Project 2
CS221: C and Systems Programming – Fall 2018

Deadline: November 12, 2018 at 11:59pm

Restricted grep (rgrep)

`grep` is a UNIX utility that is used to search for patterns in text files. It's a powerful and versatile tool, and in this project you will implement a version that, while simplified, should still be useful.[1]

Your project is to complete the implementation of `rgrep`, our simplified, restricted `grep`. `rgrep` is "restricted" in the sense that the patterns it matches only support a few regular operators (the easier ones). The way `rgrep` is used is that a pattern is specified on the command line. `rgrep` then reads lines from its standard input and prints them out on its standard output if and only if the pattern "matches" the line. For example, we can use `rgrep` to search for lines that contain text file names that are at least 3 characters long (plus the extension) in a file like the following:

`#` so you can see what lines are in the file:

```
$ cat testin

a.out
cs221.txt
cs221.pdf
usf.txt
nope.pdf
.txt

$ ./rgrep ‘.\.txt’ < testin

cs221.txt
usf.txt
```

What's going on here? `rgrep` was given the pattern `".\.txt"`; it printed only the lines from its standard input that matched this pattern. How can you tell if a line matches the pattern? A line matches a pattern if the pattern "appears" somewhere inside the line. In the absence of any special operators, seeing if a line matches a pattern reduces to seeing if the pattern occurs as a substring anywhere in the line. So for most characters, their meaning in a pattern is just to match themselves in the target string. However, there are a few special clauses you must implement:

.(period)	Matches any character
+(plus sign)	The preceding character may appear 1 or more times (in other words, the preceding character can be repeated several times in a row).
?(question mark)	The preceding character may appear between 0 and 1 times (in other words, the preceding character is optional).
\\(backslash)	"Escapes" the following character, nullifying any special meaning it has.

So, here are some examples of patterns and the kinds of lines they match.

(	An open parenthesis must appear somewhere in the line.
hey+	Matches a line that contains the string "hey" followed by any number (0 or more) of y's.
str?ing	Matches lines that contain the substrings "string" or "sting", since the "r" is optional..
z.z\\.txt	Matches lines that contain the substring "zaz.txt", "zbz.txt", etc., where the character between the z's can be anything, including a period.

[1]Type `man grep` in terminal for more detailed information on how grep works.

These are the only special characters you have to handle. With the exception of the null char that terminates a string, you should not have to handle any other character in any special way. You may assume that your code will not be run against patterns that don't make sense. You must follow the spec strictly - so you should neither include any library other than those specified in the skeleton, nor copy and paste code from other libraries. You may not use any code you find online.

Your `rgrep` does not need to support the following patterns:

- Operators ?,.,+ immediately follow one another (e.g., '.+').
- Same letter occurs before and after + and ? operators (e.g., 'a+a', 'b?b').
- Escape operator is the last character in the pattern (e.g., 'abc\').

Getting started
Download the skeleton code from the course webpage.
To compile, type:

```
$ make
```

To run against a particular pattern, use

```
$ ./rgrep pattern
```

The skeleton code handles reading lines from standard input and printing them out for you; you must implement the function `int rgrep_matches(char *line, char *pattern)` in `matcher.c`, which returns true if and only if the string contains the pattern. You may also choose to implement matches_leading, which is also in `matcher.c`, to guide your submission, though this is optional. You may add your helper functions in `matcher.c`, but **you should not modify any other files**. You will only submit your `matcher.c` file, so if you modify any other files, we wont be even looking at them.

Testing

```
$ make check
```

Note that this doesn't mean your solution will receive full points, since we will be running a much larger suite of test cases. Testing your code is part of your grade, so you should test your code to make sure that it properly matches lines against patterns. One way to do this is to create a text file with the lines you want to test against, say test_input.txt and then verify that running `./rgrep pattern < test_input.txt` prints only the lines that you think should match the pattern, and no others. Note that the terminal might interpret the backslash operator for you, which is not what you want. For example, when you type at your shell

```
$ ./rgrep \.hi < input.txt
```

your program might get the pattern ".hi" because the shell interpreted the backslash before it got passed to your program. The solution is to put the pattern in single quotes, so what you want to type is:

```
$ ./rgrep '\.hi' < input.txt
```

This should ensure that your pattern operators aren't expanded or consumed by the shell in the terminal.

You are encouraged to share your test cases in Piazza.

Grading
Your code must compile and run correctly on the Linux lab machines. If we cannot compile your code on the lab machines, you will receive no credit.

Feature	Points
Patterns without special characters	30
Patterns with periods	30
Patterns with plus signs	15
Patterns with question marks	15
Patterns with periods, plus signs, question marks, and backslashes	10

Submission Guidelines:
Prior to the deadline, upload one zip file containing only two files: your matcher.c source code, and status.txt, to Canvas. The ZIP file name must be in the following format: LastName_FirstName_StudentID_proj2.zip For instance if my student ID is 123456789 and I am submitting my solution for project 2, then I am going to compress `status.txt` and `matcher.c`, and rename the zip file to: Pournaghshband_Vahab_123456789_proj2.zip Do not submit/include any other file such as the executable file. A good sanity check is to check your zip file for corruption by extracting (unzipping) it and testing whether it did compress it successfully. If we cannot unzip your submission, you will receive no credit.

```
Sample status.txt file:
Vahab Pournaghshband - Project 2 The program works as required. It compiles/runs and the
output matches the correct format to the letter. However, the style and formatting is incorrect
because I DIDNT comment it (didnt even put my name in the file).
```