Nome: Hugo Mendes

Professor Raul Ikeda

Lógica da Computação – Insper 2019

EBNF SentenceScript (SS)

```
<type-specifier> ::= sem_devolucao (void)
                  | texto (string)_
                  | inteiro (int)
                  | decimal  (float)

######################################################################

<type-qualifier> ::= constante (const)
                  | mutavel(volatile)

######################################################################


<constant-expression> ::= <conditional-expression>

######################################################################

<conditional-expression> ::= <logical-or-expression>
                           | <logical-or-expression> ? <expression> :
<conditional-expression>

######################################################################

<logical-or-expression> ::= <logical-and-expression>
                          | <logical-or-expression> ou (||) <logical-and-expression>

######################################################################


<logical-and-expression> ::= <inclusive-or-expression>
                           | <logical-and-expression> e (&&) <inclusive-or-expression>

######################################################################

<and-expression> ::= <equality-expression>
                  | <and-expression> e (&&) <equality-expression>

######################################################################


<equality-expression> ::= <relational-expression>
                        | <equality-expression> for (==) <relational-expression>
                        | <equality-expression> diferente (!=)<relational-expression>

######################################################################


<additive-expression> ::= <multiplicative-expression>
                        | <additive-expression> + <multiplicative-expression>
                        | <additive-expression> - <multiplicative-expression>
```

```
#########################################################################

<multiplicative-expression> ::= <cast-expression>
                    | <multiplicative-expression> * <cast-expression>
                    | <multiplicative-expression> / <cast-expression>

#########################################################################

<cast-expression> ::= <unary-expression>
                    | ( <type-name> ) <cast-expression>

#########################################################################


<unary-expression> ::= <postfix-expression>
                     | ++ <unary-expression>
                     | -- <unary-expression>
                     | <unary-operator> <cast-expression>
                     | sizeof <unary-expression>
                     | sizeof <type-name>

#########################################################################

<postfix-expression> ::= <primary-expression>
                       | <postfix-expression> [ <expression> ]
                       | <postfix-expression> ( {<assignment-
expression>}* )
                       | <postfix-expression> . <identifier>
                       | <postfix-expression> -> <identifier>
                       | <postfix-expression> ++
                       | <postfix-expression> --

#########################################################################


<primary-expression> ::= <identifier>
               | <constant>
               | <texto(string)>
               | ( <expression> )

#########################################################################


<constant> ::= <constante-inteira (integer-constant)>
         | <constante-texto (character-constant)>
         | <constante-decimal (floating-constant)>

#########################################################################


<expression> ::= <assignment-expression>
          | <expression> , <assignment-expression>

#########################################################################


<assignment-expression> ::= <conditional-expression>
                 | <unary-expression> <assignment-expression>
```

```
##########################################################################


<unary-operator> ::= &
                  | *
                  | +
                  | -
                  | ~
                  | !

##########################################################################

<expression-statement> ::= {<expression>}? ;

##########################################################################


<selection-statement> ::= se(if) ( <expression> ) <statement>
                       | se(if) ( <expression> ) <statement> entao <statement>
                       | troque_para(switch) ( <expression> ) <statement>

##########################################################################


<iteration-statement> ::= enquanto(while) ( <expression> ) <statement>
                       | execute (do) <statement> enquanto(while) ( <expression> ) ;
                       | para_cada(for) ( {<expression>}? ; {<expression>}? ; {<expression>}? )
<statement>
```