Flex's Patterns

The patterns in the input are written using an extended set of regular expressions. These are:

`'x'`
match the character `'x'`

`'.'`
any character (byte) except newline

`'[xyz]'`
a "character class"; in this case, the pattern matches either an `'x'`, a `'y'`, or a `'z'`

`'[abj-oZ]'`
a "character class"with a range in it; matches an `'a'`, a `'b'`, any letter from `'j'` through `'o'`, or a `'Z'`

`'[^A-Z]'`
a "negated character class", i.e., any character but those in the class. In this case, any character EXCEPT an uppercase letter.

`'[^A-Z\n]'`
any character EXCEPT an uppercase letter or a newline

`'[a-z]{-}[aeiou]'`
the lowercase consonants

`'r*'`
zero or more r's, where r is any regular expression

`'r+'`
one or more r's

`'r?'`
zero or one r's (that is, "an optional r")

`'r{2,5}'`
anywhere from two to five r's

`'r{2,}'`
two or more r's

`'r{4}'`
exactly 4 r's

`'{name}'`
the expansion of the `'name'` definition

`'"[xyz]\"foo"'`
the literal string: `'[xyz]"foo'`

`'\x'`
if `'x'` is `'a'`, `'b'`, `'f'`, `'n'`, `'r'`, `'t'`, or `'v'`, then the ANSI-C interpretation of `'\x'`. Otherwise, a literal `'x'` (used to escape operators such as `'*'`)

`'\0'`
a NUL character (ASCII code 0)

`'\123'`
the character with octal value 123

`'\x2a'`
the character with hexadecimal value 2a

'(r)'
match an 'r'; parentheses are used to override precedence (see below)

'(?r-s:pattern)'
apply option 'r' and omit option 's' while interpreting pattern. Options may be zero or more of the characters 'i', 's', or 'x'.

'i' means case-insensitive. '-i' means case-sensitive.

's' alters the meaning of the '.' syntax to match any single byte whatsoever. '-s' alters the meaning of '.' to match any byte except '\n'.

'x' ignores comments and whitespace in patterns. Whitespace is ignored unless it is backslash-escaped, contained within '""'s, or appears inside a character class.

The following are all valid:

`(?:foo)`	same as	`(foo)`
`(?i:ab7)`	same as	`([aA][bB]7)`
`(?-i:ab)`	same as	`(ab)`
`(?s:.)`	same as	`[\x00-\xFF]`
`(?-s:.)`	same as	`[^\n]`
`(?ix-s: a . b)`	same as	`([Aa][^\n][bB])`
`(?x:a  b)`	same as	`("ab")`
`(?x:a\ b)`	same as	`("a b")`
`(?x:a" "b)`	same as	`("a b")`
`(?x:a[ ]b)`	same as	`("a b")`

`(?x:a`
`/* comment */`
`b`
`c)`
	same as	`(abc)`

'(?# comment)'
omit everything within '()'. The first ')' character encountered ends the pattern. It is not possible to for the comment to contain a ')' character. The comment may span lines.

'rs'
the regular expression 'r' followed by the regular expression 's'; called "concatenation"

'r|s'
either an 'r' or an 's'

'r/s'
an 'r' but only if it is followed by an 's'. The text matched by 's' is included when determining whether this rule is the longest match, but is then returned to the input before the action is executed. So the action only sees the text matched by 'r'. This type of pattern is called "trailing context". (There are some combinations of 'r/s' that flex cannot match correctly. Ver documentação.)

'^r'
an 'r', but only at the beginning of a line (i.e., when just starting to scan, or right after a newline has been scanned).

'r$'
an 'r', but only at the end of a line (i.e., just before a newline). Equivalent to 'r/\n'.

Note that 'flex''s notion of "newline" is exactly whatever the C compiler used to compile 'flex' interprets '\n' as; in particular, on some DOS systems you must either filter out '\r's in the input yourself, or explicitly use 'r/\r\n' for 'r$'.

'<s>r'
an 'r', but only in start condition 's' (ver documentação)

'<s1,s2,s3>r'
: same, but in any of start conditions 's1', 's2', or 's3'.

'<*>r'
: an 'r' in any start condition, even an exclusive one.

'<<EOF>>'
: an end-of-file.

'<s1,s2><<EOF>>'
: an end-of-file when in start condition 's1' or 's2'

Note that inside of a character class, all regular expression operators lose their special meaning except escape ('\') and the character class operators, '-', ']]', and, at the beginning of the class, '^'.

The regular expressions listed above are grouped according to precedence, from highest precedence at the top to lowest at the bottom. Those grouped together have equal precedence (see special note on the precedence of the repeat operator, '{}', under the documentation for the '--posix' POSIX compliance option). For example,

foobar*|

is the same as

(foo)(ba(r*))|

since the '*' operator has higher precedence than concatenation, and concatenation higher than alternation (''|). This pattern therefore matches **either** the string 'foo' **or** the string 'ba' followed by zero-or-more 'r''s. To match 'foo' or zero-or-more repetitions of the string 'bar', use:

foo(bar)*|

And to match a sequence of zero or more repetitions of 'foo' and 'bar':

(foobar)*|

In addition to characters and ranges of characters, character classes can also contain "character class expressions". These are expressions enclosed inside '[:' and ':]' delimiters (which themselves must appear between the '[' and ']' of the character class. Other elements may occur inside the character class, too). The valid expressions are:

```
[:alnum:] [:alpha:] [:blank:]
[:cntrl:] [:digit:] [:graph:]
[:lower:] [:print:] [:punct:]
[:space:] [:upper:] [:xdigit:]
```

These expressions all designate a set of characters equivalent to the corresponding standard C 'isXXX' function. For example, '[:alnum:]' designates those characters for which 'isalnum()' returns true - i.e., any alphabetic or numeric character. Some systems don't provide 'isblank()', so flex defines '[:blank:]' as a blank or a tab.

For example, the following character classes are all equivalent:

```
[[:alnum:]]
[[:alpha:][:digit:]]
[[:alpha:][0-9]]
[a-zA-Z0-9]
```

A word of caution. Character classes are expanded immediately when seen in the 'flex' input. This means the character classes are sensitive to the locale in which 'flex' is executed, and the resulting scanner will not be sensitive to the runtime locale. This may or may not be desirable.

- If your scanner is case-insensitive (the '-i' flag), then '[:upper:]' and '[:lower:]' are equivalent to '[:alpha:]'.

- Character classes with ranges, such as '[a-Z]', should be used with caution in a case-insensitive scanner if the range spans upper or lowercase characters. Flex does not know if you want to fold all upper and lowercase characters together, or if you want the literal numeric range specified (with no case folding). When in doubt, flex will assume that you meant the literal numeric range, and will issue a warning. The exception to this rule is a character range such as '[a-z]' or '[S-W]' where it is obvious that you want case-folding to occur. Here are some examples with the '-i' flag enabled:

Range	Result	Literal Range	Alternate Range
'[a-t]'	ok	'[a-tA-T]'	
'[A-T]'	ok	'[a-tA-T]'	
'[A-t]'	ambiguous	'[A-Z\[\\\]_`a-t]'	'[a-tA-T]'
'[_-{]'	ambiguous	'[_`a-z{]'	'[_`a-zA-Z{]'
'[@-C]'	ambiguous	'[@ABC]'	'[@A-Z\[\\\]_`abc]'

- A negated character class such as the example '[^A-Z]' above **will** match a newline unless '\n' (or an equivalent escape sequence) is one of the characters explicitly present in the negated character class (e.g., '[^A-Z\n]'). This is unlike how many other regular expression tools treat negated character classes, but unfortunately the inconsistency is historically entrenched. Matching newlines means that a pattern like '[^"]*' can match the entire input unless there's another quote in the input.

Flex allows negation of character class expressions by prepending '^' to the POSIX character class name.

```
[:^alnum:] [:^alpha:] [:^blank:]
[:^cntrl:] [:^digit:] [:^graph:]
[:^lower:] [:^print:] [:^punct:]
[:^space:] [:^upper:] [:^xdigit:]
```

Flex will issue a warning if the expressions '[:^upper:]' and '[:^lower:]' appear in a case-insensitive scanner, since their meaning is unclear. The current behavior is to skip them entirely, but this may change without notice in future revisions of flex.

- The '{-}' operator computes the difference of two character classes. For example, '[a-c]{-}[b-z]' represents all the characters in the class '[a-c]' that are not in the class '[b-z]' (which in this case, is just the single character 'a'). The '{-}' operator is left associative, so '[abc]{-}[b]{-}[c]' is the same as '[a]'. Be careful not to accidentally create an empty set, which will never match.

- The '{+}' operator computes the union of two character classes. For example, '[a-z]{+}[0-9]' is t
z0-9]'|. This operator is useful when preceded by the result of a difference operation, as in, '[[:alpha:]]{-}[[:lower:]]{+}[q]', which is equivalent to '[A-Zq]' in the "C"locale.

- A rule can have at most one instance of trailing context (the '/' operator or the '$' operator). The start condition, '^', and '<<EOF>>' patterns can only occur at the beginning of a pattern, and, as well as with '/' and '$', cannot be grouped inside parentheses. A '^' which does not occur at the beginning of a rule or a '$' which does not occur at the end of a rule loses its special properties and is treated as a normal character.

- The following are invalid:

  ```
  foo/bar$
  <sc1>foo<sc2>bar
  ```

  Note that the first of these can be written 'foo/bar\n'.

- The following will result in '$' or '^' being treated as a normal character:

```
foo|(bar$)
foo|^bar
```

If the desired meaning is a 'foo' or a 'bar'-followed-by-a-newline, the following could be used (the special '|' action is explained below e na documentação):

```
foo       |
bar$      /* action goes here */
```

A similar trick will work for matching a 'foo' or a 'bar'-at-the-beginning-of-a-line.