CSCI 4963/6963 — Large-Scale Programming and Testing
Homework 1 (document version 1.0)
Regular Expressions and Pattern Matching in C

Overview

- This homework is due by 11:59:59 PM on Tuesday, September 19, 2017.
- This homework will count as 10% of your final course grade.
- This homework is to be completed **individually**. Do not share your code with anyone else.
- You **must** use C for this homework assignment, and your code **must** successfully compile via `gcc` with absolutely no warning messages when the `-Wall` (i.e., warn all) compiler option is used. We will also use `-Werror`, which will treat all warnings as critical errors.
- Your code **must** successfully compile and run on Submitty, which uses Ubuntu v16.04.3 LTS. Note that the `gcc` compiler is version 5.4.0 (`Ubuntu 5.4.0-6ubuntu1~16.04.4`).

Homework Specifications

In this first homework, you will use C to implement a rudimentary `grep` tool that supports line-based regular expressions and simple pattern matching. The goals here are to become more comfortable programming in C on Linux and write high-quality and easily maintainable code.

In brief, your program must apply a regular expression to a given input file, displaying all lines of the input file that match the given regular expression.

Command-Line Arguments

A text file containing the regular expression is specified on the command-line as the first argument. The input file is then specified as the second argument.

As an initial example, you could execute your program and have it apply the regular expression specified in `regex.txt` to input file `hw1-input01.txt` as follows:

```
bash$ ./a.out regex.txt hw1-input01.txt
```

Specifying Regular Expressions

For this assignment, we will implement a subset of general regular expression syntax that focuses on finding substrings.

A regular expression (regex) is a string used as a pattern for matching. The regex may contain characters that must match exactly; for example, if the pattern is `"printf"`, then it will only match exact substrings (i.e., all lines containing `"printf"`).

In addition to this simple substring matching, there are other special characters that enable patterns of characters to be matched.

The specific individual character regex syntax that must be supported is:

```
.     match any character
\d    match any digit [0-9]
\w    match any letter [a-zA-Z]
\s    match any whitespace character (space, tab '\t', newline '\n)
```

Further, repetition may be specified by appending the following to any individual character or regex symbol shown above.

```
?     match zero or one instances
+     match one or more instances
*     match zero or more instances
```

Some example regular expressions to support (and test for) are:

```
printf                match "printf"
colou?r               match either "color" or "colour"
\w+                   match a word consisting of one or more alpha characters
<\w+>                 match a tag, such as "<title>" or "<body>"
\w+\d*@rpi.edu        match an RPI email address by matching one or more alpha
                       characters, followed by zero or more digits, then "@rpi.edu"
                         (though note that the "." here matches any character...)
\d\d\d-\d\d\d\d       match a phone number, such as "276-2819"
if\s*(.+)             match an if statement (to some degree) in C/C++/Java
```

Note that for both `"*"` and `"+"`, use a greedy approach, meaning that you should match as many characters as possible. As an example, the `"if\s*(.+)"` pattern above should match through to the rightmost closing parenthesis (if present).

Ignore the need (or temptation) to add more regex syntax here! While regex matching can be rather complex, especially as the regex syntax is expanded, keep your implementation as simple as possible. Start by focusing on the examples above.

Implementation Details

For your implementation, consider using `fopen()`, `fscanf()`, `fgets()`, `fclose()`, `isalpha()`, `isdigit()`, `isspace()`, and other such string and character functions here. Be sure to check out the details of each function by reviewing the corresponding `man` pages.

If improper command-line arguments are given, report an error message to `stderr` and abort further program execution. In general, if an error is encountered, display a meaningful error message on `stderr` by using either `perror()` or `fprintf()`, then aborting further program execution.

Error messages must be one line only and use the following format:

```
ERROR: <error-text-here>
```

Assumptions

For this initial implementation, you can make the following assumptions:

1. Assume that the maximum line length for any input file is `256` characters.
2. Assume that the regex file contains exactly one non-empty line.
3. Assume that the regex is correct in terms of syntax.

Required Output

When you execute your program, unless errors occur, your program must display all lines from the given input file that match the given regular expression.

As an example, consider the following `hw1-input01.txt` file:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main()
{
  int x = 300-1200;
  printf( "I like Cocoa Puffs\n" );
  printf( "and also Trix %d\n", x );
  printf( "Green is my favorite color\n" );
  printf( "My number is 276-2819\n" );
  return EXIT_SUCCESS;
}
```

Executing your program as follows with the regex `"printf"` must display the output shown below:

```
bash$ ./a.out regex.txt hw1-input01.txt
  printf( "I like Cocoa Puffs\n" );
  printf( "and also Trix %d\n", x );
  printf( "Green is my favorite color\n" );
  printf( "My number is 276-2819\n" );
```

Executing your program as follows with the regex `"colou?r"` must display the output shown below:

```
bash$ ./a.out regex.txt hw1-input01.txt
  printf( "Green is my favorite color\n" );
```

Executing your program as follows with the regex `"\d\d\d-\d\d\d\d"` must display the output shown below:

```
bash$ ./a.out regex.txt hw1-input01.txt
  int x = 300-1200;
  printf( "My number is 276-2819\n" );
```

Submission Instructions

To submit your assignment (and also perform final testing of your code), please use Submitty, the homework submission server. The specific URL is on the course website.

Note that this assignment will be available on Submitty a few days before the due date. Please do not ask on Piazza when Submitty will be available, as you should perform adequate testing on your own.

That said, to make sure that your program does execute properly everywhere, including Submitty, read the following tip.

Output to standard output (`stdout`) is buffered. To ensure buffered output is properly flushed to a file for grading on Submitty, use `fflush()` after every set of `printf()` statements, as follows:

```
printf( ... );     /* print something out to stdout */
fflush( stdout );  /* make sure that the output is sent to a */
                   /*  redirected output file, if specified  */
```