Context-free Grammar for the syntax of regular expression over the ASCII character set

assumption :

- A regular expression is to be interpreted a Haskell string, then is used to match against a Haskell string. Therefore, each regexp is enclosed inside a pair of double quotes, just like any Haskell string. For clarity, a `regexp` is highlighted and a "Haskell input string" is quoted for the examples in this document.

- Since ASCII character strings will be encoded as in Haskell, therefore special control ASCII characters such as NUL and DEL are handled by Haskell.

context-free grammar :

BNF notation is used to describe the syntax of regular expressions defined in this document, with the following basic rules:

- `<nonterminal> ::= choice1 | choice2 | ...`
- Double quotes are used when necessary to reflect the literal meaning of the content itself.

```
<regexp>    ::=   <union> | <concat>

<union>     ::=   <regexp> "|" <concat>

<concat>    ::=   <term><concat> | <term>

<term>      ::=   <star> | <element>

<star>      ::=   <element>*

<element>   ::=   <group> | <char> | <emptySet> | <emptyStr>

<group>     ::=   (<regexp>)

<char>      ::=   <alphanum> | <symbol> | <white>
<alphanum>  ::=   A | B | C | ... | Z |
                  a | b | c | ... | z |
                  0 | 1 | 2 | ... | 9
<symbol>    ::=   ! | " | # | $ | % | & | ' | + | , | - | . | / | : | ; | < | = |
                  > | ? | @ | [ | ] | ^ | _ | ` | { | } | ~ | <sp> | \<metachar>
<sp>        ::=   " "
<metachar>  ::=   \ | "|" | ( | ) | * | <white>
<white>     ::=   <tab> | <vtab> | <nline>
<tab>       ::=   \t
<vtab>      ::=   \v
<nline>     ::=   \n
<emptySet>  ::=   Ø
<emptyStr>  ::=   ""
```

Explanations:

1. Definition of `<metachar>` in our definition of regexp:

Symbol	**meaning**
`\`	Used to escape a metacharacter, `\*` means the star char itself
`|`	Specifies alternatives, `y|n|m` means y OR n OR m
`(...)`	Used for grouping, giving the group priority
`*`	Used to indicate zero or more of a regexp, `a*` matches the empty string, "a", "aa", "aaa" and so on
Whitespace char	**meaning**
`\n`	A new line character
`\t`	A horizontal tab character
`\v`	A vertical tab character

2. Notice that a space character in our regexp definition is just a literal space. However, special symbols are used to represent other whitespace characters such as a tab and a new line characters.

3. Since the backslash is used as our escape character, and so does the Haskell compiler, we need to be careful when we use our regexp as a Haskell string to match against a Haskell input string. At the top level is the regexp compiler, the next level is the Haskell string compiler, and the bottom level is the literal string itself. This is more clear with the illustration of an example:

Say we want to match the string *alice**chen* , which is literally *a backslash between alice and chen*. The literal string is illegal in Haskell; it needs to be converted to "alice\\chen", which is matched with the regexp `alice\\chen`, which in turn, becomes "alice\\\\chen" as a Haskell string.

4. Note also that the double quote " is not a metacharacter in our regexp definition; however, it does have a special meaning as a Haskell character. So it needs to be escaped as a Haskell string.

5. the definition of concatenation `<term><concat> | <term>` indicates that concatenation is right associative

6. this context free grammar reveals the precedence of various operators used in our definition of a regular expression as follows, from highest priority to lowest: the `<char>` themselves, `<group>`, `<star>`, `<concat>`, then `<union>`

7. In our definition, both `<emptyStr>` and `<emptySet>` are closed under `<union>`,`<concat>`,`<group>` and `<star>`. According to set theory, concatenation of a string to an empty string is always the string itself; however, any concatenation with the empty set results in the empty set. On the other hand, Ø* would be the set with the empty string.

Reference:

1. Haskell 98 Report Syntax Reference http://www.haskell.org/onlinereport/syntax-iso.html#sect9
2. Haskell escape rules http://book.realworldhaskell.org/read/characters-strings-and-escaping-rules.html
3. Regular expression reference http://www.regular-expressions.info/reference.html
4. Java Language Specifications http://java.sun.com/docs/books/jls/third_edition/html/j3TOC.html
5. Regular expressions in Java http://www.wellho.net/solutions/java-regular-expressions-in-java.html
6. Programming in Haskell (textbook) p83
7. Set theory concept http://unicode.org/mail-arch/unicode-ml/y2003-m06/0034.html