CS 241 Data Organization using C
Parsing Command Line Arguments, Error Checking, and Conversion between Binary and Decimal

Fall 2014

1 Reading Command-Line Arguments in a C Program

Many C programs created and used in both industry and in academia are designed with a command-line human interface. Kernighan and Ritchie provide a description of how to program this in section 5.10: Command-line Arguments.

2 Requirements

Write a C program, `binary.c`, with the usage:

```
usage:
binary SIZE OPTION NUMBER
  SIZE:
    -8   input is an unsigned 8-bit integer.
    -16   input is an unsigned 16-bit integer.
    -32   input is an unsigned 32-bit integer.
    -64   input is an unsigned 64-bit integer.

  OPTION:
    -b   NUMBER is binary and output will be in decimal.
    -d   NUMBER is decimal and output will be in binary.

  NUMBER:
    number to be converted.
```

2.1 Errors

When bad input is given, your program must print the appropriate error message followed by the usage message shown above. (See the example output for the possible error messages.)

2.2 Output Format for Binary Numbers

Every 4 binary numerals must be separated by a space. All leading zeros must be displayed. Thus, an 8 bit integer must display 8 bits and a 16 bit integer must display 16 bits.

Command	Output
`./binary -8 -d 63`	`0011 1111`
`./binary -16 -d 63`	`0000 0000 0011 1111`

If a number is too large to be represented in the specified number of bits, then output only the lower bits of that number. For example, the number 259 is $2^8 + 2^1 + 2^0$; however, output as an 8-bit integer drops the 2^8 place:

Command	Output
`./binary -8 -d 259`	`0000 0011`
`./binary -16 -d 259`	`0000 0001 0000 0011`

The modulus operator is useful in deciding when to leave a space.

2.3 Output Format for Decimal Numbers

Decimal numbers must be formatted with comma thousand separators and with leading spaces (not leading zeros). For example, the largest, unsigned 8-bit number is 11111111 (base 2) = 255 (base 10). Since 255 has three digits, then all 8-bit numbers should be printed right justified within a field of 3 characters:

Command	Output
`./binary -8 -b 11111111`	`255`
`./binary -8 -b 1111`	`15`
`./binary -8 -b 11`	`3`

Similarly, the largest unsigned 16 bit number (1111111111111111_2) is 65,535 in base 10. This contains 5 digits plus a comma. Thus, all 16-bit decimal numbers must be right justified in a field of 6 characters:

Command	Output
`./binary -16 -b 1111111111111111`	`65,535`
`./binary -16 -b 11111111`	`255`
`./binary -16 -b 1111`	`15`
`./binary -16 -b 11`	`3`

WARNING: ANCI C does not include a format option for automatically adding comma thousand separators nor does the gcc used on `moons.unm.edu`. This format option is standard in C++ and in many expended versions of C. If you are writing your code using a different C compiler and turn in code with such an option specified, your program will most likely not compile when tested and you will get a zero for the assignment.

It is not that hard to write your own code for outputting numbers with comma thousand separators. However, this is such a common task that a quick Google search should return a solution in ANCI C. It is ok for you to use such code. *If you do use any code you did not write, be sure to avoid plagiarism by giving proper credit for the source.*

Hint: You might find it helpful to use the `strtol` function of `<stdlib.h>`.

3 Grading Rubric (total of 25 points)

-2 point : The program does not start with a comment stating the students first and last name and/or the source file is not named correctly.

-2 points : Program compiles with warnings on moons.cs.unm.edu using `/usr/bin/gcc` with no options (except any standard libraries you may want to use such as `-lm` for `math.h`). If you do use a standard library that requires a linking option, please include this as a note when you submit in Learn.

2 points : All code follows the CS-241 standards (including comments).

23 points : Passes 23 known test cases: one point each. The test file is `binary.cmd` and the output file is `binary.out`. Note, unlike past test files, this test file is not a data file to be input into your program by redirecting the standard input stream. The file `binary.cmd` is a Linux shell executable file (that just means it is a text file where each record is a Linux command). Note: for this shell file to run, it must have execution permission turned on:

```
chmod u+x binary.cmd
```

Also, the script expects your executable to be named `binary` (not `a.out`), so make sure you specify that when you compile.

```
gcc -o binary binary.c
```

I've provided you with a simple makefile that you may find helpful.