

# Utilitário Lex

O que faz/O que é

Como usar / Como funciona

Formato geral do Arquivo Submetido ao Lex

ER estendidas / Exemplos

The Lex & YACC page:

<http://dinosaur.compilertools.net/>

Flex: versão livre

<http://simplesamples.info/C++/Flex.php>

<http://contentbuilder.merlot.org/toolkit/html/snapshot.php?id=5373359979478>

A idéia é muito boa!

- Um gerador de programas...

## O que faz

- Lex gera programas (em C) para análises léxicas simples, incluindo analisadores léxicos de compiladores.
- Utiliza Expressões Regulares (ER) **Estendidas** para definir a especificação da análise léxica desejada

## Como Usar

1) editar arquivo (arq\_ent) a ser submetido ao lex que é da forma de uma tabela:

Padrão (ER estendida)	Ação código C
--------------------------	------------------

2) Lex arq\_ent → gera lex.yy.c

3) Cc lex.yy.c - ll → se tudo correr bem gera a.out

OBS: a biblioteca l fornece um main() que chama a função yylex(), assim não precisamos fazer o nosso programa principal.

4) a.out < teste\_arq\_ent > arq\_saida →  
gera a análise léxica

## Como funciona

- a.out copia a entrada (teste\_arq\_ent) para a saída (arq\_saida) exceto quando uma cadeia especificada como uma ER é encontrada na entrada, o que faz a ação associada a ela ser executada.
- Um exemplo de uma ação poderia ser imprimir o padrão encontrado:

ER                    ECHO

ou

ER                    printf("%s", yytext)

OBS: a cadeia que "casa" com a ER fica na variável acima, tem o tamanho copiado para `yytext` e ocorre na linha `yylineno`

## Regras para o casamento de padrões

- Lex privilegia a ER que dá o maior casamento
- Se houver 2 ou mais ER que casam o mesmo tamanho a ER que aparece 1o é privilegiada
- Temos que saber destas regras para escrever o arquivo de entrada!

# Exemplos

Suponha que tenhamos as regras:

<code>integer</code>	identificador pré-definido
<code>[a-z][a-z0-9]*</code>	identificador

- Se a entrada for `integer`, ambas regras casam e a 1a ganha
- Se a entrada for `integers` a 2a regra ganha pois casa com 8 caracteres
- Se a entrada for `int` a 2a ganha

## O que fazer com

- Identificadores e palavras reservadas que tem o mesmo padrão???
- A parte inteira de um número real que casa com números inteiros???

# Trecho de um arquivo Lex

...parte das definições ...

```
%%  
{DIGIT}+      {  
                printf("An integer: %s (%d)\n", yytext,  
                atoi(yytext));  
                }  
{DIGIT}+"."{DIGIT}*  {  
                printf("A float: %s (%g)\n", yytext,  
                atof(yytext));  
                }  
if|then|begin|end|procedure|function      {  
                printf("A keyword: %s\n", yytext);  
                }  
{ID}      printf("An identifier: %s\n", yytext);
```

..... regras continuam.....

## Respondendo

- Identificadores e palavras reservadas que tem o mesmo padrão???
  - Coloquem as reservadas primeiro, pois a primeira regra é privilegiada
- A parte inteira de um número real que casa com números inteiros???
  - Lex prefere os maiores casamentos, assim **2.3** vai casar somente com **{DIGIT}+"."{DIGIT}\***

# Formato Geral do arquivo submetido ao Lex

Definições

%%

Regras

%%

Subrotinas do usuário

Menor programa Lex:

%%

Copia a entrada para a saída sem alteração

OBS: as regras tem o formato **ER 1oumaisbrancos Ação**

O 2o %% é opcional e aparece somente se há subrotinas do usuário que são copiadas para o arquivo lex.yy.c

# Definições

- Usadas para simplificar as ER

nome definição



1a posição, segue padrão do C para variáveis

- A definição pode ser referenciada por {nome} que expandirá para {definição}
- Exemplo:

Digito [0-9]

Letra [a-zA-Z]

- Na parte de Regras, o uso das definições é assim:

%%

{Digito}+ "." {Digito}\* .....ação aqui.....

- que é idêntica ao padrão para buscar 1 ou mais dígitos seguidos de . seguidos de zero ou mais dígitos:

[0-9]+". "[0-9]\*

## Outros exemplos de definições úteis

a [aA]

b [bB]

...

z [zZ]

para serem usados na parte das Regras para  
**Pascal**:

{a}{n}{d}

{a}{r}{r}{a}{y}

...

# Definições

- Se usarmos o padrão abaixo nessa parte:

```
%{
```

```
%}
```

O que estiver entre os delimitadores vai ser copiado para a área externa de definição do `lex.yy.c`.

Exemplo:

```
%{  
/* need this for the call to atof() below */  
#include <math.h>  
/* need this for printf(), fopen() and stdin below */  
#include <stdio.h>  
%}
```

- Se usarmos os delimitadores na área das regras, antes das regras de casamento de padrão:
  - o código vai aparecer no `lex.yy.c` depois das declarações de variáveis da função `yylex()` e antes da primeira linha de código
- Isso é bom para definição de cabeçalhos, por exemplo:

TOKEN      CÓDIGO

Definições

```
%%
```

```
%{
```

```
printf("\n\n", "TOKEN      CÓDIGO")
```

```
%}
```

## Regras: Padrão Ação

- O padrão não deve ser indentado e a ação começa na mesma linha
- Um padrão é expresso com **ER estendidas** que podem conter os seguintes operadores:

" \ [ ] ^ - ? . \* + | ( ) \$ / { } % < >

Que para serem utilizados como texto comum devem aparecer como o caractere de escape \ ou entre aspas.

Exemplo: `xyz"++"` ou `"xyz++"` ou `xyz\+\+`

## ER estendidas

- Classe de caracteres
- Caractere .
- Opcionalidade
- Repetições
- União e agrupamento
- Sensibilidade ao contexto
- Repetições e Definições

## Classe de caracteres

- Podem ser especificadas utilizando o par de operadores [ ].
- [abc] associa a um simples caractere a possibilidade de ser "a", "b", ou "c"
- Dentro dos colchetes, o significado dos operadores são ignorados.
- Somente 3 caracteres são especiais: \ - ^
  - Indica intervalo, por exemplo: [a-z0-9\_]Se quiser incluir "-" ela deve ser a primeira ou a última. Por exemplo: [-+0-9]

## Classe de caracteres

^ deve aparecer como o primeiro caractere depois do colchete esquerdo indicando o complemento. Por exemplo:

[^abc] associa todos os caracteres exceto "a" "b" ou "c", incluindo todos os especiais e de controle.

[\40-\176] associa todos os caracteres imprimíveis do conjunto ASCII (uso de octais)

## O caractere . (ponto)

O . associa quase todos caracteres.

É a classe de todos menos \n

Tomem cuidado ao usá-lo.

Geralmente é o último a ser usado num arquivo submetido ao lex.

## Expressões Opcionais & Repetições

- O operador ? Indica um elemento opcional de uma expressão
- Por exemplo:  $ab?c$  associa "ac" ou "abc"
- Repetições são indicadas pelo operador "+" ou "\*"
- $a^*$  indica zero ou mais a's consecutivos
- $a^+$  indica 1 ou mais instâncias de a's
- Exemplo:  $[a-z]^+$

## União e Agrupamento

- O operador | indica união.
- Por exemplo:  $(ab|cd)$  associa "ab" ou "cd".
- Os parênteses são utilizados para agrupar embora não necessários a não que queiramos mudar a ordem de precedência dos operadores

## Sensitividade ao contexto

- Embora pequena, Lex usa dois operadores para isso:  $\wedge$   $\$$
- Se o primeiro caractere de uma expressão é  $\wedge$  a expressão somente é associada se estiver no começo de uma linha
- Se o último caractere é  $\$$  a expressão é associada somente se estiver no fim da linha
- A expressão "ab\$" é a mesma de "ab\n"

## Repetições e o separador %

- Os operadores `{ }` especificam ou repetições (se fecham números) ou expansão de definições (já vista).
- `a{1,5}` procura por 1 a 5 ocorrências de `a`
- `a{2,}` duas ou mais ocorrências
- `a{4}` exatamente 4 ocorrências
- `%` separa trechos de fontes Lex

# ER estendidas

$x$  the character "x"

"x" an "x", even if x is an operator.

$\backslash x$  an "x", even if x is an operator.

$[xy]$  the character x or y.

$[x-z]$  the characters x, y or z.

$[\hat{x}]$  any character but x.

$.$  any character but newline.

$\wedge x$  an x at the beginning of a line.

$\langle y \rangle x$  an x when Lex is in start condition y.

$x\$$  an x at the end of a line.

$x?$  an optional x.

$x^*$  0,1,2, ... instances of x.

$x^+$  1,2,3, ... instances of x.

$x|y$  an x or a y.

$(x)$  an x.

$x/y$  an x but only if followed by y.

$\{xx\}$  the translation of xx from the definitions section.

$x\{m,n\}$  m through n occurrences of x

## Ações Lex

- A ação mais simples é ignorar a entrada usando o comando nulo do C.
- Exemplo:

[ \t\n] ;

" " |  
"\t" |  
"\n" ;

OU

As aspas não  
são  
necessárias!

## Exemplo de arquivo Lex - 1

```
%{  
/* need this for the call to atof() below */  
#include <math.h>  
/* need this for printf(), fopen() and stdin  
   below */  
#include <stdio.h>  
%}
```

```
DIGIT [0-9]
```

```
ID [a-z][a-z0-9]*
```

```

%%
{DIGIT}+          {
    printf("An integer: %s (%d)\n", yytext,
    atoi(yytext));
}

{DIGIT}+"."{DIGIT}* {
    printf("A float: %s (%g)\n", yytext,
    atof(yytext));
}

if|then|begin|end|procedure|function      {
    printf("A keyword: %s\n", yytext);
}

{ID}          printf("An identifier: %s\n", yytext);
"+"|"-"|"*"|"|" /"      printf("An operator: %s\n", yytext);
"{"[^\n]*}"      ; /* eat up one-line comments */
[ \t\n]+      ; /* eat up white space */
.      printf("Unrecognized character: %s\n", yytext);

```

```
%%
```

```
int main(int argc, char *argv[])
```

```
{
```

```
    ++argv, --argc; /* skip over program name */
```

```
    if (argc > 0)
```

```
        yyin = fopen(argv[0], "r");
```

```
    else
```

```
        yyin = stdin;
```

```
    yylex();
```

```
}
```

## Exercício

- Modelar o reconhecimento de TOKENS da linguagem C, via ER estendidas do Lex
- Testar o reconhecimento com programas em C cuja saída do lex tabule cada token e seu código.