[MAC0211] Laboratório de Programação I
Aula 22
Flex (Gerador de Analisadores Léxicos)
Bison (Gerador de Analisadores Sintáticos)

Alair Pereira do Lago

DCC–IME–USP

26 de maio de 2015

Na aula passada...

- Geradores de Analisadores Léxicos – Flex (Parte 1)

[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 – 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 */
}
```

[Aula passada] Expressões regulares no Flex

São similares às expressões estendidas do Unix e do Awk.
Novidades:

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

```
/* Gera um programa que substitui cadeias delimitadas por aspas
   pela palavra "string" */

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

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

Flex – mais um exemplo usando condições de início

Veja o arquivo `remove_comentario.l`, disponível no Paca, para a geração de um *scanner* que varre um texto, eliminando comentários definidos no estilo da linaguagem C.

Flex – um exemplo completo

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

Mudando de assunto...

Geradores de analisadores sintáticos

- O UNIX original tinha como um de seus componentes, o YACC – *Yet Another Compiler Compiler*
- O YACC gera um **analisador sintático** (= ***parser***) a partir de uma especificação formal de uma gramática
- O **GNU Bison** é a implementação da FSF (*Free Software Foundation*) para o YACC e é completamente compatível com ele, mas inclui novas funcionalidades
- Usando o Bison, podemos criar de simples programas interativos (como calculadoras) a analisadores de linguagens de programação bem complexas

O GNU Bison

- O Bison recebe como entrada a especificação de uma gramática no formato BNF (*Backus-Naur Form*)
- Ao definirmos uma gramática formal para uma linguagem, cada tipo de unidade sintática ou agrupamento é associado a um símbolo
- Símbolos que são definidos a partir do agrupamento de símbolos menores são chamados de **símbolos não-terminais**
- Símbolos que não podem ser subdivididos são chamados de **símbolos terminais** (= *tokens*)

Exemplo na linguagem C

- Símbolos não-terminais: expressão, sentença (*statement*), declaração e definição de função
- Símbolos terminais: identificador, número, string, if, while, do, for, continue, break, return, cont, static, int, char, float, double, +, -, *, /, %, &, &&, =, ==, {, }, [,], ", ', etc.

(Um "parênteses" para relembrar um exemplo de gramática)

Gramática da calculadora infixa vista na Aula 16

```
<programa> ::= <programa> <expr> ; | <expr> ;
<expr> ::= <expr> + <termo> | <expr> - <termo> | <termo>
<termo> ::= <termo> * <fator> | <termo> / <fator> | <fator>
<fator> ::= NUMERO | -<fator> | (<expr>) |
            K[<expr>] | K[<expr>] := <expr>
```

Estrutura de um arquivo de entrada do Bison

É análoga à entrada do Flex:

```
%{
    Prólogo
%}
Declarações do Bison
%%
Regras gramaticais
%%
Epílogo
```

Estrutura de um arquivo de entrada do Bison

- **Prólogo** – contém definições iniciais em C que são jogadas diretamente para o arquivo de saída do Bison, antes da definição da função `yyparse` (que é o analisador sintático em si)
- **Declarações do Bison** – onde são definidos símbolos terminais e não-terminais e relações de precedência. Em gramáticas simples, pode ser deixado vazio
- **Regras gramaticais** – é o coração do arquivo. É obrigatório ter pelo menos 1 regra gramatical
- **Epílogo** – é copiado *ipsis literis* para o arquivo de saída

Descrição da gramática no Bison – símbolos

- Um símbolo não-terminal é representado por um identificador (como um identificador na linguagem C). Por convenção, usa-se somente letras minúsculas, como em `expressao`, `comando` e `declaracao`
- Um símbolo terminal (ou *token*) pode ser representado por:
  - um identificador. Para diferenciá-lo dos símbolos não-terminais, usa-se letras maiúsculas, como em `INTEIRO`, `IDENTIFICADOR` e `RETURN`
  - um caracter literal (como um caracter C constante). Isso é útil quando o *token* se refere a um único caracter mesmo (como no caso de *abre-parênteses*, *fecha-parênteses*, etc.)
  - uma string literal, como "<=" ou ":="

Descrição da gramática no Bison – regras

- Formato geral de uma regra:

  ```
  resultado:  componentes...;
  ```

  onde `resultado` é um não-terminal.

  Exemplo:

  ```
  exp:  exp '+' exp;
  ```

- Múltiplas regras para um mesmo resultado podem ser escritas separadamente ou podem ser unidas por meio do caracter '|', como mostrado a seguir:

  ```
  resultado:
    componentes-da-regra1...
  | componentes-da-regra2...
  ...
  ;
  ```

  Exemplo:

  ```
  exp:  exp '+' exp | NUMERO;
  ```

Valor Semântico

- Para ser útil, um programa tem que fazer algo mais do que simplesmente analisar a sintaxe da entrada, ele tem que gerar alguma saída
- Para gerar a saída, ele precisa definir a semântica (o significado) daquilo que ele está processando
- No caso de um compilador, a saída é um programa numa outra linguagem (p.ex., linguagem de montagem)
- No caso de uma calculadora, a saída é o resultado do cálculo

Símbolos – tipo e valor semântico

- Cada *token* reconhecido em um arquivo de entrada para um *parser*, além de ter um tipo (um símbolo não-terminal), tem também um valor semântico associado a ele
- Por exemplo, a expressão 1+2 possui 2 *tokens* de um mesmo tipo (por exemplo, INTEIRO ou NUMERO), mas que possuem valores semânticos diferentes
- Mesmo símbolos não-terminais possuem um valor semântico. Por exemplo, poderíamos atribuir como valor semântico para a expressão 1+2 o valor 3

Atribuindo valor semântico com o Bison

- Em um arquivo de entrada para o Bison, **cada regra pode ser seguida de uma ação que define o que deve ser feito quando a regra é detectada**.
  Por exemplo, a regra

  ```
  expr: expr '+' expr   { $$ = $1 + $3; }
  ```

  define que o valor semântico de `expr` quando essa regra é detectada é a soma das duas sub-expressões.
- Os tipos dos valores semânticos dos símbolos são definidos na seção de *Declarações do Bison*, na primeira parte do arquivo de entrada para o Bison
- Nessa mesma seção, também devem ser definidos os símbolos usados para a definição da gramática

Regras e suas ações

- Uma **ação** é um código em C delimitado por { }
- Uma ação pode ser colocada em qualquer posição da regra. A maioria das regras possuem apenas uma única ação, posicionada no final da regra (depois de todos os seus componentes)
- Ações definidas no meio de uma regra são complicadas e são usadas somente em contextos especiais
- Quando uma ação não é definida para uma regra, o Bison fornece uma ação padrão: `$$ = $1`, ou seja, o valor do primeiro símbolo na regra será o valor da regra inteira

Como criar um analisador sintático

- Especifique a gramática em um arquivo `.y` e, para cada regra gramatical, descreva a ação a ser tomada quando uma instância daquela regra é detectada
- Escreva um analisador léxico para fornecer os itens léxicos (tokens) para o Bison (pode ser manualmente ou via Flex)
- Escreva uma função (p.ex., `main`) para chamar a função `yyparse` gerada pelo Bison
- Escreva, opcionalmente, funções de relatos de erros

Como gerar o código, compilar e executar o analisador sintático

- Os arquivos com gramáticas do bison (`*.y`) são processados pelo bison e geram arquivos `*.tab.c`
- Os `*.tab.c` são compilados pelo `gcc` para gerar um executável, p.ex., `prog`
- O executável `prog` processa um arquivo de entrada (escrito na linguagem criada), gera a árvore sintática do programa e executa os comandos definidos pelo programador, associados à gramática

Bison + Flex – alguns exemplos

No diretório de exemplos desta aula na Paca, vocês encontrarão 3 exemplos [1] de calculadoras geradas com o auxílio do Flex e o Bison:

1. calculadora polonesa simples (diretório *calcpr*)
2. calculadora infixa
3. calculadora com variáveis e funções

As explicações sobre os exemplos estão no manual do Bison:
`http://www.gnu.org/software/bison/manual/html_node/Examples.html#Examples`

[1] Os exemplos estão listados em ordem crescente de dificuldade

Bibliografia e materiais recomendados

- Manual do Flex
  `http://flex.sourceforge.net/manual/`
- Manual do GNU Bison
  `http://www.gnu.org/software/bison/manual/index.html`
  em particular, vale a pena ler a seção que descreve os principais conceitos e a seção de exemplos:
  `http://www.gnu.org/software/bison/manual/html_node/Concepts.html#Concepts`
  `http://www.gnu.org/software/bison/manual/html_node/Examples.html#Examples`
- Texto que traça uma "ponte" entre o Flex e o Bison:
  `http://flex.sourceforge.net/manual/Bison-Bridge.html`
- Notas das aulas de MAC0211 de 2010, feitas pelo Prof. Kon
  `http://www.ime.usp.br/~kon/MAC211`

Cenas dos próximos capítulos...

- Sistemas de controle de versão (como o Git)