**CS250 Lab 3 – Regular Expressions**

**Objectives:** In this lab you will learn how to

- use regular expressions to express patterns

Regular expressions, as the name suggests, represent regular languages. JFLAP allows us to enter a regular expression, build an equivalent NFA, and test out the language represented by the expression. Just select the Regular Expression button on the initial menu, after entering the regular expression select Convert to NFA. Select the Do All button and then the Export button. You should now have an equivalent NFA which you can test.

**Assignment 1**:
For each regular expression below, enter it in JFLAP (be sure not to include spaces in the regular expression), convert to an NFA, use multiple run tester to list five input strings in the language (if there exists at least five). Finally, clearly explain the regular language that it represents:

1. a*b*+c
2. (ab+$\lambda$)b – Hint: $\lambda$ is represented by ! in JFLAP
3. (cd+b)*

**Assignment 2**:
Write a regular expression for each of the following languages and test it for correctness in JFLAP.

1. The language with an even number of `a`'s followed by an odd number of `b`'s
2. The language with zero or more `a`'s followed by three or fewer `b`'s
3. The language in which every `a` must have `b` adjacent to it on both sides.

The program `egrep` is an acronym for "extended global regular expressions print" and may be used to search for a string or a more complex pattern in a file. Regular expressions provide a convenient, compact way of expressing patterns. The internal workings of egrep are based on finite automata.

The command format is:

```
egrep 'regexp' filename
```

Egrep returns all lines in the file which contain a match for the regular expression.

The format of a regular expression in egrep is as follows:

| regular expression | egrep notation |
|---|---|
| r* | `r*` |
| $r^+$ | `r+` |
| r + $\lambda$ | `r?` |
| r + s | `r\|s` |
| rs | `rs` |
| ( r ) | `(r)` |
| char c | c |
| special char | `\c` |
| any symbol | `.` |
| beginning-of-line | `^` |
| end-of-line | `$` |
| any character listed | `[ ]` |
| any character not listed | `[^ ]` |

**Assignment 3**:
In a terminal window on phoenix try the following commands and then clear and succinctly describe the pattern expressed by the regular expression:

1. `egrep 'depend' /usr/share/dict/words`
2. `egrep '^y.*y$' /usr/share/dict/words`
3. `egrep '^s..u.t..e$' /usr/share/dict/words`
4. `egrep '[qQ][^u]' /usr/share/dict/words`

**Assignment 4**:
Write `egrep` commands for the following:

1. All lines that contain the letter `a` and the letter `b` (either upper or lower case)
2. All lines that contain the word `a` (either upper or lower case) followed by a word starting with a vowel

Test your commands with the file `~jillz/cs250/lab3/testfile1` and on any other testfile of your choosing.

**Assignment 5**:
Try the regex `[0-9]?[0-9]:[0-9][0-9](am|pm)` on the file
`~jillz/cs250/lab3/testfile2`
Correct this regex so that it does not match illegal times like 99:99pm

Regular expressions are part of the pattern matching facility of the language perl. A perl program is available for you to try:
`~jillz/cs250/lab3/convert`

Copy this file to your own directory. The program takes user input which is a temperature in either celcius or fahrenheit and converts it to the other scale.
The line `$input = ~ m/^([-+]?[0-9]+)([CF])$/)` reads a line of input and matches it to the given regular expression. After the match, the special variables `$1` and `$2` will contain the parts of the inputs that matched the parenthesized groups in the expression. So if the user entered -32C then `$1` would contain -32 and `$2` would contain C.

**Assignment 6**:
Test out the program `convert`:

```
perl convert
```

Modify the regular expression in `convert` so that the temperatures may be decimal values and the C and F may be either lower or upper case. Also allow there to be possible whitespace between the number and the C or F.

Submit your files in goucherLearn for grading.