# Análise Léxica – Parte II

Prof. Magnos Martinello
Março de 2009

# Definições Regulares

- Os números sem sinal em Pascal são cadeias tais como 5280, 39.37 e 1.87E-4. A seguinte definição regular corresponde a essa classe de cadeias:
  - dígito → 0 | 1 | ...| 9
  - dígitos → dígito+ /* um ou mais dígito */
  - fração_opcional → (.dígitos)? /* *? zero ou uma ocorrência* */
  - expoente_opcional → (E(+|-|)?dígitos)?
  - Num → digitos fração_opcional expoente_opcional

# O Reconhecimento de Tokens

Até o presente momento consideramos o problema de como especificar tokens. A partir de agora a pergunta é como reconhecer tokens ?

- Vamos considerar o seguinte fragmento de gramática:
- cmd → if *expr* then *cmd* | if *expr* then *cmd* else *cmd* | ε
- expr → *termo* relop *termo* | *termo*
- termo → id | num
- If → if
- then → then
- else → else
- Relop → < | <= | = | <> | > | >=
- Id → letra (letra | dígito)*
- Num → dígito+(.dígito+)? (E(+|-)?dígito+)?

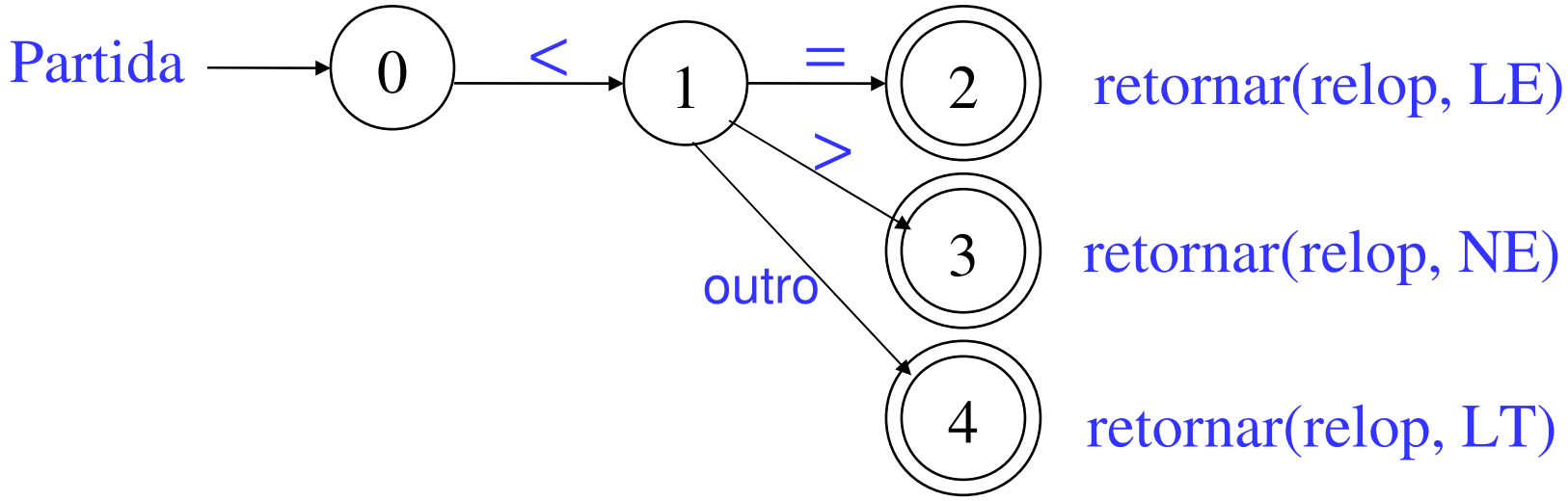
# O Reconhecimento de Tokens

Adicionalmente, assumimos que os lexemas sejam separados por espaços em branco, consistindo de sequências não nulas de espaços, tabulações e avanços de linha.

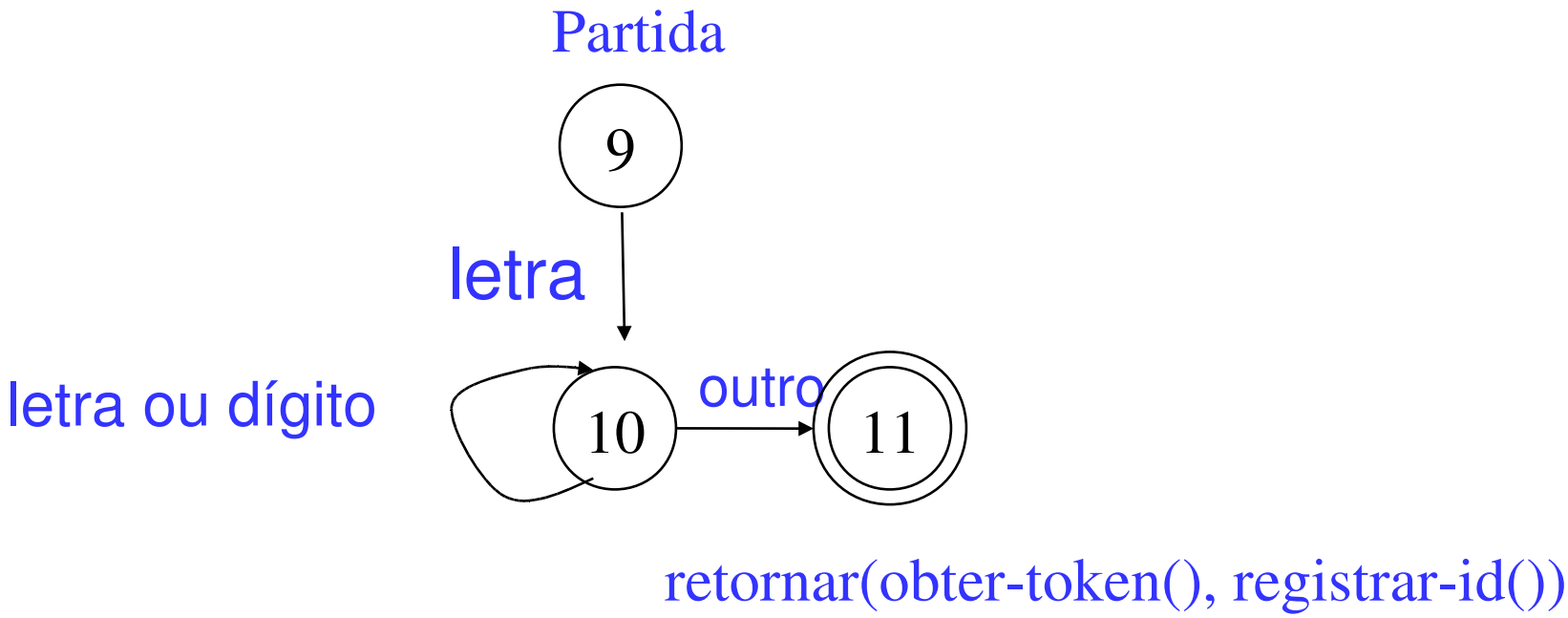
- *Delim → branco | tabulação | avança de linha*
- *WS → delim+*

# Diagrama de Transições

- Como passo intermediário para construção de um analisador léxico, diagrama de transições podem ser criados a fim de obter o próximo tok
- Outro: qualquer outro caracter que não esteja na linguagem (exemplo: espaço em branco, tabulação, quebra de linha)



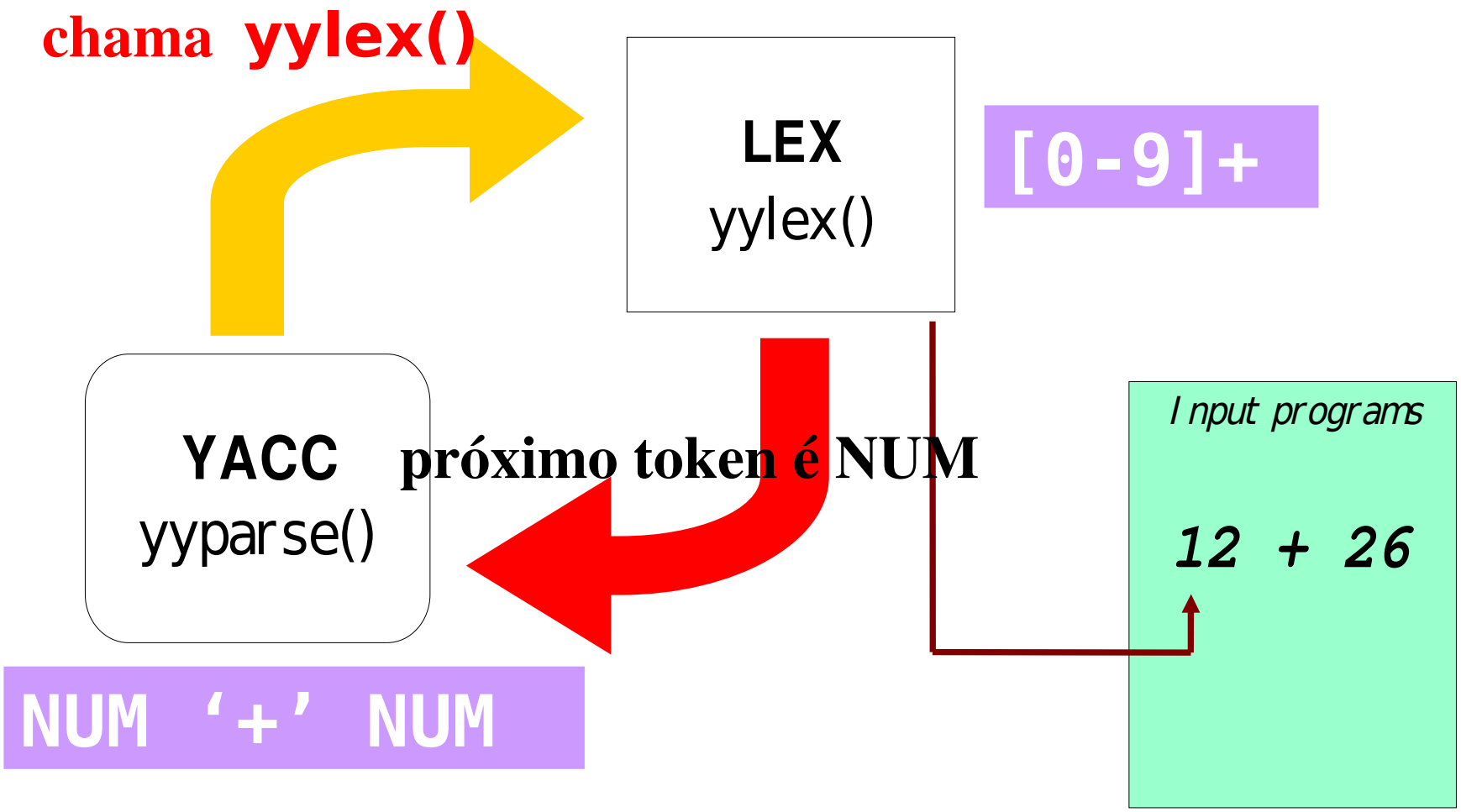
# Diagrama de Transições



# Ferramenta para especificação de analisadores léxicos (Flex)

- Ferramentas desenvolvidas no Bell Labs em meados dos anos 70
  - Flex = fast lexical analyser generator
  - Yacc – yet another compiler/compiler (futuro próximo)
- Entrada para o gerado (lexer)
  - Lista de expressões regulares em ordem de prioridade
  - Associar uma ação com cada expressão regular
- Saída
  - Programa que lê um fluxo de entrada e quebra em tokens de acordo com as REs

# Equipe FLEX and YACC(Bison)



# Programa no FLex

Um programa Flex é constituído de 3 partes:

Seção de Declarações

%%

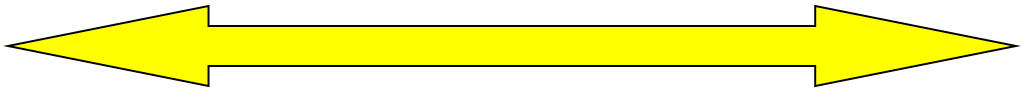
Regras de Tradução  (Produções)

%%

Procedimentos Auxiliares

# Programa parcial no Flex

```
D               [0-9]
%%
if              printf ("IF statement\n");
[a-z]+          printf ("tag, value %s\n", yytext);
{D}+            printf ("decimal number %s\n", yytext);
"++"            printf ("unary op\n");
"+"             printf ("binary op\n");
```



padrão ação

# Programa em Flex

```
%{
        #include <stdio.h>
        int num_lines = 0, num_chars = 0;
%}

%%
\n      ++num_lines; ++num_chars;
.       ++num_chars;

%%
main()
{
        yylex();
        printf( "# of lines = %d, # of chars = %d \n",num_lines, num_chars );
}
```

## Running the above program:

```
 [ 17 ] sandbox -: flex count.l
 [ 18 ] sandbox -: gcc lex.yy.c -lfl
 [ 19 ] sandbox -: a.out < count.l
 # of lines = 16, # of chars = 245
```

# Outro Programa em Flex

```
%{
/* recognize articles a, an, the */
#include <stdio.h>
%}

%%
[ \t]+              /* skip white space - action: do nothing */ ;
a |           /* | indicates do same action as next pattern */
an |
the           {printf("%s: is an article\n", yytext);}
[a-zA-Z]+       {printf("%s: ???\n", yytext);}

%%
main()
{
        yylex();
}
```

Observação: yytext é um ponteiro para o primeiro char do token de tamanho yyleng i.e yytext é uma variável usada nas ações que contém os caracteres aceitos pela expressão regular char yytext[yyleng]).

# Lex Regular Expression Meta Chars

| Meta Char | Meaning |
|---|---|
| . | match any single char (except \n ?) |
| * | Kleene closure (0 or more) |
| [] | Match any character within brackets |
| | - in first position matches - |
| | ^ in first position inverts set |
| ^ | matches beginning of line |
| $ | matches end of line |
| {a,b} | match count of preceding pattern from a to b times, b optional |
| \ | escape for metacharacters |
| + | positive closure (1 or more) |
| ? | matches 0 or 1 REs |
| \| | alteration |
| / | provides lookahead |
| () | grouping of RE |
| <> | restricts pattern to matching only in that state |