
# UNIVERSIDADE FEDERAL DO TOCANTINS

**CURSO DE CIÊNCIA DA COMPUTAÇÃO**

**PESQUISA BIBLIOGRÁFICA SOBRE GERADORES DE ANALISADORES LÉXICOS**

**MAIO/ 2016**

# UNIVERSIDADE FEDERAL DO TOCANTINS

**PESQUISA BIBLIOGRÁFICA SOBRE GERADORES DE ANALISADORES LÉXICOS**

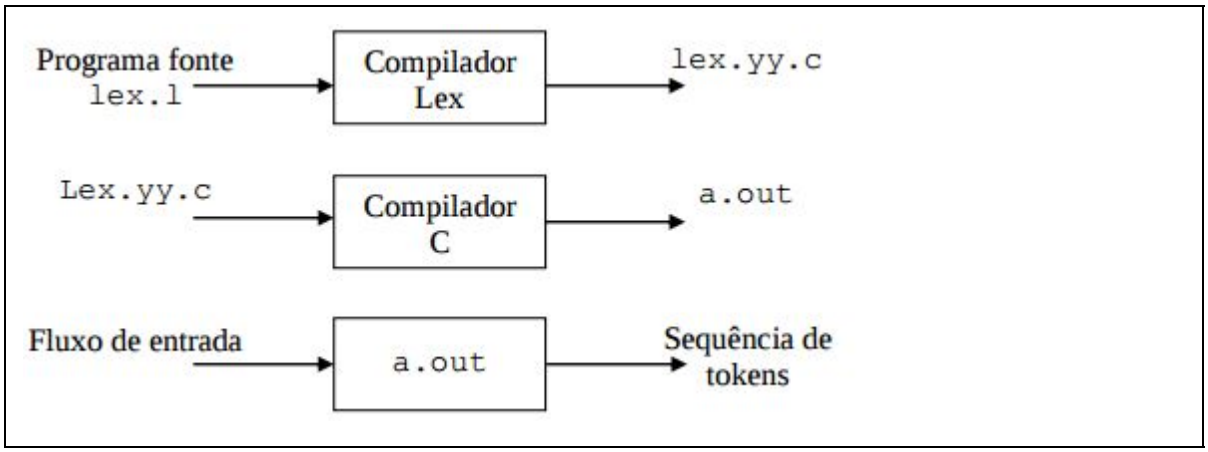
**ALUNO: IDNILSON NUNES DE AGUIAR**
**E GUSTAVO MACEDO**
**PROFESSOR: ALEXANDRE ROSSINI**
**DISCIPLINA: COMPILADORES**

**MAIO/ 2016**

GERADORES DE ANALISADORES LÉXICOS – LEX / FLEX

Lex ou Flex, permite especificar um analisador léxico definindo expressões regulares para descrever padrões para os tokens. Existem muitas versões diferentes de Lex. A versão mais popular é denominada Flex (Fast Lex). Ela é distribuída como parte do pacote de compilação Gnu, produzido pela Free Sfotware Foundation, e está disponível gratuitamente em diversos endereços na internet. Lex é um programa que recebe como entrada um arquivo de texto contendo expressões regulares, juntamente com as ações associadas a cada expressão. A notação de entrada para a ferramenta Lex é chamada de linguagem Lex, e a ferramenta em si é o compilador Lex. O compilador Lex transforma os padrões de entrada em um diagrama de transição e gera código em um arquivo chamado lex.yy.c, que simula esse diagrama de transição.

A figura abaixo ilustra como criar um analisador léxico com o Lex:




Um arquivo de entrada, Lex.1, é escrito na linguagem Lex e descreve o analisador léxico a ser gerado. O compilador Lex transforma Lex.1 em um programa C, e o armazena em um arquivo que sempre se chama Lex.yy.c. Esse último arquivo é compilado pelo compilador C em um arquivo sempre chamado a.out. A saída do compilador C é o analisador léxico gerado, que pode receber como entrada um fluxo de caracteres e produzir como saída um fluxo de tokens.

O LEX / FLEX servem para gerar automaticamente programas (usualmente em “C”) fazendo a leitura de uma entrada, de modo a varrer um texto e/ou programa (“scanners”) a fim de obter uma sequência de unidades léxicas (“tokens”). Os tokens gerados pelos programas criados pelo LEX/FLEX serão usualmente processados posteriormente por um programa que realizará a analise sintática.

Lex => Gerador de analisadores léxicos (UNIX – Ex.: Lex AT&T, Berkeley BSD)
Flex => Gerador de analisadores léxicos (LINUX / Windows-DOS – GNU Lex)

No LEX, a especificação de entrada é formada de expressões regulares e comandos em linguagem de programação. O analisador é gerado de forma que quando uma dada

expressão é reconhecida, os comandos associados com ela são executados. Isto pode facilitar o trabalho a princípio, mas juntar a especificação com o código objeto torna a leitura da primeira mais difícil. Além de que para se utilizar esta especificação para gerar um analisador em uma outra linguagem é preciso altera-la.

O Flex lê os arquivos de entrada especificados, ou a entrada padrão se nenhum arquivo for especificado, obtendo assim uma descrição do scanner a ser gerado. Este arquivo de entrada é o que chamamos arquivo de definição ou arquivo de descrição. A descrição é realizada na forma de pares de expressões regulares e código C. Tais pares são denominados regras. As regras definem simultaneamente quais padrões devem ser procurados e quais as ações que devem ser executada quando da identificação deste padrão, ou seja, para cada padrão desejado pode ser associado um conjunto de ações escritas sob a forma de código C. Flex gera como saída um arquivo fonte em linguagem C, cujo nome é lex.yy.c, no qual é definida a função yylex() e as variáveis global yytext e yyleng. A função yylex() é na verdade o analisador léxico gerado pelo arquivo de definição através do Flex. A variável global yytext contêm o texto do padrão reconhecido (uma string) no momento enquanto yyleng contêm o número de caracteres de tal string, podendo ambas serem usadas no trechos de código C que definem as ações associadas a cada padrão.

Um arquivo de entrada Lex é composto por três partes: coleção de definições (ou declarações), coleção de regras de tradução e coleção de rotinas auxiliares (ou rotinas de usuário). As três seções são separadas por dois sinais de porcentagem que aparecem em linhas separadas iniciando na primeira coluna.

```
{definições}
 %%
Contém declarações de variáveis , includes e constantes
Código nesta seção é diretamente copiado para código na linguagem alvo
----------------------------------------
{regras}
%%
p1 {ação}
   ... ....
pn {ação}
Onde, pi é uma expressão regular e cada ação é um fragmento de programa descrevendo a ação a ser tomada quando o padrão for reconhecido
----------------------------------------
{rotinas auxiliares}
Contém procedimentos auxiliares que seja necessário para execução das ações.
Código nesta seção é diretamente copiado para código na linguagem alvo
```

A seção de definições inclui declarações de variáveis, constantes e definições regulares. Cada uma das regras de tradução possui o formato padrão {ação} Cada padrão é uma expressão regular, que pode usar as definições regulares da seção de declaração. As ações são fragmentos de código, normalmente escritos em C. A terceira seção contém

quaisquer funções adicionais usadas nas ações. Essa seção pode também conter um programa principal, se quisermos compilar a saída Lex como um programa independente.

**Usando o FLEX:**

1. FLEX -o<arq_saida.c> <arq_def>.1

   *.l => Arquivos que contêm as definições das unidades léxicas a serem identificadas Contém um conjunto de especificações de expressões regulares que serão usadas para reconhecer os tokens.

   *.c => Arquivo do programa “C” que implementa o analisador léxico especificado.

   Principais opções do FLEX:

   -i => Case Insensitive (ignora diferença entre maiúscula/minúsculas) --version => Exibe a versão atual do programa flex em uso -+ => Geração de código de saída em C++

2. GCC <arq_saida.c> -o<arq_executavel> -lfl

   Observações importantes sobre a geração do programa de análise léxica:

   - É necessário linkar (-l) uma bibliotec (“lib”) do analisador léxico na compilação do código gerado. O FLEX usa a lib “fl” e o LEX usa a lib “l”.

   - O programa gerado pode ser executado, onde usualmente a entrada do texto a ser analisado é feita pela “stdin” (teclado).

3. Executar o programa gerado.

**EXPRESSÕES REGULARES – Usada pelo LEX / FLEX:**

[0-9] => Reconhece um dígito
[a-zA-Z] => Reconhece uma letra (comum = sem acentos)
[\ \t\n] => Reconhece um espaço em branco ou um tab ou uma nova linha
xxxxx => Reconhece a seqüência de caracteres “xxxxx”

| Símbolos especiais: | Exemplo: |
|---|---|
| + => 1 ou mais ocorrências | [0-9]+ => Um número |
| * => 0 (nenhuma) ou mais ocorrências | [0-9][0-9]* => Um número |
| ? => 0 (nenhuma) ou apenas 1 ocorrência | -?[0-9]+ => Um número com/sem sinal |

\n => Reconhece a marca de fim de linha / nova linha
. => Aceita um caracter qualquer de entrada
xxx$ => Reconhece xxx se for seguido de um fim de linha
^xxx => Reconhece xxx se este estiver imediatamente após o início de uma linha [^x] => Reconhece qualquer caracter menos “x”
[xyz] => Reconhece um dos caracteres “xyz” indicados
[a-z] => Reconhece um caracter pertencente ao intervalo de “a-z”
x{n} => Reconhece um número exato “n” de ocorrência de “x”
x{n,} => Reconhece a ocorrência de no mínimo “n” vezes de “x”

x{n,m}=> Reconhece a ocorrência de “x” entre no mínimo “n” e no máximo “m” vezes
xx|yy => Reconhece a ocorrência de “xx” ou de “yy”
(x|y) => Agrupa (sub)expressões regulares
“x” => Reconhece exatamente o caracter “x” (usado com caracteres especiais). Ex.: “+”
Exemplos de expressões regulares simples:
DIGITO [0-9]
LETRA [a-zA-Z]
ESPACO [\ \t\n]
INTEIRO [0-9]+
INTSIGNED -?[0-9]+
DECIMAL [0-9]*\.[0-9]+ => aceita .33 / não aceita números sem casas decimais INTOUDEC ([0-9]+)|([0-9]*\.[0-9]+)
IOUDSIGNED -?(([0-9]+)|([0-9]*\.[0-9]+))
NOMEVAR [a-z][a-z0-9_]* => usando opção “case insensitive”.

**Outros geradores analisadores léxicos.**

Microlex

O programa possui uma interface com o usuário que lhe permite criar/modificar o arquivo de entrada, bem como fazer testes com ele antes de gerar o analisador. A forma de o usuário entrar com a especificação léxica da linguagem é através de expressões regulares, são definidas expressões para cada token da linguagem e o Microlex se encarrega da geração do autômato finito para a identificação destes. O gerador final pode ser obtido em três linguagens: Object Pascal, C++ e Java, e ainda é possível escolher duas possíveis técnicas para o gerador: através de tabelas de análise, com uma rotina auxiliar, ou com o autômato programado diretamente na linguagem objeto

Bison

Bison é um software do grupo GNU (Software Livre) disponibilizado para quase todos os sistemas operacionais, que se trata de um gerador automático de analisadores sintáticos, no caso gerador de ‘parser’, descendente da ferramenta ‘Yacc (Yet Another Compiler Compiler)’, que também é um gerador de analisador sintático desenvolvido por Stephen C. Johnson da AT&T para o sistema operacional Unix, aonde oferece muitas melhorias. Utilizado junto com ferramentas de analisadores ‘Flex’ (Analisador léxico Flex).
A ferramenta ‘Bison’ é: “um gerador de interpretadores para fins gerais que converte uma descrição gramatical de uma gramática livre de contexto ‘LALR’ para um programa C que analisa aquela gramática. Uma vez que você esteja acostumado com ‘Bison’, você pode usá-lo para desenvolver um ampla quantidade de interpretadores de linguagem, daqueles usados em simples calculadoras de mesa até linguagens de programação complexas. Bison tem compatibilidade ascendente com ‘Yacc’: todas as gramáticas válidas para ‘Yacc’ devem funcionar com ‘Bison’ sem alterações. “Qualquer pessoa familiarizada com Yacc deve conseguir usar Bison sem maiores problemas.”

Usando Bison

O bison transforma esta descrição da gramática e ações associadas em um parser (programa capaz de analisar uma sequência de tokens de entrada, detectar produções e agir sobre elas). São 4 passos para criar um parser: Escrever uma especificação de uma gramática no formato do bison. O arquivo terá a extensão .y. Escrever uma especificação de um analisador léxico que pode produzir tokens; extensão .l. Executar o bison sobre a especificação .y e o flex sobre a especificação .l. Compilar e lincar os códigos fontes do parser, do analisador léxico e suas rotinas auxiliares.

**Diferença do LEX/FLEX**

Uma diferença simples é que o Flex é a ferramenta preferencial para gerar a primeira fase de um compilador. Já o LEX é quase sempre acoplado com o YACC, que efetua a fase de análise sintática. O yacc funciona de forma muito parecida e chama o yyflex().

**Exemplos de uso do LEX/FLEX**

Exemplo: programa Lex adicionar números de linhas a linhas de um texto, e imprimir o novo texto para a saída padrão.

```
%{

/*
o programa adiciona números de linhas de um texto
e imprime o novo texto na tela (ou um arquivo de
redirecionamento)
*/

#include <stdio.h>
int linha = 1;
%}
linha .*\n
%%
{linha} {printf("%5d %s", linha++, yytext);}
%%
main()
{
  yylex();
  return 0;
}
```

Comentário do código acima:

Qualquer texto entre %{ e %} na seção de definição será copiado diretamente no programa de saída externamente a todos os procedimentos. Qualquer texto na seção de procedimentos auxiliares será copiado diretamente no programa de saída no final do código Lex. Qualquer código seguinte a uma expressão regular (separado dela por pelo menos um espaço) na seção de ações (após o primeiro %%) será inserido no ponto apropriado no procedimento de reconhecimento yylex e executado quando ocorrer um casamento com a

expressão regular correspondente. O código em C que representa uma ação pode ser uma declaração única em C ou uma declaração composta por declarações apresentadas entre chaves

Outro exemplo:

```
1  /* separe a primeira definicao por um tab */
2   int numeroDeLinhas = 0, numeroDeCaracteres = 0;
3  %%
4  \n{
5   /* incrementa numero de linhas */
6   ++numeroDeLinhas;
7   /* incrementa numero de caracteres */
8   ++numeroDeCaracteres;
9   }
10
11 .{
12  /* incrementa numero de caracteres */
13  ++numeroDeCaracteres;
14  }
15 %%
16 /* recomendavel declarar sempre funcao yywrap() */
17 int yywrap ();
18 main() /* programa principal */
19 {
20   yylex(); /* scanner gerado por Flex */
21   printf("Numero de Linhas = %d\n", numeroDeLinhas);
22   printf("Numero de Caracteres = %d\n",numeroDeCaracteres);
23 }
24
25 int yywrap() {
26  return 1;
27 }
```

Comentário do código acima:

Na linha 18, temos declarada uma função main(), que define o início do programa que pode efetuar uma chamada a função yylex() (o scanner gerado por Flex). Outras funções, utilizadas nas ações definidas pelas regras, podem ser colocadas. Este scanner conta o número de caracteres e o número de linhas da entrada fornecida, produzindo como saída apenas duas mensagens informando o valor dos contadores de caracteres e linhas. Neste exemplo a seção DEFINIÇÕES contêm apenas a declaração de duas variáveis globais numeroDeLinhas e numeroDeCaracteres, ambas acessíveis para a função yylex() que será construída pelo Flex e para main() declarado na seção CÓDIGO. Na seção REGRAS temos a declaração de duas regras: Uma para o caractere "\n" que incrementa tanto o número de

linhas como o número de caracteres lidos. outra para os demais caracteres, indicada pelo caractere "." que incrementa o número de caracteres lidos. Note que o caractere "." tem o comportamento de uma cláusula default de uma diretiva switch da linguagem C.

**Referências bibliográficas:**


https://projetos.inf.ufsc.br/arquivos_projetos/projeto_353/Gals.pdf
https://erinaldosn.files.wordpress.com/2011/04/aula-8-lex.pdf
http://osorio.wait4.org/oldsite/compil/aula-lex-flex.pdf
https://projetos.inf.ufsc.br/arquivos_projetos/projeto_353/Gals.pdf