CS 214 Project 2: Practicing with BNFs

This week's project is to practice creating BNF definitions and work with derivations.

The are three key ideas to keep in mind:

- If a portion of a language construct is *complicated*, hide the complexity by creating a nonterminal for it, and then define the nonterminal (later).
- If a language construct is *optional*, create a nonterminal that has two productions, one of which defines the optional part, and the other of which is an ø-production.
- If a language construct can be *repeated*, create a non-terminal that has two productions, one of which is a recursive production providing the repetition, and the other of which is an ø-production by which the recursion can terminate.

1. Using these ideas, define the constructs below with BNF productions. Underline the terminals in your productions, to distinguish them from the nonterminals. You may assume that the following have already been defined:

```
<letter>   ::=   A | B | C | D | E | F | G | H | I | J | K | L
                 | M | N | O | P | Q | R | S | T | U | V | W
                 | X | Y | Z | a | b | c | d | e | f | g | h
                 | i | j | k | l | m | n | o | p | q | r | s
                 | t | u | v | w | x | y | z
<digit>    ::=   0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

a. A Java character literal

<character literal> ::= ‘<letter> ‘| ‘<digit>’

b. A Java character string literal

<string > ::= ‘<char>’
<char> ::= <letters>| <digit> | <symbol> | ε
<letters> ::= <letter> | <letter> <letters>| ε
<symbol> ::= @ | # | $ | ! | ? | % | & | * |

c. A Java integer literal

<integer literal> ::= <decimal integer literal > | <hex integer literal > | <octal integer literal >
<hex integer literal> ::= <hex numeral> | ε
<decimal integer literal > ::= <decimal numeral > | ε
<octal integer literal > ::= <octal numeral> | ε
<decimal numeral> ::= <digit>| <digits> | ε
<digit> ::= <digit> | <digits> | <symbol>

d. A Java real (floating point) literal

<floating point literal > ::= <digits> | <decimals>
<digits> ::= <digit>| <digit> <digits>
<decimal> ::= <digit> | <digits> <digit>
<sign>::= + | -

e. A Java identifier:

<identifier> ::= <initial>|<initial> <more>
<initial> ::= <letter> |_| | $ | ε
< more> ::= <final>|<more><final>
<final>::= <initial>|<digit>
<letter>::= a|b|…c|A|…Z|
<digit>

f. A Java function declaration (prototype):

<method declaration> ::= <method header><method body>
<method header> ::= <result type> <method declaration> | <result type>

g. A Java `if` statement (you may assume that the nonterminals `<statement>` and `<expression>` are defined elsewhere):

<if statement> ::= if (<expression>) < statement > | if (<expression>). <statement> else <statement >

h. A Java `while` statement (you may assume that the nonterminals `<statement>` and `<expression>` are defined elsewhere):

<While statement> ::= while (<expression>) <statement>

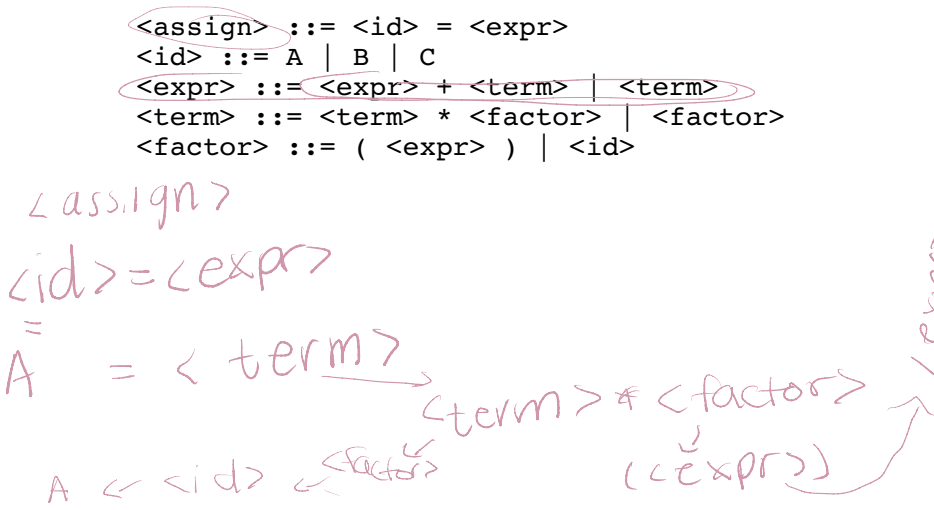
2. Prove that the following grammar is ambiguous:

```
<S> ::= <A>
<A> ::= <A> + <A> | <id>
<id> ::= a | b | c
```


This grammar has two different Parse trees the grammer is ambigious

3. Give a left-most derivation for `A = A * (B + C)` using the following BNF grammar:

```
<assign> ::= <id> = <expr>
<id> ::= A | B | C
<expr> ::= <expr> + <term> | <term>
<term> ::= <term> * <factor> | <factor>
<factor> ::= ( <expr> ) | <id>
```



<assign>

<Id>=<expr>
A = <term>
<term>*<factor>
A = <term>
<term><factor>
A<ID><factor>
<expression>
<expr>+<term>
B<ID>+<term>

Turn in. Using any text editor or word processor, make an electronic copy of this page and write your name prominently at the top. Then enter your solutions directly on that electronic copy. When you are finished, copy it into your personal folder in `/home/cs/214/current/`.

This page maintained by Joel Adams.