# Descrição da linguagem para a turma COMPILADORES.2014.2

Esta linguagem não é case sensitive tanto para palavras reservadas quanto para variáveis. Logo, “for”, “FOR” e “For” representam a mesma palavra reservada, bem como “abcd”, “ABCD” e “aBCd” representam a mesma variável.

Além disso, o compilador só deve aceitar arquivos contendo caracteres com valores na tabela ASCII entre 9 e 10 (tab e quebra de linha) ou entre 32 e 126 (caracteres imprimíveis da tabela ASCII). Portanto, caracteres fora desta escala não são considerados nesta especificação.

## 1. Palavras e símbolos reservados:

DECLARE AS NUMBER LETTER PUT IN IF THEN ELSE FOREACH DO FOR FROM TO RESIZE READ PRINT , . [ ] + - * / % ( ) < > <= >= = <>

### 1.1. Separadores

Separadores são caracteres que indicam o fim de uma possível sequência de caracteres que formam um único elemento. Além dos tradicionais espaço em branco, tab e quebra de linha, existem vários outros possíveis separadores.

São considerados separadores os seguintes caracteres que ocorrem na linguagem: , . [ ] + - * / % ( ) = < > ' “

Além disso, todo caractere que não faz parte de nenhuma palavra ou símbolo reservado também é considerado um separador.

OBS: O caracter '<' não é um separador quando seguido de '>' ou '='. O caracter '>' não é um separador quando seguido de '='. Nestes casos, ambos os caracteres ('<>', '<=' ou '>=') formam um único token que é um separador.

OBS2: Aspa simples e aspas duplas são um tipo especial de separadores. Elas apenas separam o que vem antes ou depois de uma constante do tipo caractere (ver item 2.3) ou do tipo string (ver item 2.4), mas não são separadas da constante em si.

## 2. Constantes e identificadores

### 2.1. Identificador:

1 letra [a-zA-Z] seguida por 0 ou mais letras ou números [0-9].

Ex:
a
abCD
x0
a0B1c7

### 2.2. Constante numérica:

1 ou mais números [0-9] até um limite de 10 digitos.

Ex:
0123
1234567890

**2.3. Constante do tipo caractere:**

1 caractere imprimível entre aspas simples. São exceções o tab, a quebra de linha, a contra-barra, a aspa simples e as aspas duplas, que são respectivamente representados por '\t', '\n', '\\', '\" e '\”'. Não podem existir caracteres com os códigos 9 ou 10 entre aspas simples (os mesmos são representados por \t e \n, respectivamente). Se um destes caracteres for encontrado (ou o final de arquivo for alcançado) antes da aspa de fechamento, deve-se considerar que este caracter é o fechamento.

Ex:
'a'
'A'
'-'
'\\'
'\n'
'\"

**2.4. Constante do tipo string:**

Sequência de até 256 caracteres imprimíveis entre aspas duplas. As mesmas exceções apresentadas no item 2.3 são válidas aqui, e contam como um único caractere. Novamente, não podem existir caracteres com os códigos 9 ou 10 entre aspas duplas. Se um destes caracteres for encontrado (ou o final de arquivo for alcançado) antes das aspas de fechamento, deve-se considerar que este caracter é o fechamento.

Ex:
“abc\n”
“A saida esperada eh:\t”
“\\o/”

# 3. Comandos

Um programa é composto por qualquer quantidade dos comandos abaixo, em qualquer ordem.

**3.1. Declarações:**

As declarações devem ser feitas da seguinte forma:

DECLARE $lista_ids$ AS $tipo$ .

$tipo$ pode receber os valores NUMBER (inteiro de 64 bits) e LETTER (inteiro de 8 bits). $lista_ids$ é uma lista com um ou mais identificadores separados por vírgula, sendo que cada identificador pode

ser seguido de abre e fecha colchetes para indicar um vetor.

Ex:
DECLARE X AS LETTER.
DECLARE y, w,k as number .
declare vetor [ ] , lista[]as Letter.DECLare oQUE[] as NuMbEr.

**3.2. Alocação de vetores**

As alocações devem ser feitas da seguinte forma:

RESIZE $id$ TO $exp_arit$ .

$id$ é o identificador do vetor a ser alocado, e $expr_arit$ é uma expressão aritmética que definirá o tamanho do vetor.

Ex:
RESIZE vetor TO a+b.
RESIZE nome TO MAXTAM.

Uma posição do vetor pode ser indexada pela seguinte forma:

$id$ [ $exp_arit$ ]

$id$ é o identificador do vetor, e $expr_arit$ é a posição no vetor.

**3.3. Atribuição**

Segue o padrão para atribuição:

PUT $expr_arit$ IN $id$ .
PUT $expr_arit$ IN $id$ [ $expr_arit2$ ] .

$id$ é o identificador da variável de destino, e $expr_arit$ é uma expressão aritmética que definirá o valor da variável. $id$ ainda pode ser indexado por $expr_arit2$ caso seja um vetor.

Ainda há o seguinte caso especial:

PUT $string$ IN $id$ .

$id$ é o identificador do vetor do tipo LETTER de destino, e $string$ é uma constante do tipo string. Neste caso, a i-ésima letra da string será o valor da i-ésima posição do vetor. Além disso, após a última letra, é colocado o valor 0 para indicar final de string.

Ex:
PUT 1 in a.
PUT “teste” in text.

**3.4. Expressões aritméticas**

Expressões aritméticas são operações sobre variáveis dos tipos NUMBER e/ou LETTER, posições de vetores, constantes numéricas e do tipo caractere utilizando-se operadores aritméticos (“+”, “-”, “*”, “/” e “%”) e parêntesis para alterar a precedência. Entre os operadores, “*”, “/” e “%” possuem a mesma precedência, que por sua vez é superior a precedência dos operadores “+” e “-”, que também possuem a mesma precedência. O operador “-” pode ainda ser utilizado para mudar o sinal de variáveis e constantes. IMPORTANTE: Expressões aritméticas não não um comando por si só, elas apenas fazem parte de outros comandos (ex: Atribuição).

Ex:
-a
-a[0]
(a)-b[c+d]
'a'+345
-(1)
a+b*c
(a+-b)*c
a+10*'b'

### 3.5. Expressões relacionais

$expr_rel$ é uma expressão relacional, que compara duas expressões aritméticas através de um operador relacional (“<”, “>”, “<=”, “>=”, “=” e “<>”). IMPORTANTE: Expressões relacionais não não um comando por si só, elas apenas fazem parte de outros comandos (ex: Desvio condicional).

Ex:
$expr_arit$ = $expr_arit$
$expr_arit$ <> $expr_arit$

### 3.6. Desvio condicional

Um desvio condicional segue o seguinte padrão:

```
IF $expr_rel$ THEN [
...
]
ELSE [
...
]
```

Um desvio condicional sempre possui um bloco IF, seguido ou não por um bloco ELSE. Cada bloco tem 0 ou mais comandos.

Ex:
```
IF a > b THEN [
        PUT a IN c.
]
ELSE [
        PUT b IN c .
```

]

### 3.7. Loop

Um loop de contagem segue o padrão:

```
FOR $id$ FROM $exp_arit1$ TO $exp_arit2$ DO [
...
]
```

$id$ é a variável utilizada como contador (pode ser também uma posição de um vetor). Ela é inicializada com o valor de $exp_arit1$ e é incrementada até alcançar o valor de $exp_arit2$, ou seja, o bloco de comandos é executado enquanto $id$ for menor ou igual a $exp_arit2$.

Outra opção é o loop vetorial:

```
FOREACH $id1$ IN $id2$ DO [
...
]
```

$id1$ é a variável (pode ser uma posição de um vetor) que receberá cada um dos valores contidos no vetor $id2$.

Ex:

```
for I from 1 to 10 do [
        put 0 in a[i].
]
foreach n in vet do [
        put soma+n in soma.
]
```

### 3.8. Leitura e escrita

A leitura é realizada da seguinte forma:

```
READ $id$ .
READ $id$ [ $exp_arit$ ] .
```

READ é inteligente o suficiente para ler um único caractere caso $id$ seja do tipo LETTER, uma string caso $id$ seja um vetor de LETTER, um único inteiro caso $id$ seja do tipo NUMBER, ou vários inteiros caso $id$ seja um vetor de NUMBER . Para impressão, segue-se o padrão:

```
PRINT $id$ .
PRINT $id$ [ $exp_arit$ ] .
PRINT $string$ .
PRINT $const_num$ .
PRINT $const_char$ .
```

PRINT imprime o valor da variável sem espaços ou quebras de linha. Caso $id$ seja um vetor de

NUMBER, todos os números são impressos separados por espaços entre chaves (ex: “{1 2 3 4}”). Se $id$ for um vetor de LETTER, uma string é impressa. Se $id$ for do tipo NUMBER, um número é impresso, e se for do tipo LETTER, o caracter correspondente é impresso. Também é possível utilizar a função PRINT para imprimir constantes, seguindo o mesmo comportamento anterior.

Ex:
PRINT a.
PRINT 100.
PRINT “flag; a; alfa”.
PRINT '\n'.