UNIVERSIDADE DO EXTREMO SUL CATARINENSE – UNESC
CURSO DE CIÊNCIA DA COMPUTAÇÃO
DISCIPLINA: COMPILADORES
PROFESSOR: GILBERTO VIEIRA DA SILVA

# ESPECIFICAÇÕES PARA LINGUAGEM SUBSET PASCAL

## LMS – características gerais

A linguagem "LMS", é definida a partir da linguagem pascal, portanto pode ser chamada de um subset do pascal; porém por ser uma linguagem com fins didáticos, possui várias restrições, apresentando apenas algumas das características básicas do Pascal.
As características da linguagem "LMS" são:

- Possui estruturas básicas para seleção e repetição, tais como: IF, WHILE, REPEAT, FOR, CASE.
- Permite definição de rótulos, constantes (numéricas) e variáveis do tipo inteiro.
- Suporta execução de procedimentos com passagem de parâmetros e recursão.
- Aceita desvios incondicionais (GOTO).
- É uma linguagem interpretada – a geração de código é direcionada para uma maquina hipotética que possui simulador.(interpretador).

## LMS - elementos Léxicos

Os caracteres válidos para a linguagem "LMS" são subdivididos em classes definidas a seguir:

- LETRAS – A até Z, a até z
- DIGITOS – 0 até 9
- OPERADORES ARITMÉTICOS : + - * /
- SINAIS RELACIONAIS: = < > <> >= <=
- SÍMBOLOS ESPECIAIS : , ; . ( ) : .. [ ] ´ _
- DELIMITADORES – Os caracteres branco, final de arquivo ou um comentário podem ser usados como separadores de tokens.
- COMENTÁRIOS – Um comentário pode ser inserido em qualquer lugar do programa onde um delimitador é válido. É delimitado por (* e *).
  Exemplo:

  (* compilador TURTLE *)

Assim os elementos léxicos serão definidos através do conjunto de caracteres e símbolos válidos para a linguagem "LMS". Com isto os elementos léxicos ou tokens, subdividem-se nas seguintes classes:

### Palavras reservadas

AND – ARRAY – BEGIN – CALL – CASE – CONST – DO – ELSE – END – FOR – GOTO – IF – INTEGER – LABEL - NOT - OF - OR - PROCEDURE - PROGRAM – READLN - REPEAT - THEN - TO - UNTIL - VAR - WHILE - WRITELN

### Identificadores

É definido por um conjunto de caracteres, alfanuméricos (máx. 30), sendo o primeiro caracter um alfabético, seguido de um conjunto de dígitos e/ou letras.

### Números (constantes)

Conjuntos de números definidos na faixa de –32.767 a 32.767. Somente inteiros são aceitos na linguagem LMS.

Ex:

19672
137
-1
19.67 ILEGAl, não aceita ponto decimal
32800 ILEGAL, valor fora da escala

### Literais

Sequência de caracteres (letras/símbolos/números) delimitados por apóstrofe, contendo não mais do que 255 caracteres.

Ex:
‘Compilador TURTLE para a LMS ‘
‘ ‘
‘ pato d´´agua’

### Operadores

Os operadores sejam eles aritméticos ou lógicos, dividem-se em categorias:

1- operador de negação: NOT
2- operador de multiplicação: * / AND
3- operador de adição: + - OR
4- operadores relacionais: < > = <= >= <>

## Construção do analisador léxico

- O analisador léxico é construído tendo como base um autômato finito.
- O compilador será implementado em mais de um passo, assim, o analisador léxico poderá identificar, nesta fase inicial, todos os tokens do fonte e mantê-los em uma área de memória temporária. Posteriormente, estas informações serão utilizadas pelo Analisador Sintático que iniciará a análise lendo os tokens identificados.

- Variáveis importantes :

  - CODIGO – armazenará um código numérico relativo ao token encontrado (conforme tabela fornecida acima).
  - CAR – armazenará ao próximo caráter a ser analisado.
  - VALOR – armazenará o valor na base 10 de constantes inteiras encontradas (token = inteiro).
  - BUFFER_IDENT – armazenará os identificadores encontrados (token = identificador).
  - BUFFER_LITERAL – armazenará a cadeia relativa a literais (token= literal)

- Rotinas importantes :

  - PEGACAR – pega o próximo caráter no programa fonte e o coloca na variável CAR
  - BUSCA_PALAVRA_RESERVADA – verifica se **token = identificador** está na tabela de palavras reservadas. Caso não esteja trata-se realmente do **token = identificador**. Se estiver trata-se de palavra reservada cujo código está na própria tabela.

**Tabela de código (para implementação) para os símbolos terminais (tokens)**

| Cod | Símbolo | Cod | Símbolo | cod | Símbolo | cod | símbolo |
|---|---|---|---|---|---|---|---|
| 1 | Program | 14 | Then | 27 | For | 40 | = |
| 2 | Label | 15 | Else | 28 | To | 41 | > |
| 3 | Const | 16 | While | 29 | Case | 42 | >= |
| 4 | Var | 17 | Do | 30 | + | 43 | < |
| 5 | Procedure | 18 | Repeat | 31 | - | 44 | <= |
| 6 | Begin | 19 | Until | 32 | * | 45 | < > |
| 7 | End | 20 | ReadIn | 33 | / | 46 | , |
| 8 | Integer | 21 | WriteIn | 34 | [ | 47 | ; |
| 9 | Array | 22 | Or | 35 | ] | 48 | literal |
| 10 | Of | 23 | And | 36 | ( | 49 | . |
| 11 | Call | 24 | Not | 37 | ) | 50 | .. |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 12 | Goto | 25 | Identificador | 38 | := | 51 | $ |
| 13 | If | 26 | Inteiro | 39 | : | | |

**Notas :**

- o programa deverá ser acoplado a um editor de forma que se possa criar/alterar o fonte
- o programa deverá permitir a visualização passo a passo do reconhecimento dos tokens.
- **A cadeia vazia será representada com código zero, ou seja, não será armazenada na pilha.**

**Exemplos de programas em LMS**

```
1) Program testeproc1;
Var
   X, y, z :integer;

Procedure P;
Var
   A  :integer;
Begin
   Readln(a);
   If a=x then
      z:=z+x
   Else begin
      Z:=z-x;
      Call p;
   End;
End;

Begin
   Z:=0;
   Readln(x,y);
   If x>y then
      Call p
   Else
      Z:=z+x+y;
   Writeln(z);
End.
```

2) **Program testeproc2;**
   Const a=2;

```
 Var x,y:integer;

Procedure p;
Var z: integer;

    Procedure q;
    Var t: integer;
     Begin (* inicio da q*)
     z:= z – 100 ; t:= z*a;
     if t > 100 then call q else writeln(t)
     end; (* fim de q*)

Begin (* inicio da P*)
    z:= x+y*a; if z> 100 then call q else writeln(z);
end; (* fim da p*)

begin (* programa principal*)

readln(x,y);
if x>1000 then x:= 1100
          else x::= y+100;
while x>y do begin call p; readln(x,y) end;
writeln(‘ tudo ok – boas férias ‘);
end.
```