Week 2 Lab - Regular Expressions

COSC244 & TELE202

1 Introduction

The first lab consisted of an introduction to (or a recap of) using the Unix command-line. The best way to increase your proficiency at using the command-line is through regular use. Whenever possible, use a terminal to get things done rather than the GUI and before you know it it will be second nature. Like the first lab, this lab covers a topic which you might not think is directly related to networking — *regular expressions*. Regular expressions can be used in many different networking applications, for example:

- validating email or IP address formats
- extracting specific information from a web page
- configuring firewalls
- scanning log files

At its most basic level regular expressions are a way of matching strings. Most people are familiar with performing some type of *find* operation in something like a word processor. A target string is entered and every occurrence of the target string is displayed. There is also an option to substitute each matched string with a replacement string. Regular expressions allow you to perform a very powerful form of matching by giving a special meaning to some characters. Many word processors will actually let you use some form of regular expressions when searching and replacing text.

To interactively try matching regular expressions yourself type this command in a terminal

```
grep -E --colour 'exp'
```

where *exp* is the regular expression that you want to match. Note that the single quotes around the regular expression can be necessary, in order to prevent the shell from interpreting any special characters that you use. When you enter the command above it will appear to do nothing. Now start entering lines of text like this:

```
this an expression
expand this expression
hello
```

Any lines which match the regular expression will get printed out with the matching part highlighted.

2 Regular expression tutorial

A widely used tutorial for learning about regular expressions can be found online at: http://www.regular-expressions.info/tutorial.html.

Go to the regular expression tutorial website and begin the tutorial. On the top left-hand side of the page you will see a navigation box headed *Regex Tutorial*. Work your way through each of the sections starting from the `Introduction` down to `Back-references` (but don't worry about Character Class Subtraction, Intersection, or Shorthand). As you work through the tutorial open up a new firefox window and visit http://regexpal.com. This will allow you to try out examples in one window as you work through the tutorial in the other window. You might find the cheat sheet on the right hand side of the web page useful. Once you have worked through all of the sections mentioned above then complete the following exercises.

3 Exercises

Open up emacs and then enter the command **M-x lab2** to open up a copy of the exercise questions. In each of the examples given record your answer by putting a 'y' beside each number which exactly matches the given regular expression. Try to answer without using any tools to help you. Do not include partial matches (where part of the line matches the regex).

A. **a(ab)*a**

1) aabbaa
2) abababa
3) aaba
4) aba
5) aabababa

B. **ab+c?**

1) abc
2) ac
3) abbb
4) bbc

C. **abc|xyz**

1) abc
2) xyz
3) abc|xyz

D. **a.[bc]+**

1) abc
2) abbbbbbbb
3) abcbcbcbc
4) ac
5) asccbbbbcbcccc

E. **[a-z]+[.?!]**

1) battle!
2) Hot.
3) green
4) swamping.
5) undulate?
6) is.?

F. **[a-zA-Z]+[^,]=**

1) Putt=
2) BotHEr,=
3) Ample
4) FIdDlE7h=
5) Brittle =
6) Other.=

G. **<[^>]+>**

1) <an xml tag>
2) <opentag> <closetag>
3) </closetag>
4) <>
5) <with attribute="red">

H. **(very)+(fat)?(tall|jolly) man**

1) very fat man
2) fat tall man
3) very very fat jolly man
4) very short man
5) very very very tall man

I. **[a-z][.?!] +[A-Z]**

1) A. B
2) c! d
3) e f
4) g. H
5) i? J
6) k L

Once you have completed the answers in **regex.txt** run the command **M-x re-builder** in Emacs to check your answers (make sure you check the note below about different escape requirements). Put the cursor between the double quotes in the `RE-Builder` buffer and then type each expression to see which ones match.

After checking that you get the same answers for A-I, construct answers to J and K using the RE-Builder. When the cursor is in the RE-Builder buffer there will be an *RE-Builder* menu showing the available commands.

Note: Different regular expression implementations can have slightly different sets of features as well as different escape requirements. For example, when using grep you have to use `\+` instead of `+`. When using Emacs you have to escape the characters `(){}|` with a double backslash, i.e. `\\(\\)\\{\\}\\|`.

J. Write a regular expression which will match a decimal floating-point number like 23.7 or -13.549

K. Write a regular expression which will match two xml tags (with no attributes) which are not nested correctly e.g. `<a-tag> <another-tag> </a-tag> </another-tag>`
You should use grouping and back references.

L. Suppose you had a Java file which contained lots of single letter variable names like a,x,f etc, and you wanted to change them to something more meaningful. You could try replacing every occurrence of 'f' with 'filename', but that would mess up all of the places where 'f' occurs inside a word (for example in the keyword *if*). How would you go about correcting this problem using regular expression search and replace? Try your solution in Emacs.

Before you leave the lab, remember to get marked off by a demonstrator.

This lab is worth 0.5%.