Total Points: 100

COP4342 – UNIX Tools

Assignment #1: Custom Simplified *Test*

Instructor: Dr. Robert Van Engelen   Teaching Assistant: Imran Chowdhury   Spring 2018

Description: The test command line utility on UNIX and UNIX-based platforms checks a condition and exits with zero if the condition succeeds or 1 if the condition fails:

test EXPRESSION

or the shorthand

[EXPRESSION]

Assignment: In this programming assignment, you will familiarize yourself with the Unix test command utility and then implement a customized simplified version which we will name *cstest*.

Command line options: In our cstest utility EXPRESSION takes the following forms:

-N *EXPRESSION*	*Negate EXPRESSION (true->false, false->true)*
-d *FILE*	*FILE* exists and is a directory
-e *FILE*	*FILE* exists
-f *FILE*	*FILE* exists and is a regular file
-r *FILE*	*FILE* exists and read permission is granted
-w *FILE*	*FILE exists and write permission is granted*
-x *FILE*	*FILE exists and execute permission is granted*
FILE1 **-n** *FILE2*	*FILE1* is newer (modification date) than *FILE2*
-z *VALUE*	*VALUE is "0" or an empty string*
EXPRESSION1 EXPRESSION2	*EXPRESSION1 and EXPRESSION2 are tested*

When multiple conditions are given as shown in the last row of the table above then all must succeed, for example:

```
./cstest -f somefile -r somefile -w somefile
```

This example tests if somefile is a file that is readable and writeable.

Note that -N is an option that takes no option argument. This option indicates that the next EXPRESSION's logic is inverted, for example:

```
./cstest -f somefile -N -r somefile -N -w somefile -x somefile
```

This example tests if somefile is a file that executable but is not readable and not writeable. Note that -N applies to a single condition.

Exit code: the cstest utility should exit with EXIT_SUCCESS if all conditions succeed and EXIT_FAILURE otherwise. If no conditions to cstest are given then cstest should exit successfully (i.e. empty conditions are true).

To parse EXPRESSION you should use getopt(3), see the man page of getopt(3). The getopt function parses the command line options using a specified option string.

Sample C code: Use the following examples to study and implement options and arguments parsing. The example takes option -N without option argument and option -f with a FILE argument. It also parses arguments that are not options, when placed between groups of options. You will need this to handle argument *FILE1* in *FILE1* **-n** *FILE2*.

Example getopt() with GNU extension (the use of + in option strings) under Linux:

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
void usage()
{
  fprintf(stderr, "Usage: cstest EXPRESSION\n");
  exit(EXIT_FAILURE);
}
int main(int argc, char **argv)
{
  int Nflag, ch;
  Nflag = 0;
  while (1) {
    while ((ch = getopt(argc, argv, "+Nf:")) != -1) {
      switch (ch) {
        case 'N':
          Nflag = 1;
          break;
        case 'f':
          printf("-f %s\n", optarg); // print (as an example)
          // handle option -f here
          break;
        case ':':
          fprintf(stderr, "option -%c requires an argument", optopt);
          usage();
          break;
        case '?':
        default:
          fprintf(stderr, "unrecognized option -%c\n", optopt);
          usage();
      }
    }
    if (optind >= argc)
      break;
    printf("argument %s\n", argv[optind]); // print (as an example)
    // handle argument here
    ++optind; // next argument
  }
  return 0;
}
```

Example getopt() with Mac OS X:

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
void usage()
{
  fprintf(stderr, "Usage: cstest EXPRESSION\n");
  exit(EXIT_FAILURE);
}
int main(int argc, char **argv)
{
  int Nflag, ch;
  Nflag = 0;
  while (1) {
    while ((ch = getopt(argc, argv, "Nf:")) != -1) {
      switch (ch) {
        case 'N':
          Nflag = 1;
          break;
        case 'f':
          printf("-f %s\n", optarg); // print (as an example)
          // handle option -f here
          break;
        case ':':
          fprintf(stderr, "option -%c requires an argument", optopt);
          usage();
          break;
        case '?':
        default:
          fprintf(stderr, "unrecognized option -%c\n", optopt);
          usage();
      }
    }
    if (optind >= argc)
      break;
    printf("argument %s\n", argv[optind]); // print (as an example)
    // handle argument here
    ++optind; // next argument
    optreset = 1; // reset to start parsing next group of options
  }
  return 0;
}
```

Your version of cstest should not print anything to standard output (errors go to standard error). To test the attributes of a file, use stat(2) which populates a structure named ‘stat’ with file modes and modification time. You can use S_ISREG(m) to test the mode bits m to determine if the file is a regular file. You can use other macros for other purposes. See the stat(2) manual page for details.

Submission Requirements: A tar file named ‘yourCSusername_asg1.tar’ containing the following files:-

[1]. The C or C++ implementation of the above described *custom test* utility *cstest*. Do not include object files or binaries.

Grading Policy: Submitted code must compile with the ‘-Wall -ansi -pedantic’ flags without any warning messages to conform to the ANSI ISO standard. Code that does not compile or generates errors (Segmentation Fault etc.) on execution will receive zero grade points. Points distribution :-

[1]. Code Readability	(20 points)
[2]. Test Cases	(80 points)

Individual parts of the implementation will not be graded separately for correctness. There will be several cases to test the implementation logic as a whole. Also, keep in mind that your code will be tested on *linprog*. Students should test their code thoroughly on the *linprog* server before submission

Miscellaneous:

Honor code policy will be strictly enforced. Write the code by yourself and protect your submission.

Key Concepts: *System Calls, File Meta-Data, String Operations, Command Options Parsing.*

UNIX API Calls: *stat(), getopt().*