Regular Expressions
Beginning Intermediate

There is a single construct, mightily overloaded, which brings incredible power to regular expressions. The construct starts out straightforwardly, as a matched pair of parens, and works like you would expect for grouping. This gives us two immediate features at this point. The first is that we can have ORs within parentheses that apply locally, within the parens, while the second feature is that quantifiers can be applied to parens.

Thus, `/^0$|^10[10]$/` is equivalent to `/^(0|10[10])$/`

While in the above example, the lengths are the same, parentheses often bring an organizational clarity to regular expressions. The example below, which matches on words of at least two vowels, is more compelling:

`/\w*[aeiou]\w*[aeiou]\w*/` is better written as `/(\w*[aeiou]){2}\w*/`

The quantifier on a parenthesized subexpression is the same as if that subexpression appeared as many times as the quantifier indicated. In the case of a question mark (one or zero), if the subexpression matches, that is preferred (greediness), but it is also OK if there is no match. Just as before, if the quantity that applies is more than one, there is no implication that there is any repetition. For example `/[aeiou]{2}/` does not say "repeated vowel". Rather, it says "adjacent vowels" because it is like writing `/[aeiou][aeiou]/`.

It is incorrect to think of parentheses as merely being a convenience, as they were in the prior two examples. In particular, parentheses contain subexpressions that can be viewed as all or nothing when a quantifier is applied to them. For example, `/^(abc?)+$/` says that it is looking for strings consisting of `ab` repeated, each of which may have up to one `c` following it. This regular expression can't be achieved without parens.

Regular Expressions – part 2/4 (Intermediate 1)

Kim – Exercises B – Gabor

Determine a regular expression which will match only on the indicated strings, or else will find the indicated matches. The form for the submission will be a command line script that takes a single integer as input from 41 to 50, inclusive, and outputs the corresponding regular expression pattern, just as with the prior HW. The pattern is to be delimited by forward slashes and any options should immediately follow the final slash. Due F, 15 Feb.

For problems 41-43, an Othello board will be construed to be and 8×8 board, represented as a single string, where holes (blanks) are indicated by a period (`.`), and the black tokens are indicated by either an `x` or `X`, and the white tokens are indicated by an `o` or `0`. The boards have no "buffer edge". For the purposes of this problem, we are not concerned about whether we could actually reach a board in real play. For example, a board with tokens only at the corners would be considered a valid Othello board, and so would a completely empty board.

41. Write a regular expression that will match on an Othello board represented as a string.

42. Given a string of length 8, determine whether it could represent an Othello edge with exactly one hole.

43. Given an Othello edge as a string, determine whether there is a hole such that if X plays to the hole (assuming it could), it will be connected to one of the corners through X tokens.

44. Match on all strings of odd length.

45. Match on all odd length binary strings starting with `0`, and on even length binary strings starting with `1`.

46. Match all words having two adjacent vowels that differ.

47. Match on all binary strings which DON'T contain the substring `110`.

48. Match on all non-empty strings over the alphabet {`a`, `b`, `c`} that contain at most one `a`.

49. Match on all non-empty strings over the alphabet {`a`, `b`, `c`} that contain an even number of `a`'s.

50. Match on all base 3 integer strings that are even.

