CS 35L

Software Construction Laboratory

Assignment 2 - Homework

17th January, 2019

Homework 2 Tips

- Refer to Piazza for common queries
- Ahead are slides on:
  - Basic Regular Expressions & Extended Regular Expressions
  - POSIX Bracket Expressions
  - ASCII, Unicode and UTF- 8
- My office hours for Week 2 are on January 18th from 2:30pm to 4:30pm

Basic Regular Expressions (BRE) vs Extended Regular Expressions (ERE)

- In basic regular expressions the meta-characters ‘?’, ‘+’, ‘{’, ‘|’, ‘(’, and ‘)’ lose their special meaning; instead use the backslashed versions ‘\?’, ‘\+’, ‘\{’, ‘\|’, ‘\(’, and ‘\)’ for their special meaning.
- In extended regular expressions, the meta characters, ‘?’, ‘+’, ‘{’, ‘|’, ‘(’, and ‘)’ retain their special meaning. They can be literally used by escaping them:‘\?’, ‘\+’, ‘\{’, ‘\|’, ‘\(’, and ‘\)’.
- *man grep* for more information

Regular expressions

Character	BRE / ERE	Meaning in a pattern
\	Both	Usually, turn off the special meaning of the following character. Occasionally, enable a special meaning for the following character, such as for \(...\) and \{...\}.
.	Both	Match any single character except NULL. Individual programs may also disallow matching newline.
*	Both	Match any number (or none) of the single character that immediately precedes it. For EREs, the preceding character can instead be a regular expression. For example, since . (dot) means any character, .* means "match any number of any character." For BREs, * is not special if it's the first character of a regular expression.
^	Both	Match the following regular expression at the beginning of the line or string. BRE: special only at the beginning of a regular expression. ERE: special everywhere.

Regular Expressions (cont'd)

$	Both	Match the preceding regular expression at the end of the line or string. BRE: special only at the end of a regular expression. ERE: special everywhere.
[...]	Both	Termed a bracket expression, this matches any one of the enclosed characters. A hyphen (-) indicates a range of consecutive characters. (Caution: ranges are locale-sensitive, and thus not portable.) A circumflex (^) as the first character in the brackets reverses the sense: it matches any one character not in the list. A hyphen or close bracket (]) as the first character is treated as a member of the list. All other metacharacters are treated as members of the list (i.e., literally). Bracket expressions may contain collating symbols, equivalence classes, and character classes (described shortly).
\{*n*,*m*\}	BRE	Termed an *interval expression*, this matches a range of occurrences of the single character that immediately precedes it. \{*n*\} matches exactly n occurrences, \{*n*,\} matches at least n occurrences, and \{*n*,*m*\} matches any number of occurrences between n and m. n and m must be between 0 and RE_DUP_MAX (minimum value: 255), inclusive.
\(\)	BRE	Save the pattern enclosed between \(and \) in a special *holding space*. Up to nine sub patterns can be saved on a single pattern. The text matched by the sub patterns can be reused later in the same pattern, by the escape sequences \1 to \9. For example, **\(ab\).*\1** matches two occurrences of ab, with any number of characters in between.

Regular Expressions (cont'd)

\n	BRE	Replay the nth subpattern enclosed in \(and \) into the pattern at this point. n is a number from 1 to 9, with 1 starting on the left.
{n,m}	ERE	Just like the BRE \{n,m\} earlier, but without the backslashes in front of the braces.
+	ERE	Match one or more instances of the preceding regular expression.
?	ERE	Match zero or one instances of the preceding regular expression.
|	ERE	Match the regular expression specified before or after.
( )	ERE	Apply a match to the enclosed group of regular expressions.

Regular Expressions (cont'd)

*	Match zero or more of the preceding character
\{*n*\}	Exactly n occurrences of the preceding regular expression
\{*n*,\}	At least n occurrences of the preceding regular expression
\{*n*,*m*\}	Between n and m occurrences of the preceding regular expression

POSIX Bracket Expressions

Class	Matching characters	Class	Matching characters
[:alnum:]	Alphanumeric characters	**[:lower:]**	**Lowercase characters**
[:alpha:]	Alphabetic characters	[:print:]	Printable characters
[:blank:]	**Space and tab characters**	[:punct:]	Punctuation characters
[:cntrl:]	Control characters	[:space:]	Whitespace characters
[:digit:]	Numeric characters	**[:upper:]**	**Uppercase characters**
[:graph:]	Nonspace characters	[:xdigit:]	Hexadecimal digits

Useful grep options

- • -l and -L
  - Suppress normal output; instead print the name of each input file from which output would normally have been printed. The scanning will stop on the first match.
  - Uppercase L prints all complementary filenames
- -v
  - Invert sense of matching
- Egrep or grep -e
  - uses extended regular expressions

ASCII, Unicode and UTF-8

- ASCII (both a character set and an Encoding):
  - 128 Characters
  - Encoded with bytes for each character
  - Byte values 128-255 note used (invalid)
  - Uniform Length Code
- Unicode (Character set):
  - 1,112,064 valid code points
- UTF-8 (Encoding):
  - All code points between 1-4 bytes (var length)
  - Prefix-free code -- No 2 code points have same prefix
  - ASCII is a subset with same byte representation
    - Therefore, all non-ASCII characters have a prefix that starts with an ascii-invalid byte.
- https://medium.com/@apiltamang/unicode-utf-8-and-ascii-encodings-made-easy-5bfbe3a1c45a - For Additional reference