space characters of width 1.

The following shows the **-C** option. The output format is somewhat different from the other cases.

```
$ dump -C hello.o
00000000  7f 45 4c 46 01 01 01 00  00 00 00 00 00 00 00 00  |.ELF............|
00000010  01 00 03 00 01 00 00 00  00 00 00 00 00 00 00 00  |................|
00000020  c0 00 00 00 00 00 00 00  34 00 00 00 00 00 28 00  |........4.....(.|
00000030  0a 00 07 00 55 89 e5 83  ec 08 83 e4 f0 b8 00 00  |....U...........|
00000040  00 00 29 c4 83 ec 0c 68  00 00 00 00 e8 fc ff ff  |..)....h........|
... (40 lines omitted) ...
000002d0  0e 00 00 00 00 00 00 00  00 00 00 00 10 00 00 00  |................|
000002e0  00 68 65 6c 6c 6f 2e 63  00 6d 61 69 6e 00 70 72  |.hello.c.main.pr|
000002f0  69 6e 74 66 00 00 00 00  14 00 00 00 01 05 00 00  |intf............|
00000300  19 00 00 00 02 08 00 00                           |........|
00000308
```

Except for the last 2 lines, each line consists of:

- the offset printed as hexadecimal in a minimum width of 8 & padded by zeros (note the width of 8); followed by
- 2 spaces; followed by
- 8 bytes printed as 2-digit hexadecimals padded by zeros & separated by spaces of width 1; followed by
- 2 spaces; followed by
- another 8 bytes printed in a similar format as the previous 8 bytes; followed by
- 2 spaces; followed by
- the 16 "characters" corresponding to the 16 bytes enclosed by vertical bars (|) — printable characters (as determined by the `isprint` function) are printed as themselves; non-printable characters are printed as dots (.)

Note again that the last two lines are not padded by spaces at the end — in particular, in the previous output, a newline follows the last vertical bar on the penultimate line.

As a final example, we show an invocation of `dump` that specifies both the number of bytes to skip & the number of bytes to dump:

```
$ dump -c -s48 -n100 hello.o
0000030  \n  \0  \a  \0   U 211 345 203 354  \b 203 344 360 270  \0  \0
0000040  \0  \0   ) 304 203 354  \f   h  \0  \0  \0  \0 350 374 377 377
0000050 377 203 304 020 270  \0  \0  \0  \0 311 303  \0   h   e   l   l
0000060   o   ,       w   o   r   l   d   !  \n  \0  \0   G   C   C   :
0000070       (   G   N   U   )       3   .   2   .   3  \0  \0   .   s
0000080   y   m   t   a   b  \0   .   s   t   r   t   a   b  \0   .   s
0000090   h   s   t   r
0000094
```

It is important that the output of your program adheres to the specification given above as we'll be comparing your output to the expected output using a file comparison program. A sample input file together with sample output files will be provided.

4 Command-line Validation

Command-line arguments need to be validated. Following the program name, any command-line argument that starts with the character '`-`' is regarded as an option until either

- an argument that does not start with a hyphen (`-`) is encountered (such an argument is regarded as the input file name); or
- the special option "`--`" (without the quotes) is encountered, in which case what follows, if anything, is the input file name

Besides "`--`" (which is used to mark the end of options), only the options **-b**, **-c**, **-C**, **-n** and **-s** are valid. Furthermore, both **-n** and **-s** must be immediately followed by a non-negative integer (with no spaces in between). If an invalid option is encountered, the program should exit after printing an error message indicating the correct usage. Also, the user can specify at most one input file; if more than one is specified, the program should again exit with an error message. (Note that if no input file is specified, standard input is used.)

If conflicting valid options are specified, those that occur further along the command-line take precedence. For example,

`dump -c -C -b file` should dump `file` in octal format

`dump -s100 -b -s50 file` should skip 50 bytes before dumping

5 Additional Information

There are a number of "special" cases.

- if the file to dump has size 0, there should be no output
- if the number of bytes to skip is more than the number of bytes in the file, there should be no output
- when the requested number of bytes to dump exceeds the number of bytes available in the file, dumping should stop at the end of the file

6 Submission & Grading

This assignment is due at noon, Monday, July 9, 2012. Submit a zip file to `In` in the directory:

```
\COMP\2511\a2\
```

Your zip file must be named `<name_id>.zip`, where `<name_id>` is your name & student IO separated by an underscore (for example, `SimpsonHomer_a12345678.zip`). Do not use spaces to separate your last & first names. Your zip file must unzip directly to your source file(s) without creating any directory. (For this assignment, only one source file is needed. However, you may use multiple source files if you so choose.)

If you need to submit more than one version, name the zip file of each later version with a version number after your name, e.g., `SimpsonHomer_v2.zip`. We'll only mark the version with the highest version number. (There must be exactly one zip file with the highest version number.)

Do not submit rar files. They will not be marked.

If you need to submit more than one version, name the zip file of each later version with a version number after the ID, e.g., `SimpsonHomer_a12345678_v2.zip`. If more than one version is submitted, we'll only mark the file(s) in the version with the highest version number. (There must be exactly one zip file with the highest version number; otherwise, your submission will not be marked & you will get no credit for the assignment.)

The restriction to ANSI C (C89) & its standard library applies as in the previous assignment. In particular, your program must compile without errors or warnings under GCC with the following switches:

```
-ansi -W -Wall -pedantic
```

If your program does not meet these requirements, you may receive a score of 0 for the assignment. Otherwise the grade breakdown for this assignment is approximately as follows:

Design & code clarity	10%
Command-line handling	10%
Default & octal (**-b**) formats	20%
Character (**-c**) format	20%
Canonical (**-C**) format	25%
Skipping & dumping specified number of bytes	15%