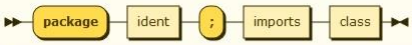
Analisador Sintático

Grupo 5:
Felipe Born
Giovanni Salvador
Vitor Pereira

Gramática:

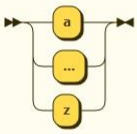
Programa:




```
Programa ::= 'package' ident ';' imports class
```

no references

ident:

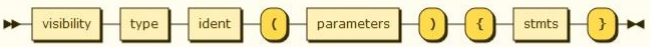



```
ident    ::= 'a'
           | '...'
           | 'z'
```

referenced by:

- Programa
- anything
- atrib
- class
- declare
- functions
- names
- type
- var

functions:

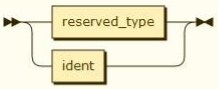



```
functions
        ::= visibility type ident '(' parameters ')' '{' stmts '}'
```

referenced by:

- class

type:

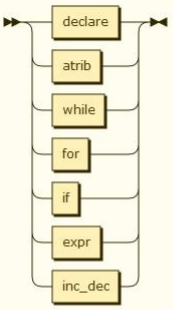



```
type     ::= reserved_type
           | ident
```

referenced by:

- atrib
- declare
- functions
- parameters

stmt:

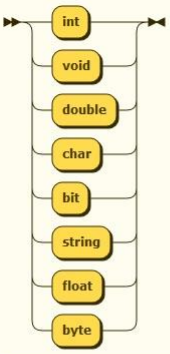



```
stmt     ::= declare
           | atrib
           | while
           | for
           | if
           | expr
           | inc_dec
```

referenced by:

- for
- stmts

reserved_type:

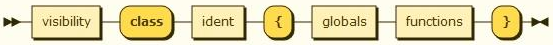



```
reserved_type
        ::= 'int'
          | 'void'
          | 'double'
          | 'char'
          | 'bit'
          | 'string'
          | 'float'
          | 'byte'
```

referenced by:

- type

class:

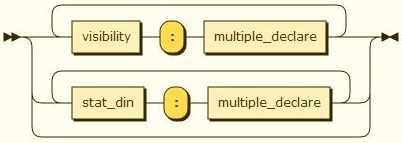



```
class    ::= visibility 'class' ident '{' globals functions '}'
```

referenced by:

- Programa

globals:

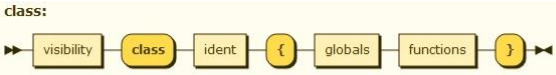



```
globals  ::= ( visibility ':' multiple_declare )*
           | ( stat_din ':' multiple_declare )*
```

referenced by:

- class

class:

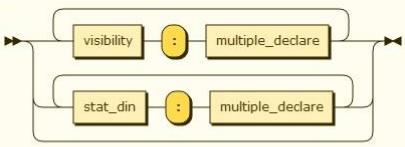



```
class    ::= visibility 'class' ident '{' globals functions '}'
```

referenced by:

- Programa

globals:

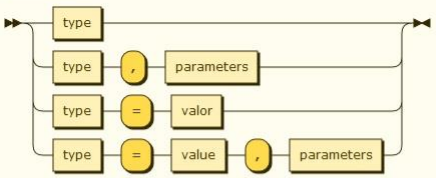



```
globals  ::= ( visibility ':' multiple_declare )*
           | ( stat_din ':' multiple_declare )*
```

referenced by:

- class

parameters:

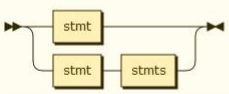



```
parameters
        ::= type
          | type ',' parameters
          | type '=' valor
          | type '=' value ',' parameters
```

referenced by:

- functions
- parameters

stmts:

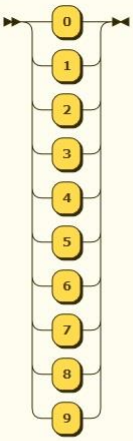



```
stmts    ::= stmt
           | stmt stmts
```

referenced by:

- for
- functions
- if
- stmts
- while

positive_num:




```
positive_num
        ::= '0'
          | '1'
          | '2'
          | '3'
          | '4'
          | '5'
          | '6'
          | '7'
          | '8'
          | '9'
```

referenced by:

- double
- numeric

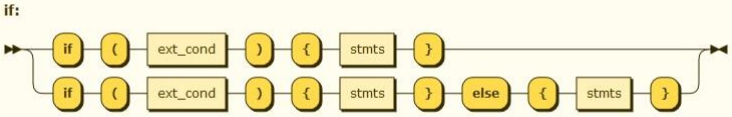
for:


```
for      ::= 'for' '(' stmt ';' ext_cond ';' stmt ')' '{' stmts '}'
```

referenced by:

- stmt

if:

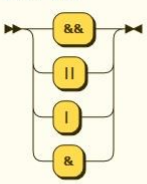



```
if       ::= 'if' '(' ext_cond ')' '{' stmts '}'
           | 'if' '(' ext_cond ')' '{' stmts '}' 'else' '{' stmts '}'
```

referenced by:

- stmt

logic_op:

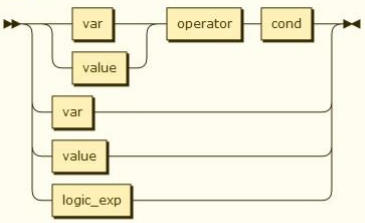



```
logic_op ::= '&&'
           | '||'
           | '|'
           | '&'
```

referenced by:

- ext_cond
- spec_sim

cond:

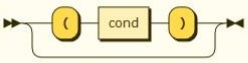



```
cond     ::= ( var | value ) operator cond
           | ( var | value )
           | logic_exp
```

referenced by:

- cond
- ext_cond
- logic_exp

logic_exp:

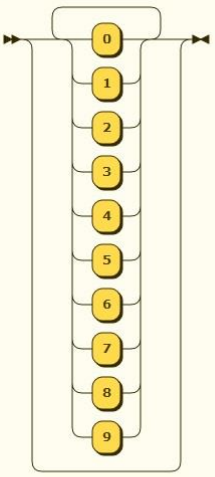



```
logic_exp
         ::= '(' cond ')'
           |
```

referenced by:

- cond

extension_num:




```
extension_num
         ::= ( '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9' )*
```

no references

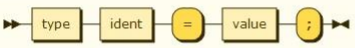
declare:


```
declare  ::= type ident ';'
```

referenced by:

- declare_atrib
- stmt

atrib:

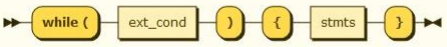



```
atrib    ::= type ident '=' value ';'
```

referenced by:

- declare_atrib
- stmt

while:

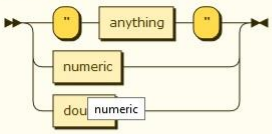



```
while    ::= 'while (' ext_cond ')' '{' stmts '}'
```

referenced by:

- stmt

value:

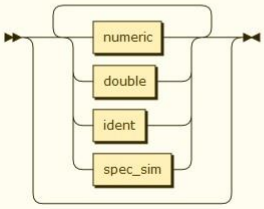



```
value    ::= '"' anything '"'
           | numeric
           | double
```

referenced by:

- atrib
- cond
- exp_p
- parameters

anything:

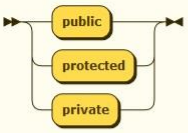



```
anything ::= ( numeric | double | ident | spec_sim )*
```

referenced by:

- value

visibility:

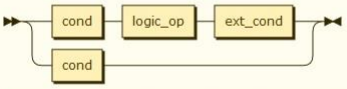



```
visibility
         ::= 'public'
           | 'protected'
           | 'private'
```

referenced by:

- class
- functions
- globals

ext_cond:

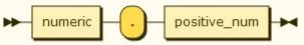



```
ext_cond ::= cond logic_op ext_cond
           | cond
```

referenced by:

- ext_cond
- for
- if
- while

double:

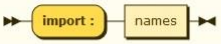



```
double   ::= numeric '.' positive_num
```

referenced by:

- anything
- value

imports:

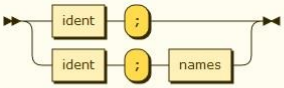



```
imports  ::= 'import :' names
```

referenced by:

- Programa

names:

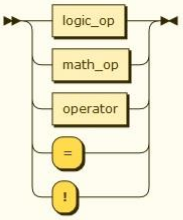



```
names    ::= ident ';'
           | ident ';' names
```

referenced by:

- imports
- names

spec_sim:

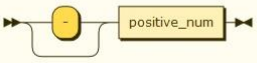



```
spec_sim ::= logic_op
           | math_op
           | operator
           | '='
           | '!'
```

referenced by:

- anything

numeric:

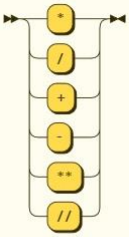



```
numeric  ::= '-'? positive_num
```

referenced by:

- anything
- double
- value

math_op:

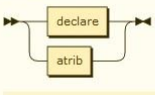



```
math_op  ::= '*'
           | '/'
           | '+'
           | '-'
           | '**'
           | '//'
```

referenced by:

- exp_p
- spec_sim

declare_atrib:

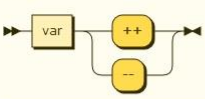



```
declare_atrib
         ::= declare
           | atrib
```

referenced by:

- multiple_declare

inc_dec:

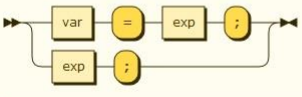



```
inc_dec  ::= var ( '++' | '--' )
```

referenced by:

- stmt

expr:

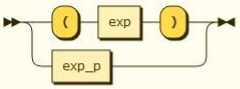



```
expr     ::= var '=' exp ';'
           | exp ';'
```

referenced by:

- stmt

exp:

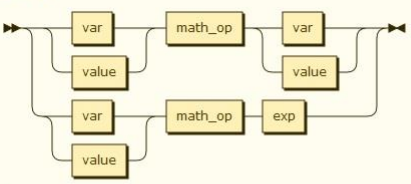



```
exp      ::= '(' exp ')'
           | exp_p
```

referenced by:

- exp
- exp_p
- expr

exp_p:

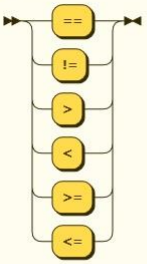



```
exp_p    ::= ( var | value ) math_op ( var | value )
           | ( var | value ) math_op exp
```

referenced by:

- exp

operator:

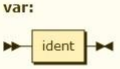



```
operator ::= '=='
           | '!='
           | '>'
           | '<'
           | '>='
           | '<='
```

referenced by:

- cond
- spec_sim

var:




```
var      ::= ident
```

referenced by:

- cond
- exp_p
- expr
- inc_dec

Código:

```
void program() :
{}
{
  < PACKAGE > < ID > < END_STMT > imports() class_def() < EOF >
}

void imports() :
{}
{
  < IMPORT > < POINT_MULTIPLE > imp_def()
}

void imp_def() :
{}
{
  (< STRING_DEF > < END_STMT >)*
}

void class_def() :
{}
{
  LOOKAHEAD(3)
  (
    LOOKAHEAD(3)
    (
      < PROTECTED >
    | < PRIVATE >
    | < PUBLIC >
    )?
    < CLASS > < ID > < LCOL > globals() functions() < RCOL >
  | LOOKAHEAD(3)
    (
      < PROTECTED >
    | < PRIVATE >
    | < PUBLIC >
    )?
    < CLASS > < ID > < LCOL > functions() < RCOL >
  )
|
  (
    < PROTECTED >
  | < PRIVATE >
  | < PUBLIC >
  )?
  < CLASS > < ID > < LCOL > < RCOL >
}

void globals() :
{}
{
  (
    LOOKAHEAD(2)
    (
      (
        (
          < PROTECTED >
        | < PRIVATE >
```

```
        | < PUBLIC >
        | < STATIC >
        | < DYNAMIC >
        )
        < POINT_MULTIPLE >
      )?
      (
        multiple_decl()
      | < POINT_MULTIPLE > multiple_decl()
      )
    )
  )+
}

void multiple_decl() :
{}
{
  (
    LOOKAHEAD(2)
    decl()
    (
      < END_STMT >
    | atrib()
    )?
  )+
}

void decl() :
{}
{
  (
    (
      < INTEGER >
    | < VOID >
    | < DOUBLE >
    | < CHAR >
    | < BIT >
    | < STRING >
    | < FLOAT >
    | < BYTE >
    )?
    < ID >
  )
}

void atrib() :
{}
{
  < ASSIGN >
  (
    < UNS_INT >
  | < INTEIRO >
  | LOOKAHEAD(3)
    < NUMERO >
  | LOOKAHEAD(3)
    < STRING_DEF >
  | LOOKAHEAD(3)
    < ID >
  | LOOKAHEAD(3)
```

```
    expr()
  )
  < END_STMT >
}

void functions() :
{}
{
  (
    function()
  )+
}

void function() :
{}
{
  (
    < PROTECTED >
  | < PRIVATE >
  | < PUBLIC >
  )?
  (
    < INTEGER >
  | < VOID >
  | < DOUBLE >
  | < CHAR >
  | < BIT >
  | < STRING >
  | < FLOAT >
  | < BYTE >
  )
  < ID >
  (
    LOOKAHEAD(2)
    < LPAREN > parameters() < RPAREN >
  | < LPAREN > < RPAREN >
  )
  (
    LOOKAHEAD(2)
    < LCOL > stmts() < RCOL >
  | < LCOL > < RCOL >
  )
}

void parameters() :
{}
{
  (
    decl_param()
    (
      assign_param()
    )?
    (
      < VIRGULA > parameters()
    )?
  )
}

void decl_param() :
```

```
{}
{
  (
    (
      < INTEGER >
    | < VOID >
    | < DOUBLE >
    | < CHAR >
    | < BIT >
    | < STRING >
    | < FLOAT >
    | < BYTE >
    )
    < ID >
  )
}

void assign_param() :
{}
{
  < ASSIGN >
  (
    < UNS_INT >
  | < INTEIRO >
  | < NUMERO >
  | < STRING_DEF >
  | < ID >
  )
}

void stmts() :
{}
{
  (
    stmt()
  )+
}

void stmt() :
{}
{
  (
    LOOKAHEAD(2)
    atribstmt()
  | while_stmt()
  | for_stmt()
  | if_stmt()
  | LOOKAHEAD(3)
    expr()
  | LOOKAHEAD(2)
    inc_dec()
  | func_call()
  )
}

void func_call() :
{}
{
  (
```

```
  < ID > (< PONTO > < ID >)? < LPAREN > send_param() < RPAREN > < END_STMT >
| < THIS > <PONTO > <ID > < LPAREN > send_param() < RPAREN > < END_STMT >  )
}

void send_param() :
{}
{
  (
    < UNS_INT >
  | < INTEIRO >
  | < NUMERO >
  | < STRING_DEF >
  | < ID >
  )
  (
    < VIRGULA > send_param()
  )?
}

void atribstmt() :
{}
{
  (
    LOOKAHEAD(2)
    lvalue() < ASSIGN >
    (
      alocexpression()
    | expression()
    )
    < END_STMT >
  | inc_dec()
  )
}

void lvalue() :
{}
{
  (
    < INTEGER >
  | < VOID >
  | < DOUBLE >
  | < CHAR >
  | < BIT >
  | < STRING >
  | < FLOAT >
  | < BYTE >
  )?
  < ID >
}

void alocexpression() :
{}
{
  < NULL >
}

void expression() :
{}
{
```

```
  numexpr()
  [
    (
      < MENOR >
    | < MAIOR >
    | < EQ >
    | < LE >
    | < GE >
    | < DIF >
    )
    numexpr()
  ]
}

void numexpr() :
{}
{
  term()
  (
    (
      < SOMA >
    | < SUB >
    )
    term()
  )*
}

void term() :
{}
{
  term_n()
  (
    (
      < MULT >
    | < DIV >
    )
    term_n()
  )*
}

void term_n() :
{}
{
  unaryexpr()
  (
    (
      < RAIZ >
    | < EXPO >
    )
    unaryexpr()
  )*
}

void unaryexpr() :
{}
{
  [
    (
      < SOMA >
```

```
    | < SUB >
    )
  ]
  factor()
}

void factor() :
{}
{
  (
    < UNS_INT >
  | < INTEIRO >
  | < NUMERO >
  | < STRING_DEF >
  | < ID >
  | < LPAREN > expression() < RPAREN >
  )
}

void inc_dec() :
{}
{
  < ID >
  (
    < INC >
  | < DEC >
  )
  < END_STMT >
}

void expr() :
{}
{
  exp() < END_STMT >
}

void exp() :
{}
{
  < LPAREN > exp() < RPAREN >
| exp_p()
}

void exp_p() :
{}
{
  (
    (
      < ID >
    | < NUMERO >
    | < STRING_DEF >
    )
    (
      < INC >
    | < DEC >
    | < SOMA >
    | < SUB >
    | < EXPO >
    | < RAIZ >
```

```
    | < MULT >
    | < DIV >
    | < MOD >
    )
    (
      LOOKAHEAD(2)
      < NUMERO >
    | LOOKAHEAD(2)
      < STRING_DEF >
    | LOOKAHEAD(2)
      < ID >
    | LOOKAHEAD(2)
      exp()
    )
  )
}

void while_stmt() :
{}
{
  < WHILE > < LPAREN > ext_cond() < RPAREN > < LCOL > stmts() < RCOL >
}

void for_stmt() :
{}
{
  < FOR > < LPAREN > atribstmt() [ expression() ] < END_STMT > [ atribstmt_noend() ] <
RPAREN > < LCOL > [ stmts() ] < RCOL >
}

void atribstmt_noend() :
{}
{
  (
    LOOKAHEAD(2)
    lvalue() < ASSIGN >
    (
      alocexpression()
    | expression()
    )
  |
    inc_dec()
  )
}

void if_stmt() :
{}
{
  < IF > < LPAREN > ext_cond() < RPAREN > < LCOL > stmts() < RCOL >
  (
    < ELSE > < LCOL > stmts() < RCOL >
  )?
}

void ext_cond() :
{}
{
  cond()
  (
```

```
    (
      < AND >
    | < OR >
    )
    ext_cond()
  )?
}

void cond() :
{}
{
  (
    (
      < UNS_INT >
    | < INTEIRO >
    | < NUMERO >
    | < STRING_DEF >
    | < ID >
    )
    (
      (
        < MENOR >
      | < MAIOR >
      | < EQ >
      | < LE >
      | < GE >
      | < DIF >
      )
      cond()
    )?
  )
| logic_exp()
}

void logic_exp() :
{}
{
  < LPAREN > cond() < RPAREN >
}
```

Programa e Resultado da Análise:

Programa	Resultado

package default;
 	
import:
  "abc.cc";
  "blabla.cc";
 	
public class MyClass
{
 static:
  int a = 1665;
  float b = -9865.501;
 	
 dynamic:
  string s;
 	
 public void setString(string ba)
 {
  s = ba;
  this.doSomething(31.5);
 }
 	
 public void doSomething(double m)
 {
  if(a == m || m != b)
  {
  a = m ** b + (a // (3)) /100.0;
  for(int i = 0; i < b; i = i + (15 / a))
  {
  doSomething(i);
  }
  }
 }
 	
}
	G5 Compiler -- > Felipe & Giovanni & Vitor
	Reading from file programa.txt . . .
	0 Lexical Errors found
	0 Syntactic Errors found

package default;
 	
import:
  "abc.cc";
  "blabla.cc";
 	
public class MyClass
{
 static:
  int a = 1665;
  float b = -9865.501;
 	
 dynamic:
  string s;
 	
 public void setString(string ba)
 {
  s = ba;
  this.doSomething(31.5);
 }
	G5 Compiler -- > Felipe & Giovanni & Vitor
	Reading from file programa.txt . . .
	0 Lexical Errors found
	1 Syntactic Errors found
	Encountered " ";" "; "" at line 24, column 25.
	Was expecting:
	  "{" ...

```
 public void doSomething(double m)
 {
   if(a == m || m != b) ;
   {
     a = m ** b + (a // (3)) /100.0;
     for(int i = 0; i < b; i = i + (15 / a))
     {
       doSomething(i);
     }
   }
 }

}
```

```
package default;

import:
  "abc.cc;
  "blabla.cc";

public class MyClass
{
 static:
  int a = 1665;
  float b = -9865.501;

 dynamic:
  string s;

 public void setString(string ba)
 {
   s = ba;
   this.doSomething(31.5);
 }

 public void doSomething(double m)
 {
   if(a == m || m != b) ;
   {
     a = m ** b + (a // (3)) /100.0;
     for(int i = 0; i < b; i = i + (15 / a))
     {
       doSomething(i);
     }
   }
 }
}
```

```
G5 Compiler -- > Felipe & Giovanni &
Vitor
Reading from file programa.txt . . .
Line 4 - String constant has a error
"abc.cc;

Encountered " ";" "; "" at line 24,
column 25.
Was expecting:
    "{" ...

1 Lexical Errors found
1 Syntactic Errors found
```

Conclusão:

O analisador foi implementado corretamente, e os resultados da análise do programa criado ocorreram de acordo como previsto, analisando o código escrito de maneira correta e obtendo zero erros sintáticos e léxicos; e analisando os dois códigos com erros propositais e identificando os erros.