CSE 3341 Midterm 1 Review Solutions

1. Let $\Sigma = \{0, 1\}$. Let $L \subseteq \Sigma^+$ be the language of all strings where the first and last character are equal. Write a regular expression that recognizes all strings from L.
   0(0|1)*0|1(0|1)*1|0|1

2. Is every finite language a regular language? Explain your answer.
   Yes. For any finite languages, select an ordering $s_1, s_2, \ldots, s_n$ of all strings in the language. Then
   $S \to s_1|s_2|\ldots|s_n$
   is a regular grammar for that language.

3. Write a regular grammar for the language of all strings composed of 'a', 'b', 'c' but each string uses at most two of the three letters. For example, this language contains strings like "aabbaa", "cacacaca", "bbbbb", but does not contain strings like "abc".

$$S \to A|B|C$$
$$A \to aA|bA|\epsilon$$
$$B \to bB|cB|\epsilon$$
$$C \to aC|cC|\epsilon$$

4. Suppose we try to solve the "dangling else" problem with this grammar:

```
<statement>   ::=  <if> | <assign>
<if>          ::=  if <condition> then <statement> <if-end>
<if-end>      ::=  ϵ
                   | else <statement>
```

   Does this solve the problem? You can assume <assign> and <condition> do the obvious thing and have no effect on the question.
   Does not solve the problem. We can still create two parse trees for strings like "if a then if b then c=1 else c=2".

5. Consider the following grammar of expressions:
   $\langle exp \rangle$ ::= id | hex_literal | $\langle exp \rangle$ + $\langle exp \rangle$ | $\langle exp \rangle$ * $\langle exp \rangle$
   In this grammar, id and hex_literal are terminal symbols; that is, tokens that will be produced by the scanner and consumed by the parser.
   Change this grammar (by adding new non-terminals and new productions) to achieve these goals:
   First, make the precedence of operator + lower than the precedence of operator *.
   Second, make operators + and * left-associative.
   Third, add a pre-increment operator ++ to represent expressions of the form ++A. The pre-increment operator applies only to identifiers, and has higher precedence than + and *.

```
<exp>      ::=  <exp> + <term> | <term>
<term>     ::=  <term> * <factor> | <factor>
<factor>   ::=  ++id | id | hex_literal
```