Project 1

Overview

You may work with a partner for this project. Each of you will submit the following:

- Completed rgrep.c
- Any test files you used to check if your solution is correct
- Name of your partner in the submission box (if any)

Background

`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 assignment you will implement a version that, while simplified, should still be useful.

Your assignment 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 simplified 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 3-5 characters long (plus the extension) with the following:

```
$ cat testin #so you can see what lines are in the file
this file is fine.txt
the filename s.txt is too short
and reallylong.txt is too long
but super.txt is just right!

$ ./rgrep ' ....?.?\.txt' < testin #note the space before the first .
this file is fine.txt
but super.txt is just right!
```

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 iff the pattern "appears" somewhere inside the line**. In the absense 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 characters you must implement:

. (period)	Matches any character.
? (question mark)	The preceding character may or may not appear in the line.
\ (backslash)	"Escapes" the following character, nullifying any special meaning it has

So, here are some examples of patterns and the kind of lines they match:

\?	a question mark must appear in the line
they?re	Matches a line that contains either the string "theyre" or the string "there"
h.d..?n	matches lines that contain substrings like "hidden", "hidin", "hbdwen", "hadbn", etc.
cu\.?t	matches lines that either contain the substring "cut" or "cu.t"

You may assume that your code will not be run against patterns that don't make sense, like "hello??" and "?oops".

Getting started

Copy the framework files from the assignment page. 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 pattern_match_startat(char *pattern, char*str)`, which returns true if and only if the string matches the pattern. **Please do not modify the skeleton code**. You may add whatever support functions, types, variables, whatever you want, but please keep the code that appears in the skeleton intact.

Testing

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. You can also just invoke `./rgrep pattern` and type lines, and verify that `rgrep` repeats lines iff they match the pattern.

Note that your shell might interpret the backslash or question mark operators 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.

Extra for Experts

- Implement grouping. Characters within parentheses are treated as a single unit with respect to the ? operator. So, for example, the pattern `f(oob)?ar` would match strings containing either "foobar" or "far".

- Implement the * operator. The * operator works like the ? operator but instead of allowing the previous thing to appear 0 or 1 times, it allows it to appear 0 or more times. So, for instance, `fa*r` would match strings containing `fr, far, faar, faaar`, etc.