/\* CS 214 Project 2: Practicing with BNFs

\* Name: ZeAi Sun

\* Date: Jan 23, 2023

\*/

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

1. A Java character literal

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

<symbols> ::= , | . | / | < | > | ? | : | ; | { | } | [ | ] | ! | @ | # | $ | % | ^ | & | \* | ( | ) | - | \_ | + | = | ~ (I’m not sure if this is all allowed symbols but these should cover most of them)

1. A Java character string literal

<character string> ::= “<first\_letter> <valid\_sequence>”

<first\_letter> ::= <letter> | \_

<valid\_sequence> ::= <character> | <character> <valid\_sequence> | Ø

(<charater> is a terminal here as defined in part a)

1. A Java integer literal

<integer> ::= <symbol> <digits>

<symbol> ::= + | -

<digits> ::= <digit> | <digit> <digits>

1. A Java real (floating point) literal

<floating point> ::= <integer> . <digits> <type>

(<integer> and <digits> are terminals here as defined in part c)

<type> ::= d | f

1. A Java identifier:

<identifier> ::= <first\_letter> <valid\_sequence>

<first\_letter> ::= <lower\_letter> | \_ | $

<lower\_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

<valid\_sequence> ::= <valid\_symbol> <valid\_sequence> | Ø

<valid\_symbol> = <letter> | <digit> | \_ | $

1. A Java function declaration (prototype):

<function> ::= <function\_header> <function\_body>

<function\_header> ::= <modifier> <return\_type> <function\_name> (<parameters>)

<modifier> ::= public | private | static | (and more, but not all listed)

<return\_type> ::= void | string | int | float | long | (and more, but not all listed)

<function\_name> ::= <identifier> (as defined in part e)

<parameters> ::= <parameter> | <paratemer> , <paratemers> | Ø

<parameter> ::= <return\_type> <parameter\_name>

<parameter\_name> ::= <identifier> (as defined in part e)

<function\_body> ::= <code block>

1. A Java if statement (<statement> and <expression> are defined elsewhere):

<if\_stmt> ::= if (<expression>) {<statement>} <else\_part>

<else\_part> ::= <else\_if part> <else\_part> | else {<statement>}| Ø

<else\_if part> ::= else if (<expression>) {<statement>}

1. A Java while statement (<statement> and <expression> are defined elsewhere):

<while\_stmt> ::= while (<expression>) {<statement>}

2. Prove that the following grammar is ambiguous:

<S> ::= <A>

<A> ::= <A> + <A> | <id>

<id> ::= a | b | c

We can get two parse trees from this grammar:

A A

/ \ / \

Id A + A A + A id

| | | |

Id + Id Id + Id

But we can’t decide which one is correct, so this grammar is ambiguous.

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>

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