140.636, Computer Science for Bioinformatics

Computer Laboratory: Regular expressions

Fernando J. Pineda

Background

The purpose of this lab is to get some practice with regular expressions. A regular expression cheat sheet may be found here:

http://www.cheat-sheets.org/saved-copy/perl-regexp-refcard-a4.pdf

Copy the **`retest2.pl`** script in the directory:

```
/users/sph140636/shared/code/03_regexp
```

into your home directory. Use it to try out regular expressions. It would be a good script to understand.

The **`retest.pl`** script is a simpler (but harder to use) command line interpreter for testing simple regular expressions.

```
#!/usr/bin/perl
print ("\nEnter string or cntl-D to quit\n");
print ("Square brackets indicate text that matched pattern\n\n");
$prompt = "test> ";
print $prompt;
while(<STDIN>) {
        chomp;
        if(/REPLACE_THIS_WITH_YOUR_PATTERN/) {
                print("$`\[$&]$'\n");
        }
        else {
                print("no match\n");
        }
        print $prompt;
}
```

The obscure looking print statement makes use of three predefined variables (with very strange names). These are **`$``**, **`$&`** and **`$'`**. The value of the variable **`$&`** is the string that matched the regular expression pattern, while **`$``** and **`$'`** are the parts of the string before and after the match respectively

```
m/pattern/i # i causes case-insensitive matching
m/pattern/g # g causes global matching
m/pattern/s # s allows '.' to match '\n'
```

The term *global match* means that the match operator will match all occurrences of a given pattern -- not just the first one. (Note: the **`retest.pl`** script only indicates the first match. The **`pos()`** function is useful for getting all the matches). Scalar interpolation works in Perl regular expressions. This is handy if you want to build a regular expression in perl or if you want to input a regular expression from a file or STDIN, e.g.

```
$variable = ‘pattern’;
m/$variable/;
```

Exercises

Use **`retest2.pl.`** It is recommend that you read each question VERY, VERY carefully.

1. <u>What is a pattern that matches the ***substring*** “world” occurring anywhere in the input string, e.g.</u>

   `hello cold cruel` *world*
   `hello` *world* `news tonight`
   `hello`*world*.pl is a script

2. <u>What is a pattern that matches the ***word*** “world” occurring anywhere in the input string, e.g.</u>

   `hello cold cruel` *world*
   `hello` *world* `news tonight`

   <u>but not</u>

   `helloworld is a script`

3. <u>What is a pattern that matches the **word** “world” only if occurs at the end of the string, i.e</u>

   `hello cold cruel` *world*
   <u>but not</u>
   `next is` *world* `news tonight`
   `hello cold cruelworld`

4. <u>What is a pattern that matches a string that starts with the **word** “hello” OR ends in the **word** “world”, e.g.</u>

   hello and good night
   that’s all for tonight world

5. What is a pattern that matches a string that starts with the **word** "hello" OR "bye", AND ends with the **word** "world", e.g.

    bye cold cruel world
    hello cold cruel world
    hello cold cruel.world

   but not

```
hello cold cruel world?
hello cold cruelworld
```

6. What is a pattern that matches a **substring** "world" occurring 1 or more times at the end of the line, e.g.

```
This string ends in world
This string ends in worldworld
This string ends in worldworldworld
```

7. What is a pattern that matches a string that ends in one or more of backslashes immediately followed by one or more asterisks that continue until the end of the string, e.g.

```
\\\\*****
```

   but not

```
\\\\\\\*****\
```

8. What is a pattern that matches any line of input that has the same word repeated two or more times *in a row*. In this problem, words can be considered to be sequences of letters **a** to **z, A** to **Z**, digits, and underscores. Whitespace between words may differ, e.g.

    Paris in the the spring
    I thought that   that was the problem

   For this example you will need to use *backreferences*. A backreference is a reference to a string captured with parentheses. (Recall that in Perl, captured strings are referred to as **$1,…,$9**) In a regular expression, you can refer to captured strings, while the pattern is being matched, as **\1,…\9**. For example, **/(AT)G(\1)/** matches a 5 character string **ATGAT**.

   Note: Strictly speaking the inclusion of back-references makes the Perl pattern recognition language able to recognize some *context-free grammars* because back-references are a form of memory.