

Trabalho Compiladores

Analizador Sintático - pasC

Professor: Gustavo Fernandes

Integrantes:

Marcos Magno de Carvalho

Marcos Vinicius Gonçalves Cunha

Package compiladores

- **compiler.syntatic**

- **Classe**

- **RunSyntatic**

Essa classe iniciar o objeto Lexer e Parser.

Atributos:

lexer (object) : Um objeto da classe Lexer

parser (object) : Um objeto da classe Parser.

- **Métodos**

- main()

Esse método é responsável por iniciar o parser.

- **Classe**

- **Parser**

Essa classe implementa o Parser

Atributos:

lexer (object) : Um objeto da classe Lexer.

tag (object) : Um objeto da classe Tag_Type.

token (String): Uma String que representa um token.

- **Métodos**

- advance()

Esse método é responsável solicitar um novo token.

Atributos:

token (String): Uma String que representa um token.

- sinaliza_erro(mensagem)

Esse método é responsável por sinalizar erros encontrados.

Parâmetros:

mensagem (string): Um mensagem de erro

- eat(recv_token)

Esse método consumo um token.

Parâmetros:

recv_token (String): Uma String que representa um token.

- start_parse()

Esse método é responsável por consumir o primeiro token, que obrigatoriamente tem que ser um KW_PROGRAM.

- prog()
Esse método é responsável por produzir o não terminal prog:
prog -> "program" "id" body(1)

- body()
Esse método é responsável por produzir o não terminal body:
body -> decl-list "{" stmt-list "}" (2)

- decl_list()
Esse método é responsável por produzir o não terminal

decl_list:

decl_list -> decl ";" decl-list (3) | vazio(4)

- decl()
Esse método é responsável por produzir o não terminal decl:
decl -> type id-list (5)

- type()
Esse método é responsável por produzir o não terminal type:
type -> "num" (6) | "char" (7)

- id_list()
Esse método é responsável por produzir o não terminal id_list:
id-list -> "id" id_list' (8)

- id_list_linha()
Esse método é responsável por produzir o não terminal

id_list_linha:

id-list' -> "," id_list (9)

- stmt_list()
Esse método é responsável por produzir o não terminal

stmt_list:

stmt-list -> stmt ";" stmt-list (11) | vazio (12)

- stmt()
Esse método é responsável por produzir o não terminal stmt:
stmt -> assing-stmt (13) | if-stmt (14) | while-stmt (15) |

read-stmt (16) | write-stmt (17)

- assing_stmt()
Esse método é responsável por produzir o não terminal

assing_stmt:

assing-stmt -> "id" "=" simple_exprt (18)

- `if_stmt()`
Esse método é responsável por produzir o não terminal

`if_stmt:`

`if_stmt -> "if" "(" condition ")" "{" stmt-list "}" if_stmt' (19)`

- `if_stmt_linha()`
Esse método é responsável por produzir o não terminal

`if_stmt_linha:`

`if_stmt_linha -> "else" "{" stmt-list "}" (20) | vazio (21)`

- `while_stmt()`
Esse método é responsável por produzir o não terminal

`while_stmt:`

`while_stmt -> stmt-prefix "{" stmt-list "}" (23)`

- `stmt_prefix()`
Esse método é responsável por produzir o não terminal

`stmt_prefix:`

`"while" "(" condition ")" (24)`

- `read_stmt()`
Esse método é responsável por produzir o não terminal

`read_stmt:`

`"read" "id" (25)`

- `write_stmt()`
Esse método é responsável por produzir o não terminal

`write_stmt:`

`write" writable`

- `writable()`
Esse método é responsável por produzir o não terminal

`writable:`

`writable-> simple-expr (27) | "literal" (28)`

- `expression()`
Esse método é responsável por produzir o não terminal

`expression:`

`expression-> simple-expr expression_linha (29)`

- `expression_linha()`
Esse método é responsável por produzir o não terminal

`expression_linha:`

`expression_linha-> relop expression (30) | vazio (31)`

- `simple_expr_linha()`

Esse método é responsável por produzir o não terminal

`simple_expr_linha`:

`simple_expr_linha -> term simple_expr (33) | vazio (34)`

- `term()`

Esse método é responsável por produzir o não terminal `term`:

`term -> factor_a term_linha (35)`

- `term_linha()`

Esse método é responsável por produzir o não terminal

`term_linha`:

`term_linha -> mulop factor_a term_linha (36) | vazio (37)`

- `factor_a()`

Esse método é responsável por produzir o não terminal

`factor_a`:

`factor_a -> factor (38) | not factor (39)`

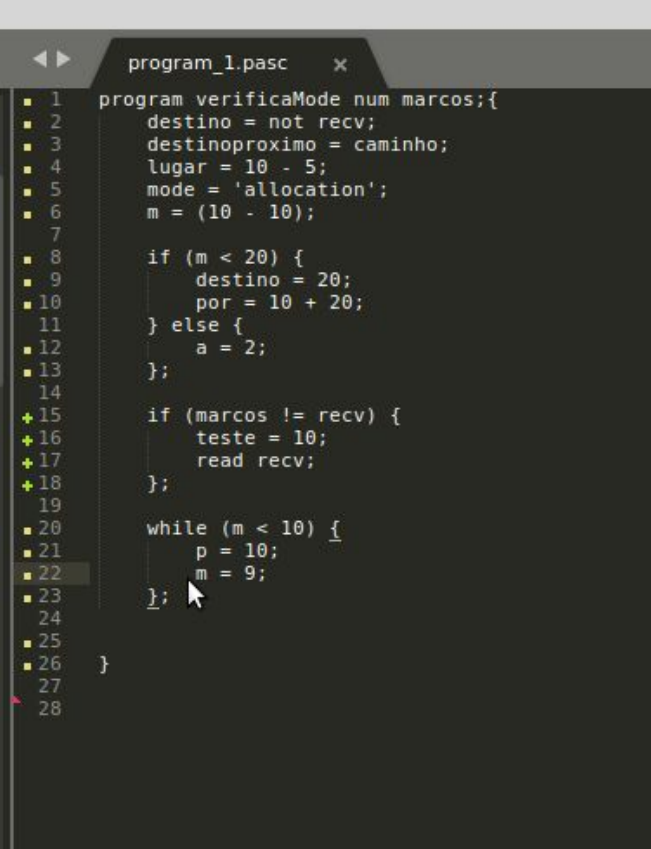
- `factor()`

Esse método é responsável por produzir o não terminal `factor`:

`factor -> "id" (40) | constant (41) | "(" expression ")" (42)`

Report

Programa Correto 1:



```
program verificaMode num marcos;{
  destino = not recv;
  destinoproximo = caminho;
  lugar = 10 - 5;
  mode = 'allocation';
  m = (10 - 10);

  if (m < 20) {
    destino = 20;
    por = 10 + 20;
  } else {
    a = 2;
  };

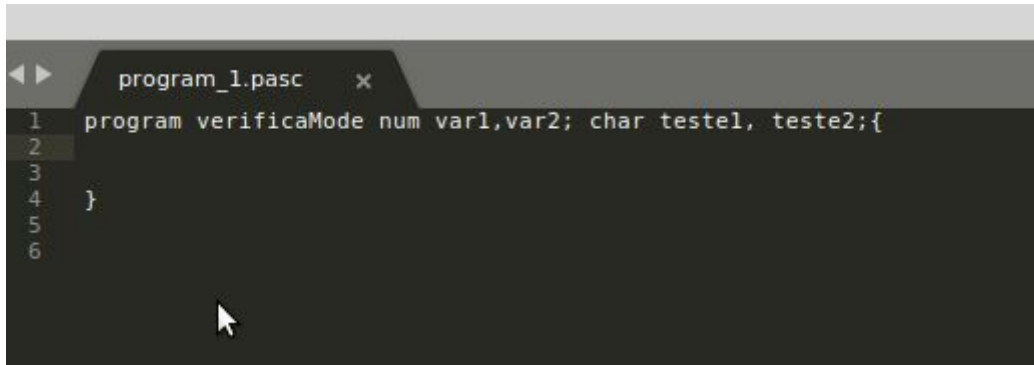
  if (marcos != recv) {
    teste = 10;
    read recv;
  };

  while (m < 10) {
    p = 10;
    m = 9;
  };
}
```

Saída:

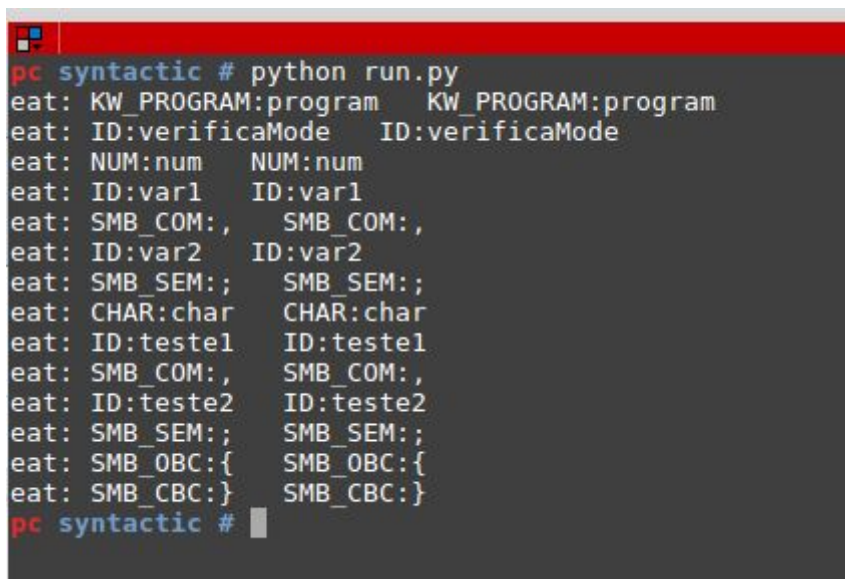
```
pc syntactic # python run.py
eat: KW PROGRAM:program KW PROGRAM:program
eat: ID:verificaMode ID:verificaMode
eat: NUM:num NUM:num
eat: ID:marcos ID:marcos
eat: SMB_SEM;; SMB_SEM;;
eat: SMB_OBC:{ SMB_OBC:{
eat: ID:destino ID:destino
eat: OP_ASS:= OP_ASS:=
eat: KW NOT:not KW NOT:not
eat: ID:recv ID:recv
eat: SMB_SEM;; SMB_SEM;;
eat: ID:destinoproximo ID:destinoproximo
eat: OP_ASS:= OP_ASS:=
eat: ID:caminho ID:caminho
eat: SMB_SEM;; SMB_SEM;;
eat: ID:lugar ID:lugar
eat: OP_ASS:= OP_ASS:=
eat: CON_NUM:10 CON_NUM:10
eat: OP_MIN:- OP_MIN:-
eat: CON_NUM:5 CON_NUM:5
eat: SMB_SEM;; SMB_SEM;;
eat: ID:mode ID:mode
eat: OP_ASS:= OP_ASS:=
eat: SMB_SEM;; SMB_SEM;;
eat: ID:m ID:m
eat: OP_ASS:= OP_ASS:=
eat: SMB_OPA:( SMB_OPA:(
eat: CON_NUM:10 CON_NUM:10
eat: OP_MIN:- OP_MIN:-
eat: CON_NUM:10 CON_NUM:10
eat: SMB_CPA:) SMB_CPA:)
eat: SMB_SEM;; SMB_SEM;;
eat: KW IF:if KW IF:if
eat: SMB_OPA:( SMB_OPA:(
eat: ID:m ID:m
eat: OP_LT:< OP_LT:<
eat: CON_NUM:20 CON_NUM:20
eat: SMB_CPA:) SMB_CPA:)
eat: SMB_OBC:{ SMB_OBC:{
eat: ID:destino ID:destino
eat: OP_ASS:= OP_ASS:=
eat: CON_NUM:20 CON_NUM:20
eat: SMB_SEM;; SMB_SEM;;
eat: ID:por ID:por
eat: OP_ASS:= OP_ASS:=
eat: CON_NUM:10 CON_NUM:10
eat: OP_AD:+ OP_AD:+
eat: CON_NUM:20 CON_NUM:20
eat: SMB_SEM;; SMB_SEM;;
eat: SMB_CBC;} SMB_CBC;}
eat: KW ELSE:else KW ELSE:else
eat: SMB_OBC:{ SMB_OBC:{
eat: ID:a ID:a
eat: OP_ASS:= OP_ASS:=
eat: CON_NUM:2 CON_NUM:2
eat: SMB_SEM;; SMB_SEM;;
eat: SMB_CBC;} SMB_CBC;}
eat: SMB_SEM;; SMB_SEM;;
eat: KW IF:if KW IF:if
eat: SMB_OPA:( SMB_OPA:(
eat: ID:marcos ID:marcos
eat: OP_NE:!= OP_NE:!=
eat: ID:recv ID:recv
eat: SMB_CPA:) SMB_CPA:)
eat: SMB_OBC:{ SMB_OBC:{
eat: ID:teste ID:teste
eat: OP_ASS:= OP_ASS:=
eat: CON_NUM:10 CON_NUM:10
eat: SMB_SEM;; SMB_SEM;;
eat: KW_READ:read KW_READ:read
eat: ID:recv ID:recv
eat: SMB_SEM;; SMB_SEM;;
eat: SMB_CBC;} SMB_CBC;}
eat: SMB_SEM;; SMB_SEM;;
eat: KW WHILE:while KW WHILE:while
eat: SMB_OPA:( SMB_OPA:(
eat: ID:m ID:m
eat: OP_LT:< OP_LT:<
eat: CON_NUM:10 CON_NUM:10
eat: SMB_CPA:) SMB_CPA:)
eat: SMB_OBC:{ SMB_OBC:{
eat: ID:p ID:p
eat: OP_ASS:= OP_ASS:=
eat: CON_NUM:10 CON_NUM:10
eat: SMB_SEM;; SMB_SEM;;
eat: ID:m ID:m
eat: OP_ASS:= OP_ASS:=
eat: CON_NUM:9 CON_NUM:9
eat: SMB_SEM;; SMB_SEM;;
eat: SMB_CBC;} SMB_CBC;}
eat: SMB_SEM;; SMB_SEM;;
eat: SMB_CBC;} SMB_CBC;}
pc syntactic #
```

Programa Correto 2:



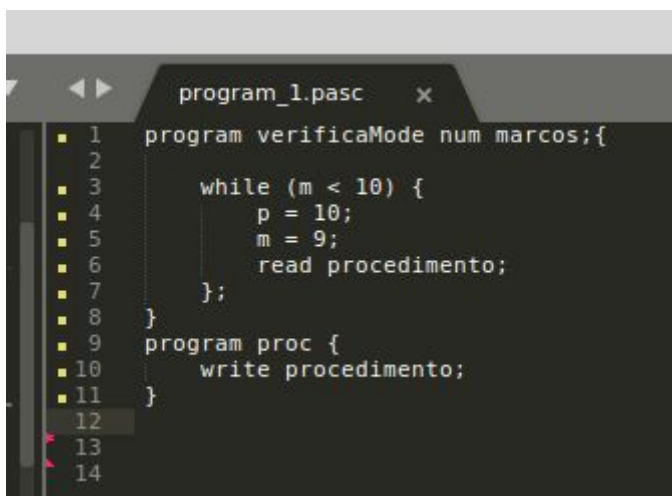
```
program_1.pasc x
1  program verificaMode num var1,var2; char teste1, teste2;{
2
3
4  }
5
6
```

Saída:



```
pc syntactic # python run.py
eat: KW_PROGRAM:program KW_PROGRAM:program
eat: ID:verificaMode ID:verificaMode
eat: NUM:num NUM:num
eat: ID:var1 ID:var1
eat: SMB_COM:, SMB_COM:,
eat: ID:var2 ID:var2
eat: SMB_SEM;; SMB_SEM;;
eat: CHAR:char CHAR:char
eat: ID:teste1 ID:teste1
eat: SMB_COM:, SMB_COM:,
eat: ID:teste2 ID:teste2
eat: SMB_SEM;; SMB_SEM;;
eat: SMB_OBC:{ SMB_OBC:{
eat: SMB_CBC:} SMB_CBC:}
pc syntactic #
```

Programa Correto 3:



```
program_1.pasc x
1  program verificaMode num marcos;{
2
3      while (m < 10) {
4          p = 10;
5          m = 9;
6          read procedimento;
7      };
8  }
9  program proc {
10     write procedimento;
11 }
12
13
14
```

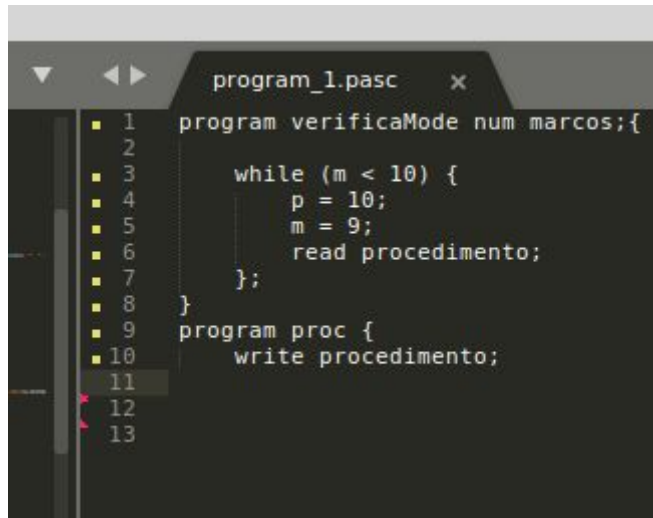

Saída:

```
pc syntactic # python run.py
eat: KW_PROGRAM:program KW_PROGRAM:program
eat: ID:verificaMode ID:verificaMode
eat: NUM:num NUM:num
eat: ID:marcos ID:marcos
eat: SMB_SEM;; SMB_SEM;;
eat: SMB_OBC:{ SMB_OBC:{
eat: KW_WHILE:while KW_WHILE:while
eat: SMB_OPA:( SMB_OPA:(
eat: ID:m ID:m
eat: OP_LT:< OP_LT:<
eat: CON_NUM:10 CON_NUM:10
eat: SMB_CPA:) SMB_CPA:)
eat: SMB_OBC:{ SMB_OBC:{
eat: ID:p ID:p
eat: OP_ASS:= OP_ASS:=
eat: CON_NUM:10 CON_NUM:10
eat: SMB_SEM;; SMB_SEM;;
eat: ID:m ID:m
eat: OP_ASS:= OP_ASS:=
eat: CON_NUM:9 CON_NUM:9
eat: SMB_SEM;; SMB_SEM;;
eat: KW_READ:read KW_READ:read
eat: ID:procedimento ID:procedimento
eat: SMB_SEM;; SMB_SEM;;
eat: SMB_CBC:} SMB_CBC:}
eat: SMB_SEM;; SMB_SEM;;
eat: SMB_CBC:} SMB_CBC:}
eat: KW_PROGRAM:program KW_PROGRAM:program
eat: ID:proc ID:proc
eat: SMB_OBC:{ SMB_OBC:{
eat: KW_WRITE:write KW_WRITE:write
eat: ID:procedimento ID:procedimento
eat: SMB_SEM;; SMB_SEM;;
eat: SMB_CBC:} SMB_CBC:}
pc syntactic #
```

Programa com Erro 1

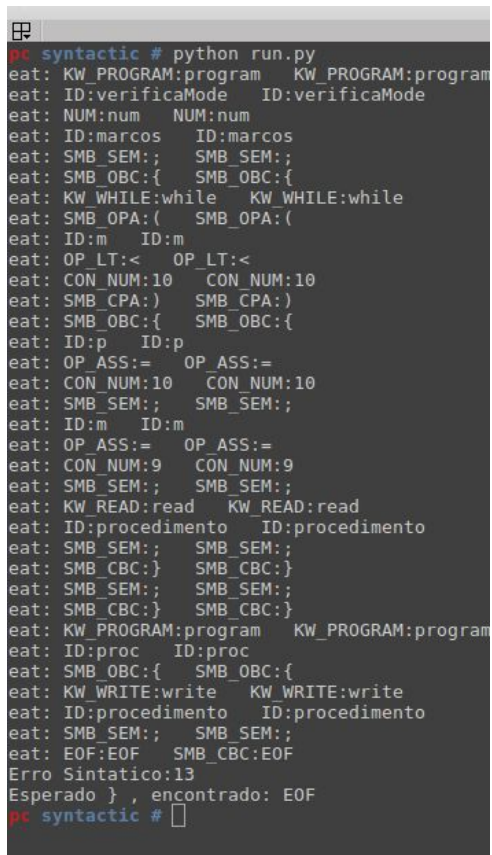
Este programa teve como objetivo testar erros sintáticos:

O primeiro erro é o não fechamento do colchete no final do arquivo.



```
program_1.pasc x
1  program verificaMode num marcos;{
2
3      while (m < 10) {
4          p = 10;
5          m = 9;
6          read procedimento;
7      };
8  }
9  program proc {
10     write procedimento;
11
12
13
```

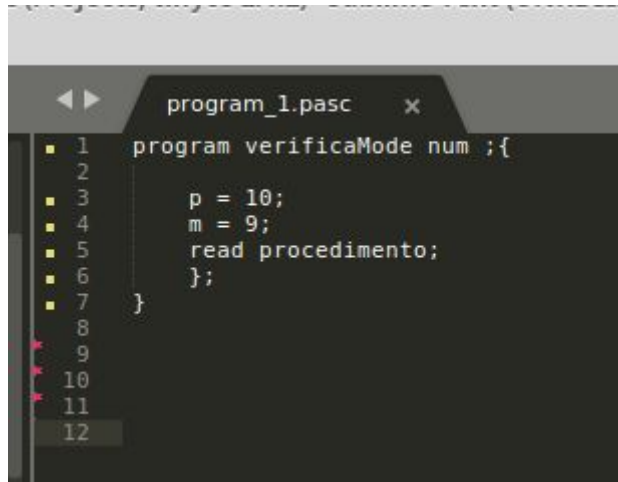
Saída:



```
pc syntactic # python run.py
eat: KW_PROGRAM:program KW_PROGRAM:program
eat: ID:verificaMode ID:verificaMode
eat: NUM:num NUM:num
eat: ID:marcos ID:marcos
eat: SMB_SEM;; SMB_SEM;;
eat: SMB_OBC:{ SMB_OBC:{
eat: KW_WHILE:while KW_WHILE:while
eat: SMB_OPA:( SMB_OPA:(
eat: ID:m ID:m
eat: OP_LT:< OP_LT:<
eat: CON_NUM:10 CON_NUM:10
eat: SMB_CPA:) SMB_CPA:)
eat: SMB_OBC:{ SMB_OBC:{
eat: ID:p ID:p
eat: OP_ASS:= OP_ASS:=
eat: CON_NUM:10 CON_NUM:10
eat: SMB_SEM;; SMB_SEM;;
eat: ID:m ID:m
eat: OP_ASS:= OP_ASS:=
eat: CON_NUM:9 CON_NUM:9
eat: SMB_SEM;; SMB_SEM;;
eat: KW_READ:read KW_READ:read
eat: ID:procedimento ID:procedimento
eat: SMB_SEM;; SMB_SEM;;
eat: SMB_CBC:} SMB_CBC:}
eat: SMB_SEM;; SMB_SEM;;
eat: SMB_CBC:} SMB_CBC:}
eat: KW_PROGRAM:program KW_PROGRAM:program
eat: ID:proc ID:proc
eat: SMB_OBC:{ SMB_OBC:{
eat: KW_WRITE:write KW_WRITE:write
eat: ID:procedimento ID:procedimento
eat: SMB_SEM;; SMB_SEM;;
eat: EOF:EOF SMB_CBC:EOF
Erro Sintatico:13
Esperado } , encontrado: EOF
pc syntactic #
```

Programa com Erro 2

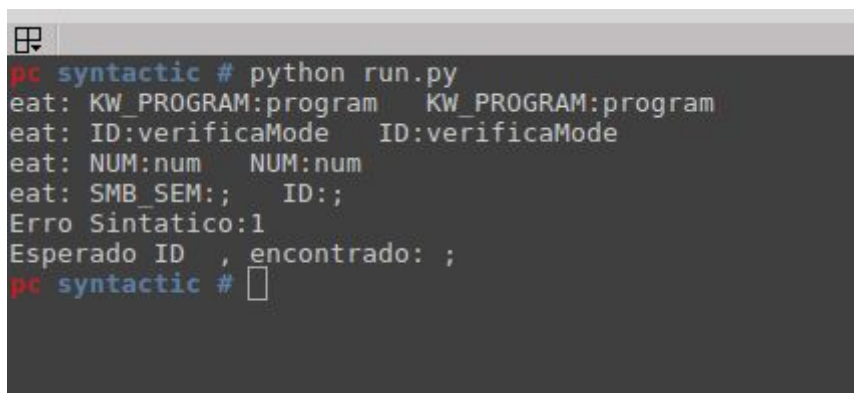
Esse programa simula a não declaração de um ID depois do type.



```
program verificaMode num ;{  
  p = 10;  
  m = 9;  
  read procedimento;  
};  
}
```

The screenshot shows a code editor window titled 'program_1.pasc'. The code is a Pascal program named 'verificaMode' with a parameter 'num'. It contains three lines of code: 'p = 10;', 'm = 9;', and 'read procedimento;'. The program ends with a closing brace '}'. The line numbers 1 through 12 are visible on the left side of the editor.

Saída:

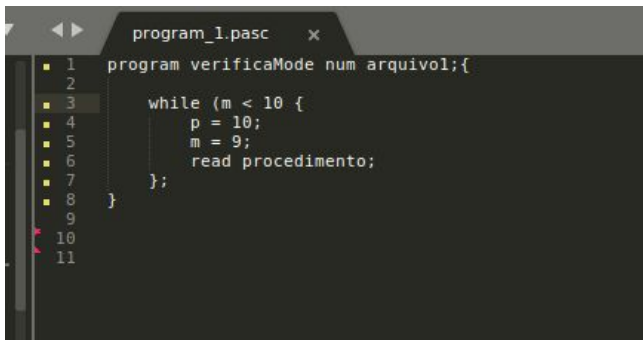


```
pc syntactic # python run.py  
eat: KW_PROGRAM:program KW_PROGRAM:program  
eat: ID:verificaMode ID:verificaMode  
eat: NUM:num NUM:num  
eat: SMB_SEM;; ID;;  
Erro Sintatico:1  
Esperado ID , encontrado: ;  
pc syntactic #
```

The screenshot shows a terminal window with the output of a Python script. The output displays the tokens 'KW_PROGRAM:program', 'ID:verificaMode', and 'NUM:num' being recognized. It then reports a syntax error: 'Erro Sintatico:1' and 'Esperado ID , encontrado: ;', indicating that the semicolon was not expected after the parameter list.

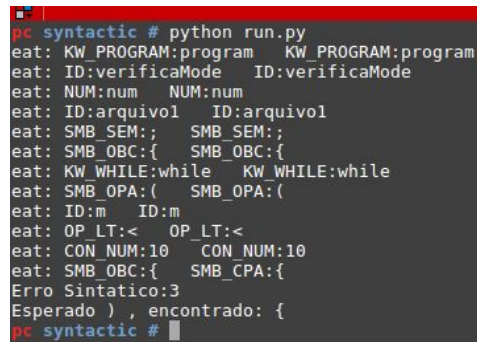
Programa com Erro 3

Este programa simula o erro do não fechamento do parêntese na declaração de uma expressão.



```
program_1.pasc x
1 program verificaMode num arquivo1;{
2
3     while (m < 10 {
4         p = 10;
5         m = 9;
6         read procedimento;
7     };
8 }
9
10
11
```

Saída:



```
pc syntactic # python run.py
eat: KW_PROGRAM:program KW_PROGRAM:program
eat: ID:verificaMode ID:verificaMode
eat: NUM:num NUM:num
eat: ID:arquivo1 ID:arquivo1
eat: SMB_SEM;; SMB_SEM;;
eat: SMB_OBC:{ SMB_OBC:{
eat: KW_WHILE:while KW_WHILE:while
eat: SMB_OPA:( SMB_OPA:(
eat: ID:m ID:m
eat: OP_LT:< OP_LT:<
eat: CON_NUM:10 CON_NUM:10
eat: SMB_OBC:{ SMB_OPA:{
Erro Sintatico:3
Esperado ) , encontrado: {
pc syntactic #
```