Assignment 2 - Regular Expression Matcher

Add more metacharacters to the Rob Pike regular expression matching code.

You can read about the regular expression matching code here:
https://www.cs.princeton.edu/courses/archive/spr09/cos333/beautiful.html

1. Starting Code

Start with the regular expression matching code. The match() function accepts a string containing the regular expression and a string to search.
It returns 1 when the regex matches to the string and 0 when there is no match.

The starting code supports the following metacharacters:

*	zero or more of the previous character
.	any single character
^	start of a line
$	end of line

and literal characters within the regular expression.

Add the functionality for the following metacharacters to the code:

+	one or more of the previous character
?	the previous character can occur 0 or 1 times
[ ]	a set of characters within the brackets
\	escape character, the character that follows the backslash is a literal

The sets can also have a range of characters within them using the dash.
  -alphabetics ranges are two letters which denote the start and end of the
   range with a dash between them e.g. [a-z], [A-Z], [a-zA-Z], [a-g]
  -numeric ranges are two numbers separated by a dash.
   e.g. [0-9],[0-3]
  -sets can contain multiple ranges, e.g. [a-zA-Z0-9]

The code currently only returns success or failure. Add the functionality so
that the location in the string which the regular expression matches will be
printed to the screen. The first character of the string is 0.

The current code will only match the first occurrence of the regular expression
in the string. Add the functionality so that it will search the entire string and find all matching substrings. The matching substrings should not overlap.

2. Input
The input will be read from a file that is specified on the command line.
The first line of the file will be a string containing the regular expression
and the second line will be the string which the regex will matched against.

The file name will be entered on the command line. There are no command line
arguments so the program will be called with ./a2 filename.

3. Output
The output should print either match or no match. If the regular expression
finds a match the string then it should also print out the location in the string where the match occurs.

For example:
  regex = "b*"
  string = "abc"
the output would be:
  match 1
because b* matches the substring "b" in the string.

For:
  regex = "d*"
  string = "abc"

the output would be:
```
  no match
```

For:
```
  regex = "[bd]"
  string = "abcdefg"
```
the output would be:
```
  match 1 3
```

Coding Practices
Write the code using standard stylistic practices. Use functions, reasonable variable names, and consistent indentation.
If the code is difficult for the TA to understand then you will lose marks.

As usual, keep backups of your work using source control software.

Submitting the Assignment
Submit the assignment using Courselink. Submit only the source code, the makefile, and readme. Bundle the code in a tar file and name it a2.tar.

The assignments will be marked on the linux.socs.uoguelph.ca server. If you develop your code on a different platform then it is a good idea to put the #include files in an #ifdef for that system so they will still compile on the server. Test your program on the server before you submit it.

Include a readme file which contains your name and student number.

The TA will unpack your code and type "make". They will then try to run executables named a2 from the current directory. Several different input files will be used for testing. If the makefile is missing, the make command or executing the programs do not work then you will lose
a substantial number of marks.

It is always a good idea to unpack and test the file you are submitting to be sure that what you submit actually compiles.