Fortran 666 - Analisador Léxico

Makhles Reuter Lange
Maurilio Atila Carvalho de Santana

August 22, 2017

1 Fortran 666

O Fortran é uma linguagem de programação imperativa desenvolvida especialmente para computação numérica e científica em meados de 1950 pela IBM. Desde a sua criação, diversas versões foram desenvolvidas, dentre as quais cita-se o FORTRAN 66, o FORTRAN 77 e o Fortran 90 como as principais. A linguagem que será utilizada para a construção do compilador ao longo da disciplina utilizará características dessas diversas versões.

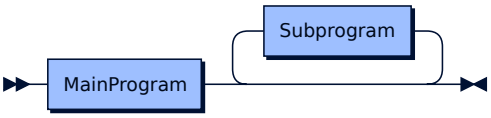
2 Gramática

A gramática do Fortran 666 foi criada pelos integrantes da equipe a partir do zero. Posteriormente, consultou-se a gramática oficial do FORTRAN 77 e decidiu-se adotar alguns dos nomes dos seus símbolos não-terminais.

A descrição formal da gramática livre de contexto foi feita utilizando-se o *Formalismo de Backus-Naur Estendido* (EBNF)[1]. A partir da gramática na forma EBNF, utilizou-se o site RailRoad[2] para gerar diagramas de sintaxe. O resultado pode ser visto na página seguinte.

[1] O EBNF é uma família de notações meta-sintaxe para a descrição formal de linguagens formais.
[2] Vide http://www.bottlecaps.de/rr/ui

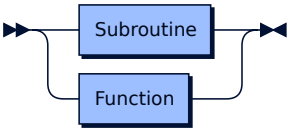
ExecutableProgram:




```
ExecutableProgram
        ::= MainProgram Subprogram*
```

no references

Subprogram:




```
Subprogram
        ::= Subroutine
          | Function
```

referenced by:

- ExecutableProgram

MainProgram:


```
MainProgram
        ::= MainProgramPrefix Body MainProgramSuffix
```

referenced by:

- ExecutableProgram

Subroutine:


```
Subroutine
        ::= SubroutinePrefix '(' ParameterList ')' Body SubroutineSuffix
```

referenced by:

- Subprogram

Function:


```
Function ::= FunctionPrefix '(' ParameterList ')' Body FunctionSuffix
```

referenced by:

- Subprogram

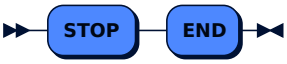
MainProgramPrefix:


```
MainProgramPrefix
         ::= 'PROGRAM' Name
```

referenced by:

- MainProgram

MainProgramSuffix:




```
MainProgramSuffix
         ::= 'STOP' 'END'
```

referenced by:

- MainProgram

SubroutinePrefix:


```
SubroutinePrefix
         ::= 'SUBROUTINE' Name
```

referenced by:

- Subroutine

SubroutineSuffix:


```
SubroutineSuffix
         ::= 'RETURN' 'END'
```

referenced by:

- Subroutine

FunctionPrefix:


```
FunctionPrefix
         ::= Type 'FUNCTION' Name
```

referenced by:

- Function

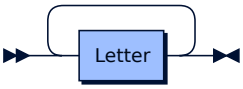
FunctionSuffix:


```
FunctionSuffix
        ::= 'RETURN' 'END'
```

referenced by:

- Function

Name:

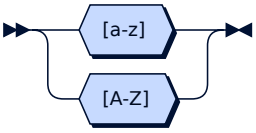



```
Name    ::= Letter+
```

referenced by:

- CallStatement
- FunctionPrefix
- MainProgramPrefix
- SubroutinePrefix

Letter:




```
Letter  ::= [a-zA-Z]
```

referenced by:

- Alphanumeric
- Identifier
- Name

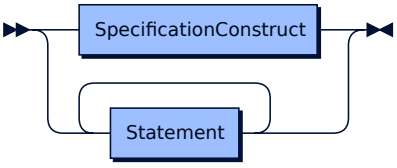
Body:


```
Body    ::= BodyConstruct+
```

referenced by:

- Function
- MainProgram
- Subroutine

BodyConstruct:

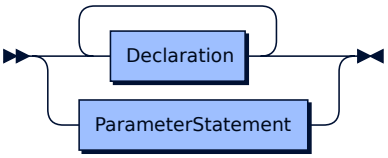


```
BodyConstruct
          ::= SpecificationConstruct
            | Statement+
```

referenced by:

- Body

SpecificationConstruct:

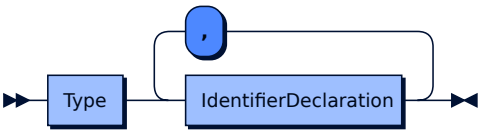



```
SpecificationConstruct
          ::= Declaration+
            | ParameterStatement
```

referenced by:

- BodyConstruct

Declaration:

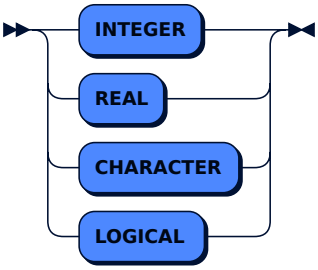



```
Declaration
          ::= Type IdentifierDeclaration ( ',' IdentifierDeclaration )*
```

referenced by:

- SpecificationConstruct

Type:

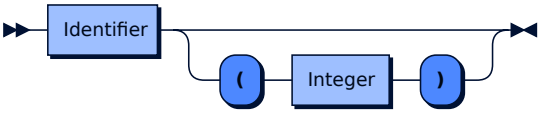



```
Type      ::= 'INTEGER'
            | 'REAL'
            | 'CHARACTER'
            | 'LOGICAL'
```

referenced by:

- Declaration
- FunctionPrefix

IdentifierDeclaration:

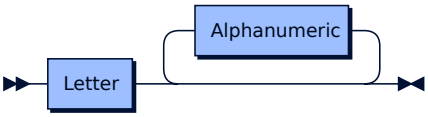



```
IdentifierDeclaration
         ::= Identifier ( '(' Integer ')' )?
```

referenced by:

- Declaration

Identifier:

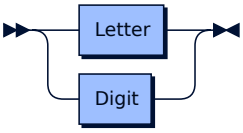



```
Identifier
         ::= Letter Alphanumeric*
```

referenced by:

- AssignmentStatement
- ConstantDefinition
- DoLoopControl
- IdentifierDeclaration
- Term

Alphanumeric:

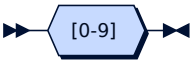



```
Alphanumeric
         ::= Letter
           | Digit
```

referenced by:

- Identifier

Digit:

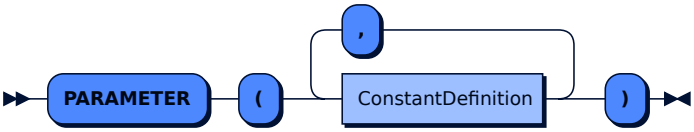



```
Digit    ::= [0-9]
```

referenced by:

- Alphanumeric
- Integer

ParameterStatement:

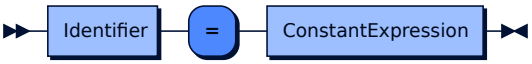


```
ParameterStatement
         ::= 'PARAMETER' '(' ConstantDefinition ( ',' ConstantDefinition )* ')'
```

referenced by:

- SpecificationConstruct

ConstantDefinition:

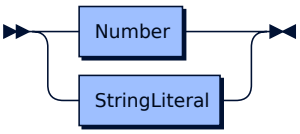



```
ConstantDefinition
         ::= Identifier '=' ConstantExpression
```

referenced by:

- ParameterStatement

ConstantExpression:

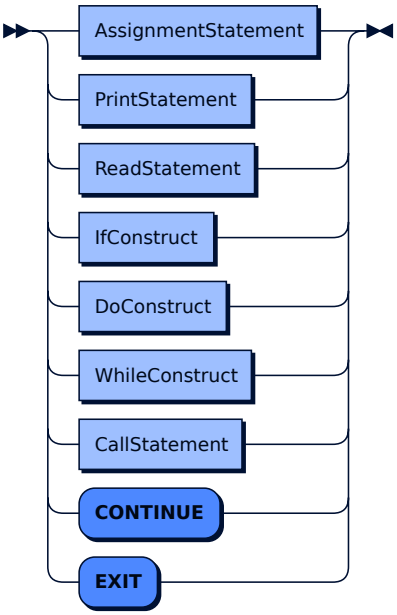



```
ConstantExpression
         ::= Number
           | StringLiteral
```

referenced by:

- ConstantDefinition

Statement:



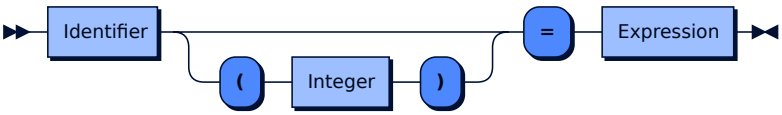

```
Statement
         ::= AssignmentStatement
           | PrintStatement
           | ReadStatement
           | IfConstruct
```

```
          | DoConstruct
          | WhileConstruct
          | CallStatement
          | 'CONTINUE'
          | 'EXIT'
```

referenced by:

- BodyConstruct
- EndDoStatement
- EndWhileStatement
- ThenConstruct

AssignmentStatement:

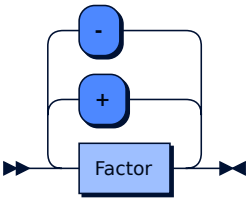



```
AssignmentStatement
         ::= Identifier ( '(' Integer ')' )? '=' Expression
```

referenced by:

- Statement

Expression:

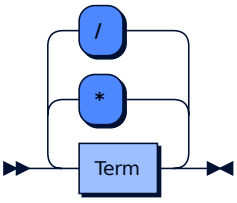



```
Expression
         ::= Factor ( ( '+' | '-' ) Factor )*
```

referenced by:

- AssignmentStatement
- DoLoopControl
- ElseConstruct
- ElseIfStatement
- LogicalExpression
- PrintItem
- Term

Factor:

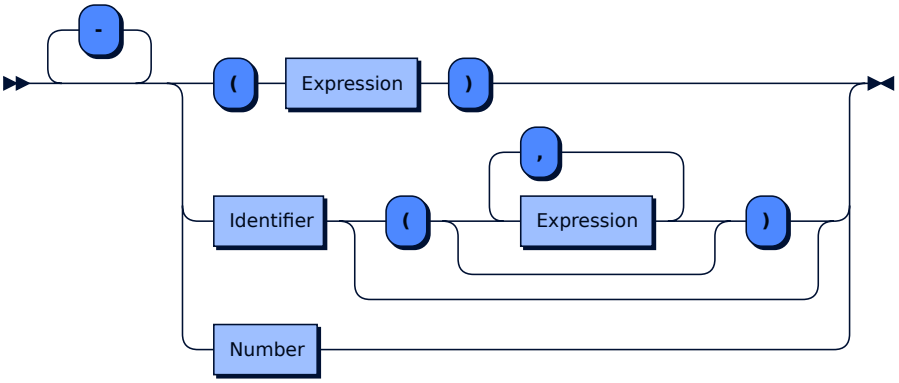



```
Factor   ::= Term ( ( '*' | '/' ) Term )*
```

referenced by:

- Expression

Term:

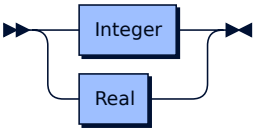



```
Term     ::= '-'* ( '(' Expression ')' | Identifier ( '(' ( Expression ( ',' Expression )* )? ')' )? | Number )
```

referenced by:

- Factor

Number:

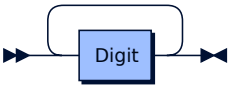



```
Number   ::= Integer
           | Real
```

referenced by:

- ConstantExpression
- Term

Integer:

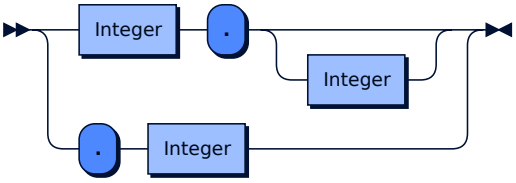



```
Integer  ::= Digit+
```

referenced by:

- AssignmentStatement
- IdentifierDeclaration
- Number
- Real

Real:

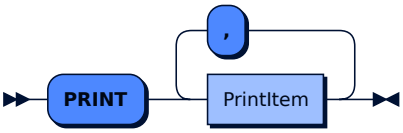



```
Real     ::= Integer '.' Integer?
           | '.' Integer
```

referenced by:

- Number

PrintStatement:

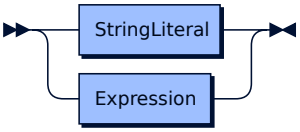



```
PrintStatement
         ::= 'PRINT' PrintItem ( ',' PrintItem )*
```

referenced by:

- Statement

PrintItem:




```
PrintItem
         ::= StringLiteral
           | Expression
```

referenced by:

- PrintStatement

StringLiteral:


```
StringLiteral
         ::= "'' Text " ''
```

referenced by:

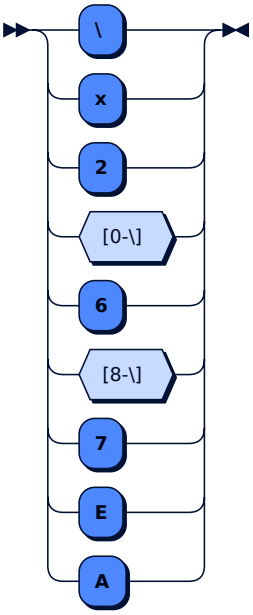
- ConstantExpression
- PrintItem

Text:


```
Text     ::= TextChar+
```

no references

TextChar:

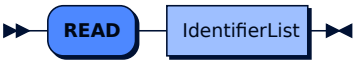



```
TextChar ::= [\x20-\68-\7EA]
```

referenced by:

- Text

ReadStatement:

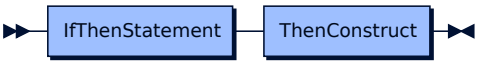



```
ReadStatement
         ::= 'READ' IdentifierList
```

referenced by:

- Statement

IfConstruct:




```
IfConstruct
         ::= IfThenStatement ThenConstruct
```

referenced by:

- Statement

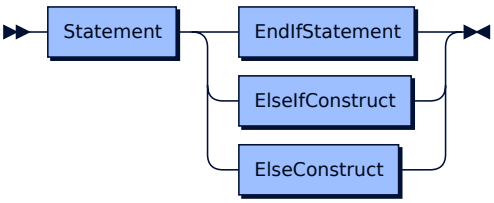
IfThenStatement:


```
IfThenStatement
         ::= 'IF' LogicalExpression 'THEN'
```

referenced by:

- IfConstruct

ThenConstruct:

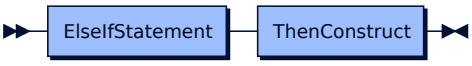



```
ThenConstruct
        ::= Statement ( EndIfStatement | ElseIfConstruct | ElseConstruct )
```

referenced by:

- ElseIfConstruct
- IfConstruct

ElseIfConstruct:




```
ElseIfConstruct
        ::= ElseIfStatement ThenConstruct
```

referenced by:

- ThenConstruct

ElseIfStatement:


```
ElseIfStatement
        ::= 'ELSEIF' Expression 'THEN'
```

referenced by:

- ElseIfConstruct

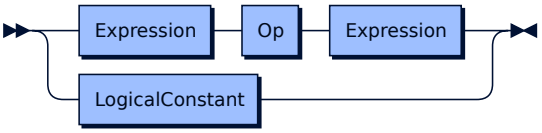
ElseConstruct:


```
ElseConstruct
        ::= 'ELSE' Expression 'END'
```

referenced by:

- ThenConstruct

LogicalExpression:

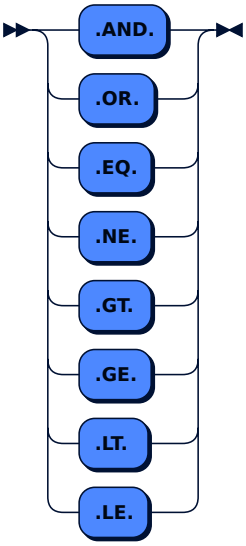


```
LogicalExpression
         ::= Expression Op Expression
           | LogicalConstant
```

referenced by:

- IfThenStatement
- WhileStatement

Op:

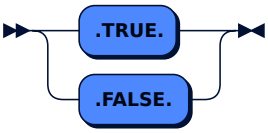



```
Op       ::= '.AND.'
           | '.OR.'
           | '.EQ.'
           | '.NE.'
           | '.GT.'
           | '.GE.'
           | '.LT.'
           | '.LE.'
```

referenced by:

- LogicalExpression

LogicalConstant:




```
LogicalConstant
         ::= '.TRUE.'
           | '.FALSE.'
```

referenced by:

- LogicalExpression

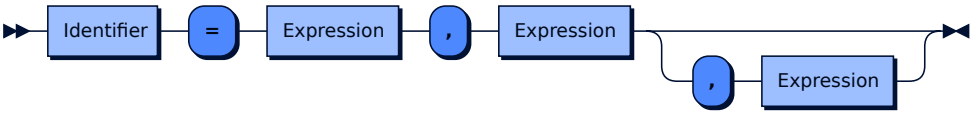
DoConstruct:

```
DoConstruct
         ::= 'DO' DoLoopControl EndDoStatement
```

referenced by:

- Statement

DoLoopControl:

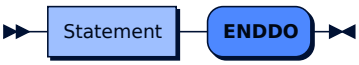



```
DoLoopControl
         ::= Identifier '=' Expression ',' Expression ( ',' Expression )?
```

referenced by:

- DoConstruct

EndDoStatement:

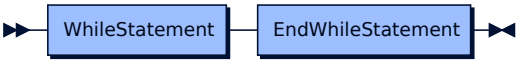



```
EndDoStatement
         ::= Statement 'ENDDO'
```

referenced by:

- DoConstruct

WhileConstruct:




```
WhileConstruct
         ::= WhileStatement EndWhileStatement
```

referenced by:

- Statement

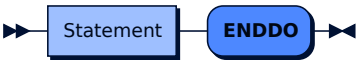
WhileStatement:


```
WhileStatement
         ::= 'WHILE' LogicalExpression 'DO'
```

referenced by:

- WhileConstruct

EndWhileStatement:

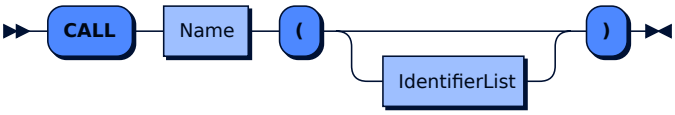


```
EndWhileStatement
        ::= Statement 'ENDDO'
```

referenced by:

- WhileConstruct

CallStatement:




```
CallStatement
        ::= 'CALL' Name '(' IdentifierList? ')'
```

referenced by:

- Statement

... generated by Railroad Diagram Generator

3 Flex

O Flex (*The Fast Lexical Analyzer*) é um gerador de analisador léxico desenvolvido por Vern Paxson em 1987 sob a licença de Berkeley. Na verdade, Paxson fez apenas uma tradução para a linguagem C de uma versão do programa *lex* escrita em *ratfor* (um Fortran estendido), originalmente desenvolvido por Mike Lesk e Eric Schmidt em 1975.

Um programa flex consiste em três seções separadas por linhas com os símbolos %%. A primeira seção contém declarações e configurações; a segunda contém uma lista de padrões e ações, enquanto que a terceira contém código C que é copiado para o analisador gerado.

O arquivo flex criado, *scanner.l*, pode ser visto abaixo.

```
%option noyywrap
%{
#include <stdio.h>
%}

letter      [A-Za-z]
digit       [0-9]
number      [-+]?({digit}*\.?{digit}+|{digit}+\.)([E]?[-+]?{digit}+)?
id          {letter}({letter}|{digit}|_)*
sliteral    \'([^\\\']|\\.)*\'
comment     "//".*\n
ws          [ \t]+
%%

"PROGRAM"       { printf(" RES "); }
"SUBROUTINE"    { printf(" RES "); }
"FUNCTION"      { printf(" RES "); }
"STOP"          { printf(" RES "); }
"RETURN"        { printf(" RES "); }
"END"           { printf(" RES "); }
"PARAMETER"     { printf(" RES "); }
"INTEGER"       { printf(" RES "); }
"REAL"          { printf(" RES "); }
"CYCLE"         { printf(" RES "); }
"EXIT"          { printf(" RES "); }
"IF"            { printf(" RES "); }
"ELSE"          { printf(" RES "); }
"ELSEIF"        { printf(" RES "); }
"ENDIF"         { printf(" RES "); }
"WHILE"         { printf(" RES "); }
"DO"            { printf(" RES "); }
"ENDDO"         { printf(" RES "); }
"PRINT"         { printf(" RES "); }
"READ"          { printf(" RES "); }
"CALL"          { printf(" RES "); }

"+"     { printf(" OP "); }
"-"     { printf(" OP "); }
```

```
"*"     { printf(" OP "); }
"/"     { printf(" OP "); }
"="     { printf(" OP "); }
".EQ."  { printf(" OP "); }
".NE."  { printf(" OP "); }
".GT."  { printf(" OP "); }
".GE."  { printf(" OP "); }
".LT."  { printf(" OP "); }
".LE."  { printf(" OP "); }

"," { printf(" PUNK "); }
"(" { printf(" PUNK "); }
")" { printf(" PUNK "); }

".TRUE." { printf(" LOG_VALUE "); }
".FALSE." { printf(" LOG_VALUE "); }

{ws}        { /* ignore whitespace */ }
{comment}   { printf(" COMMENT EOL\n"); }
{sliteral}  { printf(" STRING_LITERAL "); }
{id}        { printf(" ID "); }
{number}    { printf(" NUM "); }
\n          { printf(" EOL\n"); }
.           { printf(" ERR "); }

%%

int main (int argc, char *argv[]) {
    if (argc > 1) {
        if (!(yyin = fopen(argv[1], "r"))) {
            perror(argv[1]);
            return (1);
        }
    } else {
        printf("Missing input file.\n");
        return (1);
    }
    yylex();
    printf("\n");
    return (0);
}
```

4 Geração do analisador léxico

Ao rodar o programa flex passando-se o arquivo *scanner.l*, cria-se a rotina de análise léxica em C: *lex.yy.c*. Esse program deve ser compilado fazendo-se a ligação com a biblioteca `lbl`:

```
$ flex scanner.l
```

```
$ gcc lex.yy.c -lbl -o scanner
```

O analisador léxico gerado (*scanner*) encontra-se junto aos arquivos compactados.

5 Programas-teste

Para testar a identificação dos *tokens*, criou-se quatro arquivos escritos em Fortran 666, listados a seguir.

../test/swap.f

```
PROGRAM CALLEX
INTEGER M, N

M = 1
N = 2

CALL ISWAP(M, N)
PRINT M, N

STOP
END


SUBROUTINE ISWAP (A, B)
INTEGER A, B

// Local variables
INTEGER TMP

TMP = A
A = B
B = TMP

RETURN
END
```

../test/rain.f

```
PROGRAM RAIN
REAL R, T, SUM
INTEGER M

READ T
SUM = 0.0
DO 10 M = 1, 12
   SUM = SUM + R(M, T)
ENDDO
PRINT 'Annual rainfall is ', SUM, ' inches.'

STOP
END


// Compute the amount of rain R in the month M

REAL FUNCTION R(M,T)
```

```
INTEGER M
REAL T

R = 0.1*T * (M*(M+14) + 46)
IF (R .LT. 0) THEN
   R = 0.0
ENDIF

RETURN
END
```

../test/matmul.f

```
PROGRAM MATRIX_MULTIPLICATION

INTEGER I, M, N
LOGICAL ORDER_OK

PARAMETER (M = 10)
ORDER_OK = .FALSE.

INTEGER A(M,M), B(M,M), C(M,M)

WHILE (ORDER_OK .EQ. .FALSE.) DO
    PRINT 'Enter the order of the matrices (n <= ', M, ')'
    READ N
    IF (N .LE. 10) THEN
        ORDER_OK = .TRUE.
    ENDIF
ENDDO

CALL MATMUL(N,A,B,C)

PRINT 'Result:'
DO I = 1, N
    DO J = 1, N
        PRINT C(I, J)
    ENDDO
ENDDO

STOP
END


// Matrix multiplication

FUNCTION MATMUL(N, A, B, C)
INTEGER N
INTEGER A(N,N), B(N,N), C(N,N)

// Local variables

INTEGER I, J, K, TEMP

DO I = 1, N
    DO J = 1, N
        TEMP = 0
        DO K = 1, N
```

```
            TEMP = TEMP + A(I,K) * B(K,J)
        ENDDO
        C(I,J) = TEMP
    ENDDO
ENDDO

RETURN
END
```

../test/error.f

```
PROGRAM A
INTEGER X, Y
INTEGER SUM%  // <- Invalid character

CALL INPUT(X,Y)
PRINT "X+Y=", SUM(X,Y)

STOP
END

// Data input

SUBROUTINE INPUT(A,B)
INTEGER A,B

READ A, B

RETURN
END


// Compute the sum

INTEGER FUNCTION SUM(A,B)
INTEGER A, B

SUM = A + B

RETURN
END
```

6 Identificação dos *tokens*

A identificação dos *tokens* foi feita passando-se cada um dos arquivos listados no item anterior para o analisador léxico gerado, *scanner*. A impressão dos *tokens* é feita no próprio console.

Para facilitar a sua identificação, ao invés de imprimir cada *token* em uma linha separada, os *tokens* foram impressos nas mesmas linhas em que se encontravam no respectivo arquivo fonte. Os resultados obtidos se encontram nos arquivos com extensão `.stream` e podem ser vistos a seguir.

```
$ ./scanner ./test/swap.f
```

../test/swap.stream

```
RES  ID  EOL
RES  ID  PUNK  ID  EOL
EOL
ID  OP  NUM  EOL
ID  OP  NUM  EOL
EOL
RES  ID  PUNK  ID  PUNK  ID  PUNK  EOL
RES  ID  PUNK  ID  EOL
EOL
RES  EOL
RES  EOL
EOL
EOL
RES  ID  PUNK  ID  PUNK  ID  PUNK  EOL
RES  ID  PUNK  ID  EOL
EOL
COMMENT EOL
RES  ID  EOL
EOL
ID  OP  ID  EOL
ID  OP  ID  EOL
ID  OP  ID  EOL
EOL
RES  EOL
RES
```

```
$ ./scanner ./test/rain.f
```

../test/rain.stream

```
RES  ID  EOL
RES  ID  PUNK  ID  PUNK  ID  EOL
RES  ID  EOL
EOL
RES  ID  EOL
ID  OP  NUM  EOL
RES  NUM  ID  OP  NUM  PUNK  NUM  EOL
ID  OP  ID  OP  ID  PUNK  ID  PUNK  ID  PUNK  EOL
RES  EOL
RES  STRING_LITERAL  PUNK  ID  PUNK  STRING_LITERAL  EOL
EOL
RES  EOL
RES  EOL
EOL
EOL
COMMENT EOL
EOL
RES  RES  ID  PUNK  ID  PUNK  ID  PUNK  EOL
RES  ID  EOL
RES  ID  EOL
EOL
ID  OP  NUM  OP  ID  OP  PUNK  ID  OP  PUNK  ID  NUM  PUNK  OP  NUM  PUNK
     EOL
RES  PUNK  ID  OP  NUM  PUNK  ID  EOL
ID  OP  NUM  EOL
RES  EOL
```

```
EOL
RES  EOL
RES
```

```
$ ./scanner ./test/matmul.f
```

../test/matmul.stream

```
RES  ID  EOL
EOL
RES  ID  PUNK  ID  PUNK  ID  EOL
ID  ID  EOL
EOL
RES  PUNK  ID  OP  NUM  PUNK  EOL
ID  OP  LOG_VALUE  EOL
EOL
RES  ID  PUNK  ID  PUNK  ID  PUNK  PUNK  ID  PUNK  ID  PUNK  ID  PUNK
   PUNK  ID  PUNK  ID  PUNK  ID  PUNK  EOL
EOL
RES  PUNK  ID  OP  LOG_VALUE  PUNK  RES  EOL
RES  STRING_LITERAL  PUNK  ID  PUNK  STRING_LITERAL  EOL
RES  ID  EOL
RES  PUNK  ID  OP  NUM  PUNK  ID  EOL
ID  OP  LOG_VALUE  EOL
RES  EOL
RES  EOL
EOL
RES  ID  PUNK  ID  PUNK  ID  PUNK  ID  PUNK  ID  PUNK  EOL
EOL
RES  STRING_LITERAL  EOL
RES  ID  OP  NUM  PUNK  ID  EOL
RES  ID  OP  NUM  PUNK  ID  EOL
RES  ID  PUNK  ID  PUNK  ID  PUNK  EOL
RES  EOL
RES  EOL
EOL
RES  EOL
RES  EOL
EOL
EOL
COMMENT EOL
EOL
RES  ID  PUNK  ID  PUNK  ID  PUNK  ID  PUNK  ID  PUNK  EOL
RES  ID  EOL
RES  ID  PUNK  ID  PUNK  ID  PUNK  PUNK  ID  PUNK  ID  PUNK  ID  PUNK
   PUNK  ID  PUNK  ID  PUNK  ID  PUNK  EOL
EOL
COMMENT EOL
EOL
RES  ID  PUNK  ID  PUNK  ID  PUNK  ID  EOL
EOL
RES  ID  OP  NUM  PUNK  ID  EOL
RES  ID  OP  NUM  PUNK  ID  EOL
ID  OP  NUM  EOL
RES  ID  OP  NUM  PUNK  ID  EOL
ID  OP  ID  OP  ID  PUNK  ID  PUNK  ID  PUNK  OP  ID  PUNK  ID  PUNK  ID
    PUNK  EOL
```

```
RES  EOL
ID  PUNK  ID  PUNK  ID  PUNK  OP  ID  EOL
RES  EOL
RES  EOL
EOL
RES  EOL
RES
```

```
$ ./scanner ./test/error.f
```

../test/error.stream

```
RES  ID  EOL
RES  ID  PUNK  ID  EOL
RES  ID  ERR  COMMENT EOL
EOL
RES  ID  PUNK  ID  PUNK  ID  PUNK  EOL
RES  ERR  ID  OP  ID  OP  ERR  PUNK  ID  PUNK  ID  PUNK  ID  PUNK  EOL
EOL
RES  EOL
RES  EOL
EOL
COMMENT EOL
EOL
RES  ID  PUNK  ID  PUNK  ID  PUNK  EOL
RES  ID  PUNK  ID  EOL
EOL
RES  ID  PUNK  ID  EOL
EOL
RES  EOL
RES  EOL
EOL
EOL
COMMENT EOL
EOL
RES  RES  ID  PUNK  ID  PUNK  ID  PUNK  EOL
RES  ID  PUNK  ID  EOL
EOL
ID  OP  ID  OP  ID  EOL
EOL
RES  EOL
RES
```