CSDS 325/425: Computer Networks
Project #1

Due: Sept. 17, 11:59 PM

The first project of the semester is to write a simple IPv4 address validator. Your program will read each line from a given file and determine whether the line represents a well-formed IPv4 address or not. The aim of this project is to get your feet wet with writing C/C++ code and get used to the class development environment.

Overview

The usage of your program—which will be called `proj1`—is as follows:

```
./proj1 [-s] [-l] -f filename
```

Specifically:

- When "-s" is given the program operates in *summary mode*. See below for details.
- When "-l" is given the program operates in *list mode*. See below for details.
- The user must give either "-s" or "-l", but not both.
- The "-f" option followed by a filename must be given. This provides the name of a file that contains a list of IP addresses to validate. See below for details.
- The command line arguments may appear in any order.
- Unknown command line arguments must trigger meaningful error messages.
- When encountering an error your program will print a meaningful error message and terminate.

File Format

Each line in the text file will be validated as an IPv4 address (see format below). Make no assumptions about what the line may contain or how many lines will be in each file.

Valid IP Addresses

IPv4 addresses follow a "dotted-quad" notation. E.g., "192.5.110.90". Specifically:

- Each IP address is made up of four integer numbers (the "quads").
- The quads are separated by periods (".").
- There will be four quads and three periods in each valid IPv4 address.
- Each IP address begins with a quad.
- A quad can be any one digit number (0-9).
- A quad can be any two digit number (10-99).
- A quad can be any three digit number in the range 100-255 (inclusive).
- A quad cannot have more then three digits.
- A quad cannot begin with a zero ("0"), unless the entire quad is the single digit zero. I.e., zero padding on the left of the quads is not allowed.

Summary Mode

When your program is run with the "-s" option, it must operate in *summary mode*. In this mode you will report the total number of lines you read from the file, the number of valid IPv4 addresses in the file and the number of invalid IPv4 addresses in the file. This information will be printed to the screen after the entire file is checked. The output will be three lines printed to the screen with the following format:

```
LINES: xxx
VALID: yyy
INVALID: zzz
```

The labels must appear as they do above. I.e., at the beginning of the line, capitalized, and followed by a colon (":") and a single space. The "`xxx`" value is the number of lines in the file. The "`yyy`" value is the number of valid IPv4 addresses in the file. And, the "`zzz`" value is the number of invalid IPv4 addresses in the file. The sum of `yyy` and `zzz` will be `xxx`.

Examples:

```
% ./proj1 -s -f test-file-1.txt
LINES: 10
VALID: 3
INVALID: 7

% ./proj1 -f test-file-2.txt -s
LINES: 1234
VALID: 1035
INVALID: 199

% ./proj1 -s -f /dev/null
LINES: 0
VALID: 0
INVALID: 0
```

Formatting notes:

- *Nothing* else may appear in the output.
- The numbers must be decimal numbers with no zero-padding.
- There should be a single newline after each of the three output lines.
- There should be no extra whitespace at the beginning, end or in the middle of the output lines.

List Mode

When your program is run with the "-l" (lower case L) option, it must operate in *list mode*. In this mode your program will print every line in the input file with a trailing annotation as to whether the line is a valid IPv4 address or not. The output will be printed to the screen and include one line for every line in the input file following format:

```
input_line_1 C
input_line_2 C
input_line_3 C
input_line_4 C
```

Each "input_line" is the line read from the file *without* the trailing newline character. The "C" will be a plus ("+") if the input line is a valid IPv4 address or a minus ("-") if the input line is not a valid IPv4 address. A single newline will follow the plus or minus.

Example:

```
% ./proj1 -l -f test-file-3.txt
132.235.1.2 +
06.14.15.16 -
1.2.3 -
1.2.3.Mark -
1.2.3.4 +
.1.2.3.4 -
10.0.0.200 +
10.0.0.259 -
1..2.34 -
```

Formatting notes:

- *Nothing* else may appear in the output.
- The input line should appear exactly as it does in the input file except for the removal of the trailing newline.
- There should be a single space between the line read from the input file and the classification character ("+" or "-").
- There should be no extra whitespace at the beginning, end or in the middle of the output lines.

Hints

- Strings in C and C++ are tedious. A set of standard string processing routines helps (a little!). Use the manual pages to look for routines such as *strsep()*, *isdigit()* and *strlen()*.
- There is a sample C program that uses *getopt()* to parse command line arguments on the class web page.
- Packaged with the sample C program is a Makefile that can be readily adapted for this project.

Final Bits

1. Submission specifications:
   (a) All project files must be submitted to Canvas in a gzip-ed tar file called "[CaseID]-proj1.tar.gz" (without the brackets).
   (b) Your submission must contain all code and a Makefile that by default produces an executable called "`proj1`" (i.e., when typing "make").
   (c) Do not include executables or object files in the tarball.
   (d) Do not include sample input or output files in your tarball.
   (e) Do not include multiple versions of your program in your submission.
   (f) Do not include directories in your tarball.
   (g) Do not use spaces in file names.
   (h) Every source file must contain a header comment that includes (*i*) your name, (*ii*) your Case network ID, (*iii*) the filename, (*iv*) the date created and (*v*) a brief description of the code contained in the file.
2. You may not leverage third-party libraries for this project. The standard C library and the C++ STL are fine. Projects that rely on extra libraries or tools will be returned ungraded.
3. Your submission may include a "notes.txt" file for any information you wish to convey during the grading process. We will review the contents of this file, but not of arbitrary files in your tarball (e.g., "readme.txt").
4. There will be sample reference output on the class web page by the end of the day on September 3. (Not all sample input will have (all) corresponding reference output.)
5. Print only what is described above. Extra debugging information must not be included. Adding an extra option (e.g., "-v" for verbose mode) to dump debugging information is always fine.
6. Do not make assumptions about the size of the input.
7. If you encounter or envision a situation not well described in this assignment, please Do Something Reasonable in your code and include an explanation in the "notes.txt" file in your submission. If you'd like to ensure you're on the right track, please feel free to discuss these situations with me.
8. Hints / tips:
   (a) Needlessly reserving large amounts of memory in case it may be needed is unreasonable.
   (b) Errors will be thrown at your project.
   (c) A simple C program and Makefile are available on the class web page. The program illustrates the use of *getopt()* to parse the command line arguments. The Makefile can be easily adapted for this project.
9. *WHEN STUCK, ASK QUESTIONS!*