

[MAC0211] Laboratório de Programação I  
Aula 21  
Flex (Gerador de Analisadores Léxicos)  
Bison (Gerador de Analisadores Sintáticos)

Kelly Rosa Braghetto

DCC-IME-USP

23 de maio de 2013

## Na aula passada...

- ▶ Awk (Parte 2)
- ▶ Geradores de Analisadores Léxicos – Flex (introdução)

## [Aula passada] Geradores de Analisadores

- ▶ São programas que recebem como entrada um arquivo com uma descrição em alto do que o analisador deve reconhecer e que geram como saída um código que, depois de compilado/interpretado, vira um programa analisador
- ▶ Exemplos:
  - ▶ **geradores de analisadores léxicos**
    - ▶ para C: Lex, Flex
    - ▶ para Java: JFlex
  - ▶ **geradores de analisadores sintáticos**
    - ▶ para C: Yacc, Bison
    - ▶ para Java: JavaCC, ANTLR

Geralmente, são usados em conjunto: Lex + YACC, Flex + Bison

## [Aula passada] Flex – *Fast Lexical Analyzer Generator*

- ▶ Implementação gratuita e de código aberto do Lex
- ▶ Distribuído pelo projeto GNU, mas não é parte dele
- ▶ **Entrada do programa:** arquivo contendo uma tabela de expressões regulares e suas respectivas ações associações
- ▶ **Saída do programa:** o código fonte em C de um analisador léxico que reconhece as expressões regulares especificadas no arquivo de entrada

## Formato de um arquivo de entrada para o Flex

O arquivo é dividido em três partes, separadas por linhas que começam com `%%`:

```
[Definições]
```

```
%%
```

```
[Regras]
```

```
%%
```

```
[Código do usuário]
```

## [Aula passada] Flex – Definições [parte do início]

A seção de *Definições* pode conter:

- ▶ código “**literal**” que deve aparecer no analisador fora de qualquer função. Esse código deve ser delimitado por `%{ %}` ou então deve aparecer indentado
- ▶ declarações de **definições de nomes simples**. São feitas no formato:

`[nome] [definição]`

Exemplos:

`DIGITO [0-9]`

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

- ▶ declarações de **condições de início** para regras adicionais ou regras exclusivas. São feitas nos seguintes formatos:
  - ▶ condição para regras adicionais: `%s [nome]`
  - ▶ condição para regras exclusivas: `%x [nome]`

(Veremos mais detalhes sobre o seu uso mais adiante)

## Flex – Código do Usuário [parte do fim]

- ▶ Na seção de *Código do Usuário*, você pode colocar a função `main()` do seu programa ou outras funções que você desejar escrever
- ▶ Todo o código que é colocado nessa seção, juntamente com o que é colocado entre `%{ %}`, é copiado para o arquivo de saída gerado pelo Flex
- ▶ Essa seção é opcional; na ausência dela, não é necessário colocar o separador `%%` correspondente no arquivo de entrada para o Flex

## Flex – Regras [parte do meio]

- ▶ A seção de *Regras* contém um conjunto de regras que definem os tipos léxicos que serão entendidos pelo analisador; o Flex usa essas regras para gerar o código do analisador
- ▶ Uma regra é um par [padrão – ação]
- ▶ Um padrão é definido por meio de uma expressão regular
- ▶ Uma ação é um trecho de código em C
- ▶ Uma ação é executada sempre que o analisador léxico encontrar no texto de entrada uma cadeia de caracteres que casa com o padrão associado à ação
- ▶ Formato:  
[padrão] [ação]  
sendo que um novo padrão tem que aparecer sempre em uma nova linha, sem indentação, e a definição da ação correspondente tem que começar na mesma linha

## Flex – exemplo simples (arquivo conta\_lpc.lex)

```
/* Este código gera um programa que faz a contagem de linhas
   e caracteres de um arquivo texto da entrada padrao      */
%{
int contLinhas = 0, contCaracteres = 0;
%}

%%
\n      ++contLinhas; ++contCaracteres;
.       ++contCaracteres;
<<EOF>> yyterminate();

%%
main() {
    yylex();
    printf( "# linhas: %d, # caract. (incluindo '\\n') = %d\n",
            contLinhas, contCaracteres);
}
```

## Flex – como usar

- ▶ Execute o Flex fornecendo um arquivo de entrada com as regras:  
`$ flex contlc.lex`
- ▶ O Flex gerará como saída um arquivo chamado `lex.yy.c`, que contém o código fonte do analisador léxico gerado
- ▶ A função `yylex()` faz a análise léxica propriamente dita; é por essa razão que ela é sempre chamada no código da função `main()`
- ▶ Compile o arquivo `lex.yy.c` e ligue-o à biblioteca do Flex (a `libfl`), para gerar o programa do analisador  
`$ gcc lex.yy.c -o contlc -lfl`

## Flex – variáveis internas pré-definidas

- ▶ `char *yytext` : é um apontador para uma cadeia de caracteres contendo o último item léxico reconhecido
- ▶ `int yyleng` : é o comprimento do último item léxico reconhecido
- ▶ `FILE *yyin` : é o descritor do arquivo de entrada que será processado

## Flex – mais um exemplo simples

```
/* Conta linhas, palavras e caracteres de um arquivo texto
   cujo o nome sera passado via linha de comando */
%{
int contCaracter = 0, contPalavra = 0, contLinha = 0;
%}

EOL      \n
PALAVRA  [^ \n\t]+

%%
{PALAVRA}      {contPalavra++; contCaracter += yyleng;}
{EOL}          {contCaracter++; contLinha++;}
.             {contCaracter++;}

%%
int main(int argc, char** argv){
    yyin = fopen(argv[1], "r"); /* abre arq de entrada */
    yylex();                    /* executa o scanner */
    printf("# linhas: %d, # palavras: %d, # caracteres: %d\n",
           contLinha, contPalavra, contCaracter);
    fclose(yyin);              /* fecha arq de entrada */
}
```

## Flex – Um exemplo mais complexo

Ver exemplo do analisador léxico para uma linguagem parecida com  
Pascal.  
(arquivo `pascal.lex` no diretório de exemplos no Paca)

## Expressões regulares no Flex

São similares às expressões estendidas do Unix e do Awk.

Novidades:

<code>{nome}</code>	expande a definição <i>nome</i>
<code>"[xyz]\“foo“</code>	string literal <code>'[xyz]h“foo'</code>
<code>\123</code>	o caracter cujo código em octal é 123
<code>\x2a</code>	o caracter cujo código em hexadecimal é 2A
<code>r/s</code>	um <i>r</i> , mas só se ele for sucedido por um <i>s</i> . <i>s</i> é verificado pela regra, mas não é consumido da entrada
<code>&lt;&lt;EOF&gt;&gt;</code>	um fim de arquivo
<code>&lt;c&gt;r</code>	um <i>r</i> , mas somente na condição de início <i>c</i>
<code>&lt;c1,c2,c3&gt;r</code>	um <i>r</i> , mas em qualquer uma das condições <i>c1</i> , <i>c2</i> ou <i>c3</i>
<code>&lt;*&gt;r</code>	um <i>r</i> , em qq condição de início (mesmo nas exclusivas)

**Obs.:** considere *r* e *s* expressões regulares

## Flex – condições de início

- ▶ O Flex provê mecanismos para a habilitação condicional de regras
- ▶ Regras condicionais são habilitadas por uma **condição de início**
- ▶ **condições de início para regras adicionais** – habilitam um conjunto de regras que serão executadas adicionalmente às regras não condicionais
- ▶ **condições de início para regras exclusivas** – habilitam um conjunto de regras que serão executadas de forma exclusiva (ou seja, as demais regras de fora do conjunto não serão executadas)

## Flex – condições de início (exemplo)

```
/* Declaracao de uma condicao de inicio
   para regras exclusivas */
%x  STRING

%%
\"    {
    printf(" string ");
    /* habilita todas as regras que comecam com <STRING> */
    BEGIN(STRING);
    }
<STRING>[~"] ; /* nenhuma acao p/ um carac. <> aspas */
<STRING>\\"    {
                /* desabilita a condicao de inicio STRING */
                BEGIN(INITIAL);
    }
}
```

## Flex – um exemplo completo

Veja o arquivo `conversor.lex`, disponível no Paca, para a geração de um *scanner* que varre um texto, substituindo valores monetários em Real por valores em Dólar.

## Bibliografia e materiais recomendados

- ▶ Manual do Flex  
<http://flex.sourceforge.net/manual/>
- ▶ Notas das aulas de MAC0211 de 2010, feitas pelo Prof. Kon  
<http://www.ime.usp.br/~kon/MAC211>

# Cenas dos próximos capítulos...

## Na próxima aula

- ▶ Gerador de Analisadores Sintáticos – o GNU Bison