Analisador Léxico

Construção de Compiladores

INE5426

Anna Victoria Oikawa
José Luis Ruas
Matteus Legat

Professor Ricardo Azambuja Silveira
Universidade Federal de Santa Catarina

Descrição da Linguagem (EBNF):

```
Grammar  ::= Function* /*| (Declaration ('=' Expression)?)*/
Function ::= Type Identifier ('[' ']')* '(' Parameters ')' '{' Command* '}'
Type ::= int | string | void | boolean | double | object
Identifier ::= [a-z]+[a-zA-Z0-9_]*
Parameters ::= ((Declaration ',')*Declaration)?
SimpleCommand ::= (Expression | Declaration '=' Expression | Declaration | Identifier
'=' Expression)
Command ::= SimpleCommand ';' | While | If | For | (break | return) ';'
Declaration ::= Type Identifier ('[' [0-9] ']')*
Expression ::= Unary Expression | FunctionCall | Expression Binary Expression |
'('Expression')' | Value | Identifier ('[' [0-9]+ ']')* ('.' (Identifier ('[' [0-9]+
']')*))*

Number ::= [0-9]+ | [0-9]+ '.' [0-9]* | '.' [0-9]+
Value ::= Number | true | false | StringLiteral | Array | Object
Array ::= '[' ((Expression ',')* Expression)? ']'
ObjectElement ::= Declaration ':' Expression
Object ::= '{' ((ObjectElement ',')* ObjectElement)? '}'

Unary ::= '!' | Signal
Binary ::= Signal | '*' | '/' | '%' | '>' | '<' | '>=' | '<=' | '==' | '!=' | '&&' |
'||'
Signal ::= '+' | '-'

FunctionCall ::= Identifier '('Arguments')'
Arguments ::= ((Expression ',')*Expression)?

Whitespace ::= S | Comment
S      ::= ' ' | '\r' | '\n' | '\t'
Comment  ::= '/*' ( [^*] | '*'+ [^*/] )* '*'* '*/' | '/' '/' .* '\n'

StringLiteral ::= '"' (.-('"' | '\\' | '\r' | '\n') | '\\' ('"' | '\\' | 'n' | 'r' |
't'))* '"'

While ::= 'while' '(' Expression ')' '{' Command* '}'
For ::= 'for' '(' SimpleCommand? ';' Expression? ';' SimpleCommand? ')' '{' Command*
'}'
Switch ::= 'switch' '(' Identifier ('[' [0-9]+ ']')* ')' '{' Case* Default Case* '}'
Case ::= 'case' '(' Expression ')' '{' Command* '}'
Default ::= 'default' '{' Command* '}'
```

```
If ::= 'if' '(' Expression ')' '{' Command* '}' ('else' 'if' '(' Expression ')' '{' Command* '}')* ( 'else' '{' Command* '}')?
```

Descrição dos Grafos de Sintaxe:

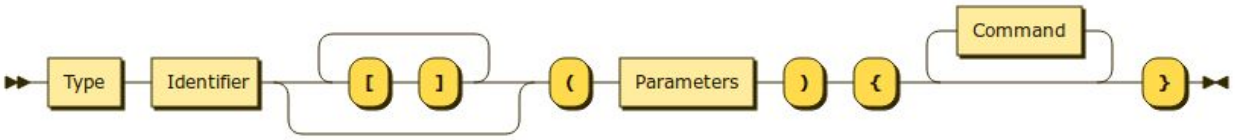
Grammar:


```
Grammar  ::= Function*
```

no references

Uma vez que a linguagem é baseada em funções, o código reconhecido por ela é uma sequência de funções como mostra o grafo acima.

Function:



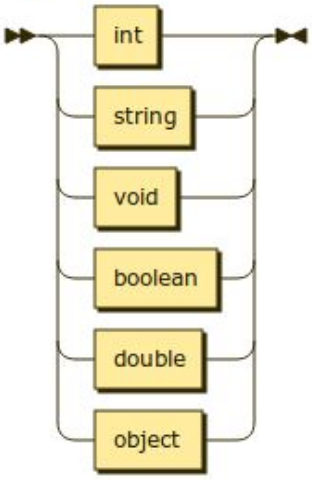

```
Function ::= Type Identifier ( '[' ']' )* '(' Parameters ')' '{' Command* '}'
```

referenced by:

- Grammar

Uma função possui um tipo de **retorno**, seguido por um **identificador**, que pode ser seguido ou não por *colchetes* para indicar que o seu retorno é um *array*. Após o identificador, os parâmetros a serem recebidos é uma lista de **identificados** entre parênteses. A função é finalizada com um *bloco de comandos*, delimitado por **chaves**.

Type:



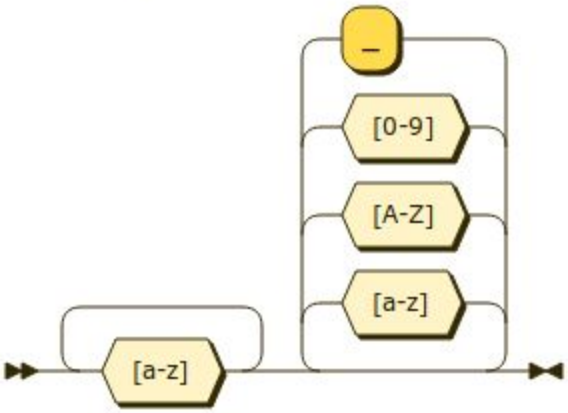

```
Type     ::= int
           | string
           | void
           | boolean
           | double
           | object
```

referenced by:

- Declaration
- Function

Os tipos aceitos pela linguagem são: “int”, “string”, “void”, “boolean”, “double” e “object”. O tipo “object” representa um grupo de valores, cada um com um tipo, um identificador único e um valor.

Identifier:



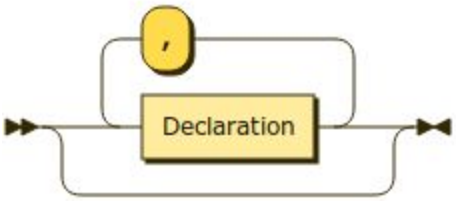

```
Identifier
        ::= [a-z]+ [a-zA-Z0-9_]*
```

referenced by:

- Command
- Declaration
- Expression
- FunctionCall

Os identificadores reconhecidos pela linguagem devem começar obrigatoriamente com letra minúscula, podendo ser seguida por 0 ou mais dígitos, números ou o sinal '_'.

Parameters:



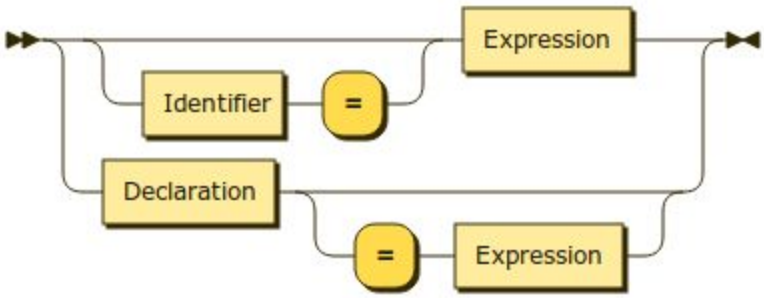

```
Parameters
        ::= ( Declaration ( ',' Declaration )* )?
```

referenced by:

- Function

A estrutura de passagem de parâmetros é dada por um parênteses, uma lista de declarações e um parênteses para fechar a expressão.

SimpleCommand:




```
SimpleCommand
         ::= ( Identifier '=' )? Expression
           | Declaration ( '=' Expression )?
```

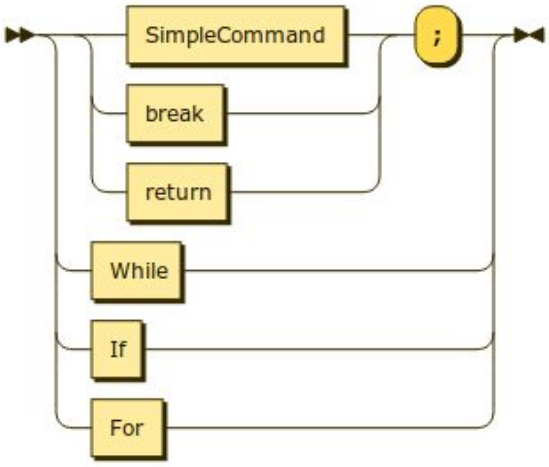
referenced by:

- Command
- For

Um comando é dado por:

- Um identificador seguido do sinal de atribuição ‘ = ‘ e uma expressão como no caso:
  a = 10
- Uma expressão:
  a + 2
- Uma declaração seguida de ‘ = ‘ e expressão:
  int a = 10

Command:




```
Command  ::= ( SimpleCommand | break | return ) ';'
           | While
           | If
           | For
```

referenced by:

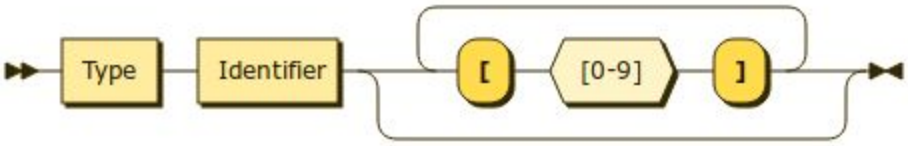
- Case
- Default
- For
- Function
- If
- While

Um comando é dado por um comando simples (definido no grafo de sintaxe anterior) ou de:

- break (sai de loops)
- return (retorno de função)
- Loops:
    - while
    - if
    - for

E finalizado com um ‘ ; ’.

Declaration:



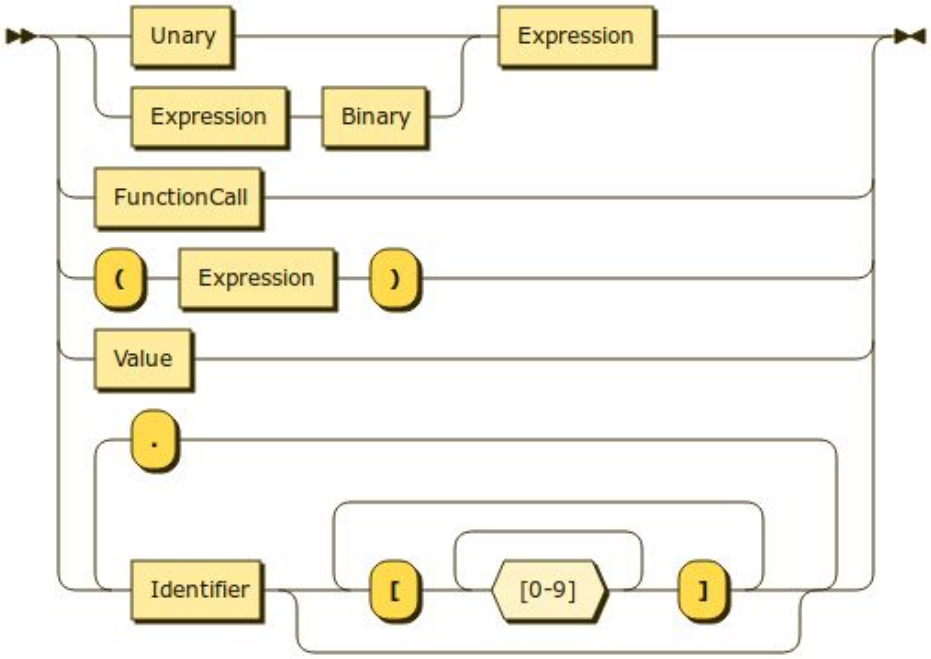

```
Declaration
        ::= Type Identifier ( '[' [0-9] ']' )*
```

referenced by:

- ObjectElement
- Parameters
- SimpleCommand

Uma declaração é composta por um tipo seguido de um identificador, o qual pode ou não estar seguido de colchetes com um valor numérico, que é utilizado no caso de a variável ser um array para definir o seu tamanho.

Expression:




```
Expression
        ::= ( Unary | Expression Binary ) Expression
          | FunctionCall
          | '(' Expression ')'
          | Value
          | Identifier ( '[' [0-9]+ ']' )* ( '.' Identifier ( '[' [0-9]+ ']' )* )*
```

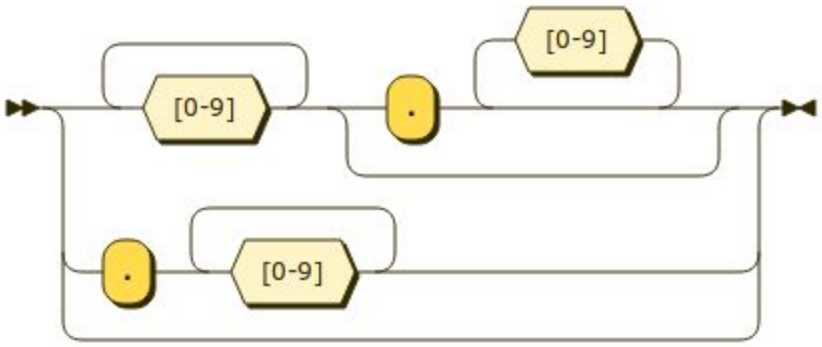
referenced by:

- Arguments
- Case
- Expression
- For
- If
- ObjectElement
- SimpleCommand
- While

Uma expressão é dada por:

- Símbolo unário seguido de expressão: -10
- Expressão com símbolo e outra expressão: a + b
- Uma expressão entre parênteses (2 * 1)
- Um valor: 10
- Um array multidimensional ou unidimensional: array_example[2][2]
- Acesso de uma variável que pertence ao elemento que está sendo acessado no array: array_example[2].value

Number:



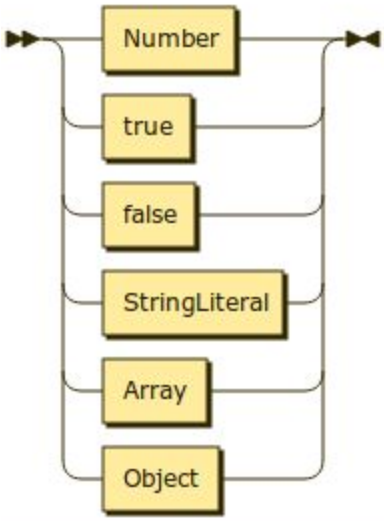

```
Number   ::= ( [0-9]+ ( '.' [0-9]* )? | '.' [0-9]+ )?
```

referenced by:

- Value

Um número é um valor numérico decimal, podendo ser ou não um número fracionário.

Value:




```
Value    ::= Number
           | true
           | false
           | StringLiteral
           | Array
           | Object
```

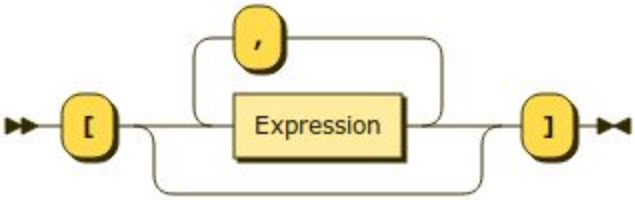
referenced by:

- Array
- Expression

Os valores da linguagem são:

- true, false: booleanos
- Literal de string
- Um array
- Um elemento do tipo Object da linguagem

Array:



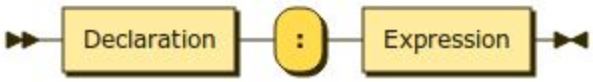

```
Array    ::= '[' ( Expression ( ',' Expression )* )? ']'
```

referenced by:

- Value

Um array literal é definido entre colchetes, contendo uma sequência de expressões separadas por virgula;

ObjectElement:



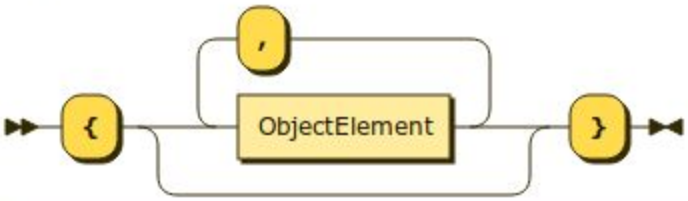

```
ObjectElement
        ::= Declaration ':' Expression
```

referenced by:

- Object

Um elemento do objeto é uma declaração (tipo e identificador) com o seu valor atribuído por uma expressão após dois pontos.

Object:




```
Object   ::= '{' ( ObjectElement ( ',' ObjectElement )* )? '}'
```

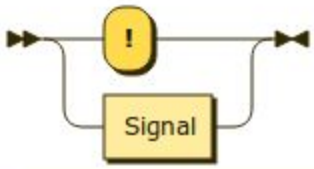
referenced by:

- Value

Um objeto é definido por uma sequência de elementos de objeto separadas por vírgula. Um objeto contém 0 ou mais valores com identificador e tipo definidos. A sua notação é baseada na notação JSON (JavaScript), com a **principal diferença** de incluir o tipo da variável.
Exemplo de declaração de objeto:

```
object joao = {
  string nome : "João Silva",
  int idade : 18,
  object carro : {
    string marca : "Volkswagen",
    string modelo : "gol",
    int ano : 2016
  }
};
```

Unary:




```
Unary    ::= '!'
           | Signal
```

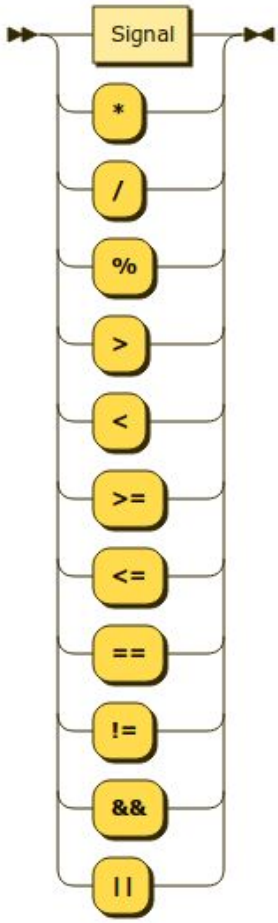
referenced by:

- Expression

Os operadores unários são:

- Negação: !
- Sinal: + ou -

Binary:

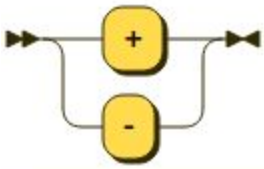



```
Binary   ::= Signal
           | '*'
           | '/'
           | '%'
           | '>'
           | '<'
           | '>='
           | '<='
           | '=='
           | '!='
           | '&&'
           | '||'
```

referenced by:

- Expression

Signal:




```
Signal   ::= '+'
           | '-'
```

referenced by:

- Binary
- Unary

FunctionCall:

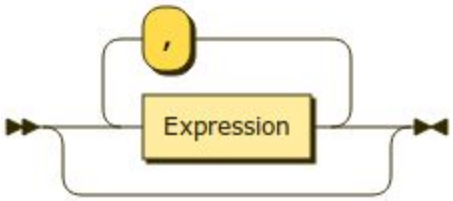

```
FunctionCall
         ::= Identifier '(' Arguments ')'
```

referenced by:

- Expression

A chamada de função é dada pelo identificador da mesma seguido de uma lista de argumentos entre parênteses: funcao(a, b)

Arguments:



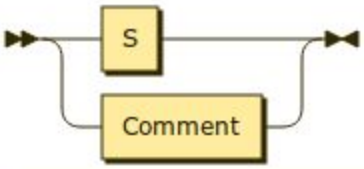

```
Arguments
         ::= ( Expression ( ',' Expression )* )?
```

referenced by:

- FunctionCall

Argumentos podem ser uma lista de expressões: (a, b, c)

Whitespace:

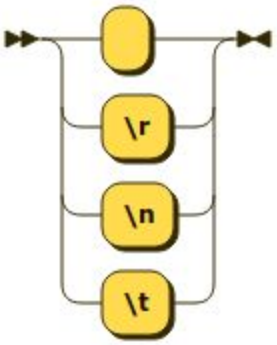



```
Whitespace
          ::= S
            | Comment
```

referenced by:

- Declaration

S:

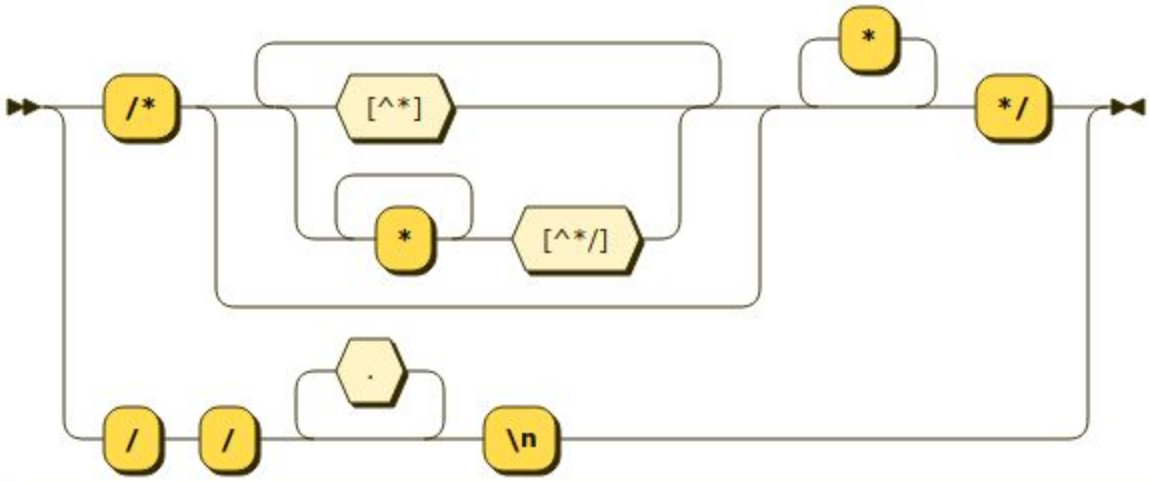



```
S         ::= ' '
            | '\r'
            | '\n'
            | '\t'
```

referenced by:

- Whitespace

Comment:



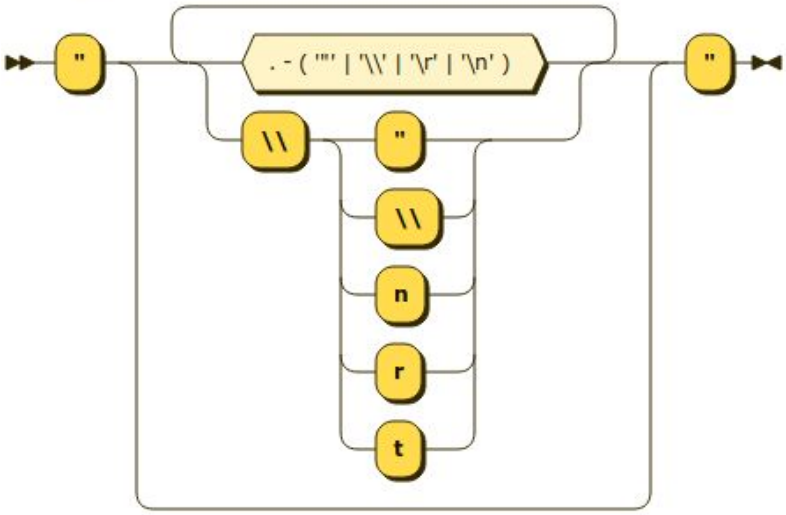

```
Comment  ::= '/*' ( [^*] | '*'+ [^*/] )* '*'* '*/'
           | '/' '/' .* '\n'
```

referenced by:

- Whitespace

A linguagem aceita comentários de linha e de bloco.

StringLiteral:



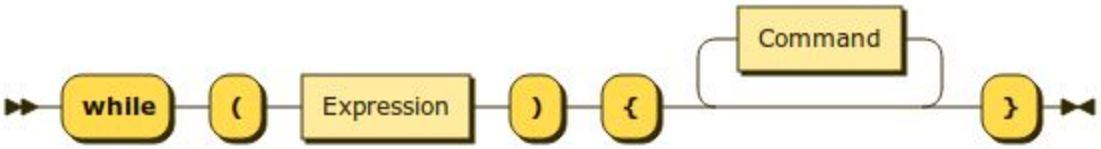

```
StringLiteral
        ::= '"' ( . - ( '"' | '\\' | '\r' | '\n' ) | '\\' ( '"' | '\\' | 'n' | 'r' | 't' ) )* '"'
```

referenced by:

- Value

Strings são dadas por valores dentro de aspas duplas ‘ “ ‘ e não é permitido quebra de linha.

While:



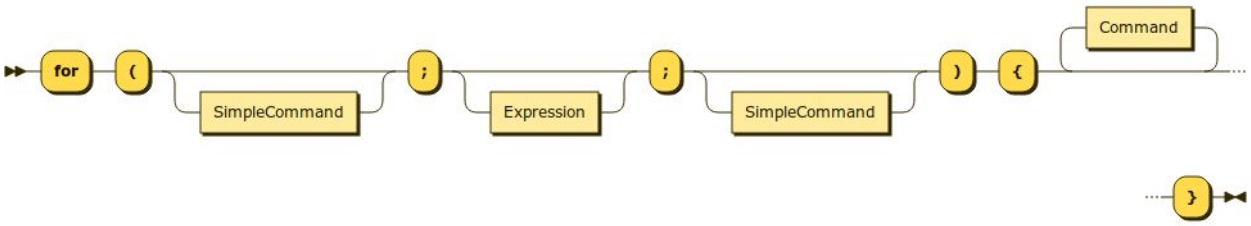

```
While    ::= 'while' '(' Expression ')' '{' Command* '}'
```

referenced by:

- Command

O comando while possui uma expressão que é a condição de entrada e permanência no loop, seguida de comandos dentro de um bloco de chaves.

For:



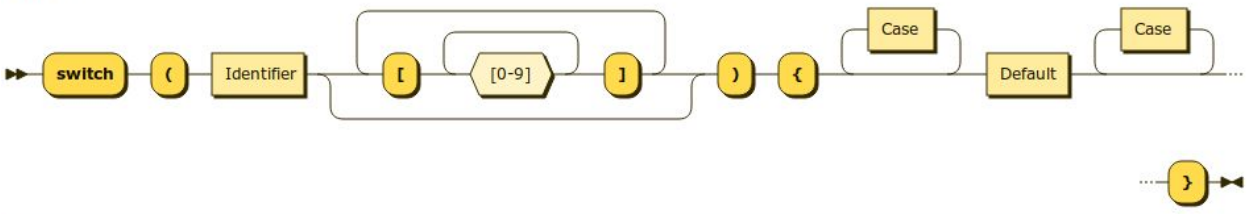

```
For      ::= 'for' '(' SimpleCommand? ';' Expression? ';' SimpleCommand? ')' '{' Command* '}'
```

referenced by:

- Command

O **for** possui a mesma estrutura de Java, mas na nossa linguagem o uso de chaves é obrigatório para delimitar o bloco de comandos.

Switch:

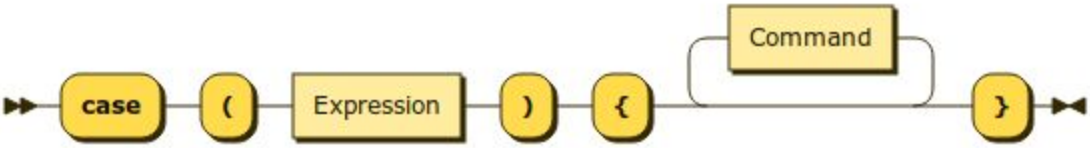



```
Switch   ::= 'switch' '(' Identifier ( '[' [0-9]+ ']' )* ')' '{' Case* Default Case* '}'
```

no references

- O **switch** é como em Java.
    - Mas nesta linguagem, não se usa break para sair de um case.
- O case é delimitado por chaves, não executando o próximo case.
- O uso de default é obrigatório.

Case:

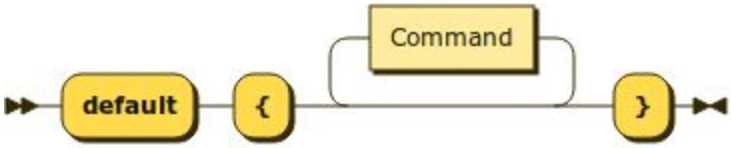



```
Case     ::= 'case' '(' Expression ')' '{' Command* '}'
```

referenced by:

- Switch

Default:



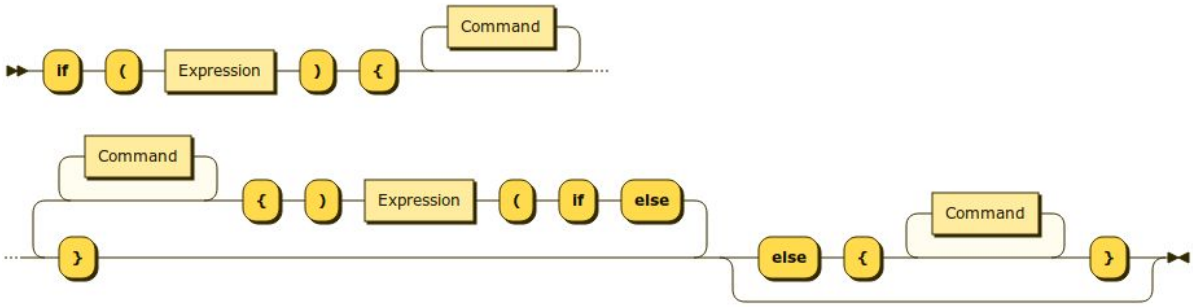

```
Default  ::= 'default' '{' Command* '}'
```

referenced by:

- Switch

O default é obrigatório na estrutura do **switch case**.

If:




```
If      ::= 'if' '(' Expression ')' '{' Command* '}' ( 'else' 'if' '(' Expression ')' '{' Command* '}' )* ( 'else' '{' Command* '}' )?
```

referenced by:

- Command

- A estrutura do if é como em Java e mostrada no exemplo abaixo:
    - if (a > b) {
      a = a - 1;
      }

    - A diferença é que o uso de chaves é obrigatório.

Descrição da Linguagem (ANTLR 4):

```
lexer grammar AMZ_lex;

// Binary Operators
GREATER : '>' ;
GREATEREQUAL : '>=' ;
LESS : '<' ;
LESSEQUAL : '<=' ;
EQUAL : '==' ;
NOTEQUAL : '!=' ;


// Arithmetic
ADD : '+' ;
MINUS : '-' ;
STAR : '*' ;
SLASH : '/' ;
MOD : '%' ;


// Logic
NOT : '!' ;
AND : '&&' ;
OR : '||' ;


// Special symbols
LCURLY : '{' ;
RCURLY : '}' ;
LSQUARE : '[' ;
RSQUARE : ']' ;
LPAREN : '(';
RPAREN : ')' ;
EQUALS : '=' ;
SEMICO : ';' ;
DOT : '.';
COMMA : ',';

// Reserved words
WHILE : 'while' ;
FOR : 'for' ;
```

```
SWITCH : 'switch' ;
CASE : 'case' ;
DEFAULT: 'default' ;
IF : 'if' ;
ELSE : 'else' ;
BREAK : 'break' ;
RETURN : 'return' ;

// Types
INT : 'int' ;
BOOLEAN : 'boolean' ;
STRING : 'string' ;
DOUBLE : 'double' ;
VOID : 'void' ;
OBJECT: 'object';
TYPE : ( INT | BOOLEAN | STRING | DOUBLE | VOID | OBJECT ) ;


// Skip spaces, tabs, newline, comments
WHITESPACE : [ \t\r\n]+ -> skip ; // skip spaces, tabs, newlines
SINGLE_LINE_COMMENT : '//' ~[\n]* '\n' -> skip ;
MULTI_LINE_COMMENT : '/*' ( ~'*' | '*'+ ~[*/] )* '*'* '*/' -> skip ;


// Values
fragment DIGIT : [0-9] ;
NUMBER : DIGIT+ | DIGIT+ '.' DIGIT* | '.' DIGIT+ ;
TRUE : 'true' ;
FALSE : 'false' ;
STRING_LITERAL : ('"' (~('"' | '\\' | '\r' | '\n') | '\\' ('"' | '\\' | 'n' | 'r' |
't'))* '"') ;


// Identifiers
fragment LETTER_LOWER : [a-z] ;
fragment LETTER_UPPER : [A-Z] ;
ID : LETTER_LOWER ( LETTER_LOWER | LETTER_UPPER | DIGIT | '_' )* ;


// Used for parsing
ESCAPE_CHAR : '\\' 't' // two char of lookahead needed,
    |   '\\' 'n' ; // due to common left-prefix
```

Programas de Exemplo:

EXEMPLO 1:

```
void main() {
    int a = 1;
    int b = 2;
    int c = a + b; // 3
    int d = (a + b) * c; /* 9 */
    while (d > 0) {
        print(d);
        d = d - 1;
        if (d % 2 == 0) {
            print ("PAR");
        } else {
            print ("IMPAR");
        }
    }

    print (six());
}


/*
 * Return six
 * @return int six
 */
int six() {
    return (6);
}
```

TOKENS RECONHECIDOS:

```
[@0,0:3='void',<'void'>,1:0]
[@1,5:8='main',<ID>,1:5]
[@2,9:9='(',<'('>,1:9]
[@3,10:10=')',<')'>,1:10]
[@4,12:12='{',<'{'>,1:12]
[@5,18:20='int',<'int'>,2:4]
[@6,22:22='a',<ID>,2:8]
[@7,24:24='=',<'='>,2:10]
[@8,26:26='1',<NUMBER>,2:12]
[@9,27:27=';',<';'>,2:13]
[@10,33:35='int',<'int'>,3:4]
[@11,37:37='b',<ID>,3:8]
[@12,39:39='=',<'='>,3:10]
[@13,41:41='2',<NUMBER>,3:12]
[@14,42:42=';',<';'>,3:13]
[@15,48:50='int',<'int'>,4:4]
[@16,52:52='c',<ID>,4:8]
[@17,54:54='=',<'='>,4:10]
[@18,56:56='a',<ID>,4:12]
[@19,58:58='+',<'+'>,4:14]
[@20,60:60='b',<ID>,4:16]
[@21,61:61=';',<';'>,4:17]
[@22,72:74='int',<'int'>,5:4]
[@23,76:76='d',<ID>,5:8]
[@24,78:78='=',<'='>,5:10]
[@25,80:80='(',<'('>,5:12]
[@26,81:81='a',<ID>,5:13]
[@27,83:83='+',<'+'>,5:15]
[@28,85:85='b',<ID>,5:17]
[@29,86:86=')',<')'>,5:18]
[@30,88:88='*',<'*'>,5:20]
[@31,90:90='c',<ID>,5:22]
[@32,91:91=';',<';'>,5:23]
[@33,105:109='while',<'while'>,6:4]
[@34,111:111='(',<'('>,6:10]
[@35,112:112='d',<ID>,6:11]
[@36,114:114='>',<'>'>,6:13]
[@37,116:116='0',<NUMBER>,6:15]
[@38,117:117=')',<')'>,6:16]
[@39,119:119='{',<'{'>,6:18]
[@40,129:133='print',<ID>,7:8]
[@41,134:134='(',<'('>,7:13]
[@42,135:135='d',<ID>,7:14]
[@43,136:136=')',<')'>,7:15]
[@44,137:137=';',<';'>,7:16]
[@45,147:147='d',<ID>,8:8]
[@46,149:149='=',<'='>,8:10]
[@47,151:151='d',<ID>,8:12]
[@48,153:153='-',<'-'>,8:14]
[@49,155:155='1',<NUMBER>,8:16]
[@50,156:156=';',<';'>,8:17]
[@51,166:167='if',<'if'>,9:8]
[@52,169:169='(',<'('>,9:11]
[@53,170:170='d',<ID>,9:12]
[@54,172:172='%',<'%'>,9:14]
[@55,174:174='2',<NUMBER>,9:16]
[@56,176:177='==',<'=='>,9:18]
[@57,179:179='0',<NUMBER>,9:21]
[@58,180:180=')',<')'>,9:22]
[@59,182:182='{',<'{'>,9:24]
[@60,196:200='print',<ID>,10:12]
[@61,202:202='(',<'('>,10:18]
[@62,203:207='"PAR"',<STRING_LITERAL>,10:19]
[@63,208:208=')',<')'>,10:24]
[@64,209:209=';',<';'>,10:25]
[@65,219:219='}',<'}'>,11:8]
[@66,221:224='else',<'else'>,11:10]
[@67,226:226='{',<'{'>,11:15]
[@68,240:244='print',<ID>,12:12]
[@69,246:246='(',<'('>,12:18]
[@70,247:253='"IMPAR"',<STRING_LITERAL>,12:19]
[@71,254:254=')',<')'>,12:26]
[@72,255:255=';',<';'>,12:27]
[@73,265:265='}',<'}'>,13:8]
[@74,271:271='}',<'}'>,14:4]
[@75,278:282='print',<ID>,16:4]
[@76,284:284='(',<'('>,16:10]
[@77,285:287='six',<ID>,16:11]
[@78,288:288='(',<'('>,16:14]
[@79,289:289=')',<')'>,16:15]
[@80,290:290=')',<')'>,16:16]
[@81,291:291=';',<';'>,16:17]
[@82,293:293='}',<'}'>,17:0]
[@83,337:339='int',<'int'>,24:0]
[@84,341:343='six',<ID>,24:4]
[@85,344:344='(',<'('>,24:7]
[@86,345:345=')',<')'>,24:8]
[@87,347:347='{',<'{'>,24:10]
[@88,353:358='return',<'return'>,25:4]
[@89,360:360='(',<'('>,25:11]
[@90,361:361='6',<NUMBER>,25:12]
[@91,362:362=')',<')'>,25:13]
[@92,363:363=';',<';'>,25:14]
[@93,365:365='}',<'}'>,26:0]
[@94,367:366='<EOF>',<EOF>,27:0]
```

EXEMPLO 2:

```
void main() {
	int a = 10;
	int b = 2;

	switch (a) {
		case 2 {
			print("Dois");
		}
		case 10 {
			while (b < 5) {
				b = b + 1;
			}

		}
		default {
			print (a % b);
		}
	}
}
```

TOKENS RECONHECIDOS:

```
[@0,0:3='void',<'void'>,1:0]
[@1,5:8='main',<ID>,1:5]
[@2,9:9='(',<'('>,1:9]
[@3,10:10=')',<')'>,1:10]
[@4,12:12='{',<'{'>,1:12]
[@5,16:18='int',<'int'>,2:2]
[@6,20:20='a',<ID>,2:6]
[@7,22:22='=',<'='>,2:8]
[@8,24:25='10',<NUMBER>,2:10]
[@9,26:26=';',<';'>,2:12]
[@10,30:32='int',<'int'>,3:2]
[@11,34:34='b',<ID>,3:6]
[@12,36:36='=',<'='>,3:8]
[@13,38:38='2',<NUMBER>,3:10]
[@14,39:39=';',<';'>,3:11]
[@15,44:49='switch',<'switch'>,5:2]
[@16,51:51='(',<'('>,5:9]
[@17,52:52='a',<ID>,5:10]
[@18,53:53=')',<')'>,5:11]
[@19,55:55='{',<'{'>,5:13]
[@20,60:63='case',<'case'>,6:3]
[@21,65:65='2',<NUMBER>,6:8]
[@22,67:67='{',<'{'>,6:10]
[@23,73:77='print',<ID>,7:4]
[@24,78:78='(',<'('>,7:9]
[@25,79:84='"Dois"',<STRING_LITERAL>,7:10]
[@26,85:85=')',<')'>,7:16]
[@27,86:86=';',<';'>,7:17]
[@28,92:92='}',<'}'>,9:3]
[@29,97:100='case',<'case'>,10:3]
[@30,102:103='10',<NUMBER>,10:8]
[@31,105:105='{',<'{'>,10:11]
[@32,111:115='while',<'while'>,11:4]
[@33,117:117='(',<'('>,11:10]
[@34,118:118='b',<ID>,11:11]
[@35,120:120='<',<'<'>,11:13]
[@36,122:122='5',<NUMBER>,11:15]
[@37,123:123=')',<')'>,11:16]
[@38,125:125='{',<'{'>,11:18]
[@39,132:132='b',<ID>,12:5]
[@40,134:134='=',<'='>,12:7]
[@41,136:136='b',<ID>,12:9]
[@42,138:138='+',<'+'>,12:11]
[@43,140:140='1',<NUMBER>,12:13]
```

```
[@44,141:141=';',<';'>,12:14]
[@45,147:147='}',<'}'>,13:4]
[@46,153:153='}',<'}'>,15:3]
[@47,158:164='default',<'default'>,16:3]
[@48,166:166='{',<'{'>,16:11]
[@49,172:176='print',<ID>,17:4]
[@50,178:178='(',<'('>,17:10]
[@51,179:179='a',<ID>,17:11]
[@52,181:181='%',<'%'>,17:13]
[@53,183:183='b',<ID>,17:15]
[@54,184:184=')',<')'>,17:16]
[@55,185:185=';',<';'>,17:17]
[@56,190:190='}',<'}'>,18:3]
[@57,194:194='}',<'}'>,19:2]
[@58,196:196='}',<'}'>,20:0]
[@59,198:197='<EOF>',<EOF>,21:0]
```

EXEMPLO 3 - erro léxico:

```
// Can't recognize 'Wrong'

/* For */
void main() {
	string Wrong = "Upper case";
	int b = 20;
	for (; b > 0; b = b-1 ) {
		print ("a\nb");
	}
}
```

TOKENS RECONHECIDOS

```
line 6:8 token recognition error at: 'W'
[@0,38:41='void',<'void'>,4:0]
[@1,43:46='main',<ID>,4:5]
[@2,47:47='(',<'('>,4:9]
[@3,48:48=')',<')'>,4:10]
[@4,50:50='{',<'{'>,4:12]
[@5,55:60='string',<'string'>,6:1]
[@6,63:66='rong',<ID>,6:9]
[@7,68:68='=',<'='>,6:14]
[@8,70:81='"Upper
case"',<STRING_LITERAL>,6:16]
[@9,82:82=';',<';'>,6:28]
[@10,85:87='int',<'int'>,7:1]
[@11,89:89='b',<ID>,7:5]
[@12,91:91='=',<'='>,7:7]
[@13,93:94='20',<NUMBER>,7:9]
[@14,95:95=';',<';'>,7:11]
[@15,98:100='for',<'for'>,8:1]
[@16,102:102='(',<'('>,8:5]
[@17,103:103=';',<';'>,8:6]
[@18,105:105='b',<ID>,8:8]
[@19,107:107='>',<'>'>,8:10]
[@20,109:109='0',<NUMBER>,8:12]
[@21,110:110=';',<';'>,8:13]
[@22,112:112='b',<ID>,8:15]
[@23,114:114='=',<'='>,8:17]
[@24,116:116='b',<ID>,8:19]
[@25,117:117='-',<'-'>,8:20]
[@26,118:118='1',<NUMBER>,8:21]
[@27,120:120=')',<')'>,8:23]
[@28,122:122='{',<'{'>,8:25]
[@29,126:130='print',<ID>,9:2]
[@30,132:132='(',<'('>,9:8]
[@31,133:138='"a\nb"',<STRING_LITERAL>,9:9]
[@32,139:139=')',<')'>,9:15]
[@33,140:140=';',<';'>,9:16]
[@34,143:143='}',<'}'>,10:1]
[@35,146:146='}',<'}'>,12:0]
[@36,148:147='<EOF>',<EOF>,13:0]
```

EXEMPLO 4:

```
/*
* Comment
*/

void main() {

	int a = 20;
	boolean bool = true;
	int b = 10;
	while (a > 3) {
		if (a / 2 != 6) {
			print (a);
		} else if (a == 10) {
			print ("dez");
		} else {
			print ("default)";
		}
	}

	if (bool) {
		bool = false;
	}

	if ((b % 2 != 0) && (!bool)) {
		b = a;
	}

}
```

TOKENS RECONHECIDOS:

```
[@0,17:20='void',<'void'>,5:0]
[@1,22:25='main',<ID>,5:5]
[@2,26:26='(',<'('>,5:9]
[@3,27:27=')',<')'>,5:10]
[@4,29:29='{',<'{'>,5:12]
[@5,34:36='int',<'int'>,7:1]
[@6,38:38='a',<ID>,7:5]
[@7,40:40='=',<'='>,7:7]
[@8,42:43='20',<NUMBER>,7:9]
[@9,44:44=';',<';'>,7:11]
[@10,47:53='boolean',<'boolean'>,8:1]
[@11,55:58='bool',<ID>,8:9]
[@12,60:60='=',<'='>,8:14]
[@13,62:65='true',<'true'>,8:16]
[@14,66:66=';',<';'>,8:20]
[@15,69:71='int',<'int'>,9:1]
[@16,73:73='b',<ID>,9:5]
[@17,75:75='=',<'='>,9:7]
[@18,77:78='10',<NUMBER>,9:9]
[@19,79:79=';',<';'>,9:11]
```

```
[@20,82:86='while',<'while'>,10:1]
[@21,88:88='(',<'('>,10:7]
[@22,89:89='a',<ID>,10:8]
[@23,91:91='>',<'>'>,10:10]
[@24,93:93='3',<NUMBER>,10:12]
[@25,94:94=')',<')'>,10:13]
[@26,96:96='{',<'{'>,10:15]
[@27,100:101='if',<'if'>,11:2]
[@28,103:103='(',<'('>,11:5]
[@29,104:104='a',<ID>,11:6]
[@30,106:106='/',<'/'>,11:8]
[@31,108:108='2',<NUMBER>,11:10]
[@32,110:111='!=',<'!='>,11:12]
[@33,113:113='6',<NUMBER>,11:15]
[@34,114:114=')',<')'>,11:16]
[@35,116:116='{',<'{'>,11:18]
[@36,121:125='print',<ID>,12:3]
[@37,127:127='(',<'('>,12:9]
[@38,128:128='a',<ID>,12:10]
[@39,129:129=')',<')'>,12:11]
[@40,130:130=';',<';'>,12:12]
[@41,134:134='}',<'}'>,13:2]
[@42,136:139='else',<'else'>,13:4]
[@43,141:142='if',<'if'>,13:9]
[@44,144:144='(',<'('>,13:12]
[@45,145:145='a',<ID>,13:13]
[@46,147:148='==',<'=='>,13:15]
[@47,150:151='10',<NUMBER>,13:18]
[@48,152:152=')',<')'>,13:20]
[@49,154:154='{',<'{'>,13:22]
[@50,159:163='print',<ID>,14:3]
[@51,165:165='(',<'('>,14:9]
[@52,166:170='"dez"',<STRING_LITERAL>,1
4:10]
[@53,171:171=')',<')'>,14:15]
[@54,172:172=';',<';'>,14:16]
[@55,176:176='}',<'}'>,15:2]
[@56,178:181='else',<'else'>,15:4]
[@57,183:183='{',<'{'>,15:9]
[@58,188:192='print',<ID>,16:3]
[@59,194:194='(',<'('>,16:9]
[@60,195:204='"default)"',<STRING_LITER
AL>,16:10]
[@61,205:205=';',<';'>,16:20]
[@62,209:209='}',<'}'>,17:2]
[@63,212:212='}',<'}'>,18:1]
[@64,217:218='if',<'if'>,20:1]
[@65,220:220='(',<'('>,20:4]
[@66,221:224='bool',<ID>,20:5]
[@67,225:225=')',<')'>,20:9]
[@68,227:227='{',<'{'>,20:11]
[@69,231:234='bool',<ID>,21:2]
[@70,236:236='=',<'='>,21:7]
[@71,238:242='false',<'false'>,21:9]
[@72,243:243=';',<';'>,21:14]
[@73,246:246='}',<'}'>,22:1]
[@74,250:251='if',<'if'>,24:1]
[@75,253:253='(',<'('>,24:4]
[@76,254:254='(',<'('>,24:5]
[@77,255:255='b',<ID>,24:6]
[@78,257:257='%',<'%'>,24:8]
[@79,259:259='2',<NUMBER>,24:10]
[@80,261:262='!=',<'!='>,24:12]
[@81,264:264='0',<NUMBER>,24:15]
[@82,265:265=')',<')'>,24:16]
[@83,267:268='&&',<'&&'>,24:18]
[@84,270:270='(',<'('>,24:21]
[@85,271:271='!',<'!'>,24:22]
[@86,272:275='bool',<ID>,24:23]
[@87,276:276=')',<')'>,24:27]
[@88,277:277=')',<')'>,24:28]
[@89,279:279='{',<'{'>,24:30]
[@90,283:283='b',<ID>,25:2]
[@91,285:285='=',<'='>,25:4]
[@92,287:287='a',<ID>,25:6]
[@93,288:288=';',<';'>,25:7]
[@94,291:291='}',<'}'>,26:1]
[@95,295:295='}',<'}'>,28:0]
[@96,298:297='<EOF>',<EOF>,30:0]
```

EXEMPLO 5 - erro léxico:

```
void _wrong_id() {
    int 0abz = 2;
    Az = 3;
}
```

TOKENS RECONHECIDOS:

```
line 1:5 token recognition error at:
'_'
line 3:4 token recognition error at:
'A'
[@0,0:3='void',<'void'>,1:0]
[@1,6:13='wrong_id',<ID>,1:6]
[@2,14:14='(',<'('>,1:14]
[@3,15:15=')',<')'>,1:15]
[@4,17:17='{',<'{'>,1:17]
[@5,23:25='int',<'int'>,2:4]
[@6,27:27='0',<NUMBER>,2:8]
[@7,28:30='abz',<ID>,2:9]
[@8,32:32='=',<'='>,2:13]
[@9,34:34='2',<NUMBER>,2:15]
[@10,35:35=';',<';'>,2:16]
[@11,42:42='z',<ID>,3:5]
[@12,44:44='=',<'='>,3:7]
[@13,46:46='3',<NUMBER>,3:9]
[@14,47:47=';',<';'>,3:10]
[@15,49:49='}',<'}'>,4:0]
[@16,51:50='<EOF>',<EOF>,5:0]
```

EXEMPLO 6 - switch case e for:

```
void main() {
	for (int i = 0; i <= 10; i = i+1) {
	switch (i) {
		case (i < 5) {
		int j = i*2;
		print ("First case: ");
		print (j / 2);
		print ("\n");
		}
		case (i < 10) {
		int j = i+15;
		print ("Second case: ");
		print (j);
		print ("\n");
		}
		default {
			print ("Default");
		}
	}
	}
}
```

TOKENS RECONHECIDOS:

```
[@0,0:3='void',<'void'>,1:0]
[@1,5:8='main',<ID>,1:5]
[@2,9:9='(',<'('>,1:9]
[@3,10:10=')',<')'>,1:10]
[@4,12:12='{',<'{'>,1:12]
[@5,15:17='for',<'for'>,2:1]
[@6,19:19='(',<'('>,2:5]
[@7,20:22='int',<'int'>,2:6]
[@8,24:24='i',<ID>,2:10]
[@9,26:26='=',<'='>,2:12]
[@10,28:28='0',<NUMBER>,2:14]
[@11,29:29=';',<';'>,2:15]
[@12,31:31='i',<ID>,2:17]
[@13,33:34='<=',<'<='>,2:19]
[@14,36:37='10',<NUMBER>,2:22]
[@15,38:38=';',<';'>,2:24]
[@16,40:40='i',<ID>,2:26]
[@17,42:42='=',<'='>,2:28]
[@18,44:44='i',<ID>,2:30]
[@19,45:45='+',<'+'>,2:31]
[@20,46:46='1',<NUMBER>,2:32]
[@21,47:47=')',<')'>,2:33]
[@22,49:49='{',<'{'>,2:35]
[@23,56:61='switch',<'switch'>,3:5]
[@24,63:63='(',<'('>,3:12]
[@25,64:64='i',<ID>,3:13]
[@26,65:65=')',<')'>,3:14]
[@27,67:67='{',<'{'>,3:16]
[@28,78:81='case',<'case'>,4:9]
[@29,83:83='(',<'('>,4:14]
[@30,84:84='i',<ID>,4:15]
[@31,86:86='<',<'<'>,4:17]
[@32,88:88='5',<NUMBER>,4:19]
[@33,89:89=')',<')'>,4:20]
[@34,91:91='{',<'{'>,4:22]
[@35,106:108='int',<'int'>,5:13]
[@36,110:110='j',<ID>,5:17]
[@37,112:112='=',<'='>,5:19]
[@38,114:114='i',<ID>,5:21]
[@39,115:115='*',<'*'>,5:22]
[@40,116:116='2',<NUMBER>,5:23]
[@41,117:117=';',<';'>,5:24]
[@42,132:136='print',<ID>,6:13]
[@43,138:138='(',<'('>,6:19]
[@44,139:152='"First case:
"',<STRING_LITERAL>,6:20]
[@45,153:153=')',<')'>,6:34]
[@46,154:154=';',<';'>,6:35]
[@47,169:173='print',<ID>,7:13]
[@48,175:175='(',<'('>,7:19]
[@49,176:176='j',<ID>,7:20]
[@50,178:178='/',<'/'>,7:22]
[@51,180:180='2',<NUMBER>,7:24]
[@52,181:181=')',<')'>,7:25]
[@53,182:182=';',<';'>,7:26]
[@54,197:201='print',<ID>,8:13]
[@55,203:203='(',<'('>,8:19]
[@56,204:207='"\n"',<STRING_LITERAL>,8:
20]
[@57,208:208=')',<')'>,8:24]
[@58,209:209=';',<';'>,8:25]
[@59,220:220='}',<'}'>,9:9]
[@60,231:234='case',<'case'>,10:9]
[@61,236:236='(',<'('>,10:14]
[@62,237:237='i',<ID>,10:15]
[@63,239:239='<',<'<'>,10:17]
[@64,241:242='10',<NUMBER>,10:19]
[@65,243:243=')',<')'>,10:21]
[@66,245:245='{',<'{'>,10:23]
[@67,260:262='int',<'int'>,11:13]
[@68,264:264='j',<ID>,11:17]
[@69,266:266='=',<'='>,11:19]
[@70,268:268='i',<ID>,11:21]
[@71,269:269='+',<'+'>,11:22]
[@72,270:271='15',<NUMBER>,11:23]
[@73,272:272=';',<';'>,11:25]
[@74,287:291='print',<ID>,12:13]
[@75,293:293='(',<'('>,12:19]
[@76,294:308='"Second case:
"',<STRING_LITERAL>,12:20]
[@77,309:309=')',<')'>,12:35]
[@78,310:310=';',<';'>,12:36]
```

```
[@79,325:329='print',<ID>,13:13]
[@80,331:331='(',<'('>,13:19]
[@81,332:332='j',<ID>,13:20]
[@82,333:333=')',<')'>,13:21]
[@83,334:334=';',<';'>,13:22]
[@84,349:353='print',<ID>,14:13]
[@85,355:355='(',<'('>,14:19]
[@86,356:359='"\n"',<STRING_LITERAL>,14
:20]
[@87,360:360=')',<')'>,14:24]
[@88,361:361=';',<';'>,14:25]
[@89,372:372='}',<'}'>,15:9]
[@90,383:389='default',<'default'>,16:9
]
[@91,391:391='{',<'{'>,16:17]
[@92,407:411='print',<ID>,17:14]
[@93,413:413='(',<'('>,17:20]
[@94,414:422='"Default"',<STRING_LITERA
L>,17:21]
[@95,423:423=')',<')'>,17:30]
[@96,424:424=';',<';'>,17:31]
[@97,435:435='}',<'}'>,18:9]
[@98,442:442='}',<'}'>,19:5]
[@99,445:445='}',<'}'>,20:1]
[@100,447:447='}',<'}'>,21:0]
[@101,449:448='<EOF>',<EOF>,22:0]
```