

## Compiladores

### Análise lexical (3) LEX

## Lembrando

- Define-se padrões para reconhecimento de tokens através de Expressões Regulares (ERs).
- ERs podem ser automaticamente transformadas em autômatos de estados finitos não determinísticos (AFNDs)
  - Algoritmo de Thompson
- AFNDs podem ser automaticamente tornados determinísticos
  - AFD
- Um AFD pode ser simulado simplesmente por um programa:
  - Usa-se uma tabela de transições;
  - A cada estado é associado uma rotina que efetua o procedimento adaptado.

## O que fazer quando se reconhece um token?

- **Atualizar a tabela de símbolos:**
  - Verificar se já existe o lexema na tabela;
    - Se sim, erro potencial!
    - Se não, deve-se inserir o lexema na tabela.
  - A tabela é inicializada com a lista de palavras/tokens chave da linguagem.
    - if, for, while, '=', class, const, extern, static...
- Retornar informação ao programa que originou a análise lexical:
  - Por exemplo: qual token foi reconhecido!
  - Dependendo do token: outras informações...

## Atributos

- Caso trivial: o token é um "espaço branco"
  - Nada para fazer... Só ler o próximo token.
- Caso simples: o token é um símbolo reservado ('if', por exemplo)
  - Basta retornar o token (por exemplo um número que o representa).
- Caso menos simples (1): o token é um identificador
  - Retorna-se o token 'identificador';
  - Em geral, deve se recuperar também o string associado!
- Caso menos simples (2): o token é um número
  - Retorna-se o token 'numero';
  - Em geral, deve se recuperar também o valor associado!
- Fala-se de **atributos** para designar o conjunto de informações a retornar, junto com o token, pelo analisador lexical.

## (F)LEX

- LEX é uma ferramenta Unix para gerar analisadores lexicais sob-demanda.
  - Lesk & Schmidt, Bell Laboratories, 1975.
- Versão free: FLEX (*Fast Lex*)
  - <http://www.gnu.org/software/flex/>
- Funcionamento:
  1. Especificação LEX (arquivo fonte .l)
    - Declarações, Padrões, regras para aplicar ações
  2. Compilação LEX
    - Obtenção de um programa C (lex.yy.c)
  3. Compilação do programa C
    - Obtenção de um analisador lexical

## Compilação típica com FLEX

- Edição do texto de especificação
  - No arquivo 'minhas\_regras.l'
- **flex minha\_regras.l**
  - O programa C resultado se encontra em 'lex.yy.c'
  - Contém a rotina 'yylex()' que efetua a análise lexical.
  - 'flex -o analisador.c minha\_regras.l' retorna o programa C em 'analisador.c'
- **gcc -o analisador lex.yy.c -lfl**
  - Compila e amarra o analisador com a biblioteca libfl.a
  - O executável se encontra em 'analisador'
  - Obs: deve ter um 'main' em algum lugar!

## Especificação (F)lex

### Definições

%%

### Regras

%%

### Subrotinas opcionais (código C)

- Lex define várias variáveis globais:
  - yytext e yyleng
    - Lexema corrente e seu tamanho
  - yyval
    - De tipo YYSTYPE, a partir de #defines
    - Depende do parser, você escolhe

7

## A seção de definições

- Opcional (pode ser vazia)
- Seu conteúdo será copiado no início do arquivo C gerado em saída
  - Pode conter diretivas do tipo #include que serão úteis ao analisador!
  - Em geral, quaisquer declarações C que serão úteis.
  - Por exemplo os tokens que irão ser reconhecidos (constantes retornadas pelo analisador);
  - Essas declaração devem vir entre %>{ e %}'
- Contém também a declaração de ERs nomeadas que podem ser usadas na seção de regras.

8

## Definição de ERs no Lex

- Segue quase a mesma sintaxe que foi apresentada para outras ferramentas do Unix.
- \*, +, ?, .
- Alternativa: |
- **[a-z]** é equivalente a a|b|c|...|z
  - [0-9]
  - [A-Z], [abcd]
  - [^a-|] : negação de [a-|]
- ^ ... \$ delimitam uma linha.
- Exemplo:  
-?([0-9+])((0-9)^.[0-9+])((eE)[-+]?[0-9+])
- Usa-se {xxx} para distinguir uma variável xxx da ER 'xxx'

9

## A seção das regras

- Consiste de linhas do tipo:  
ER { ação }
- A ER é especificada conforme a sintaxe de LEX, ou pode ser uma variável.
- '{ }' é a ação nula – não faz nada.
  - Retorna para o yylex() que analisará o próximo token!
  - Usado para detectar os "brancos".
- '{ printf("é isso a\n"); }' é a "ação" que consiste em imprimir uma mensagem na tela.

10

## A seção de procedimentos adicionais

- Opcional (pode ser vazia)
- Qualquer código C nessa seção será copiado diretamente no arquivo produzido pelo Lex.
  - Pode definir código para as ações complexas usadas na seção anterior.
- Por exemplo, pode conter um 'main'.
  - Vantagem: se obtém tudo o analisador com um arquivo fonte Lex só.
  - Inconveniente: se é preciso de muitos procedimentos internos, não é bom colocá-los todos no mesmo arquivo...

11

## Escolha das regras

- Quando há ambigüidade entre regras, o LEX escolhe a que provê o lexema maior.
- A ação default é copiar para a saída o texto de entrada.
- A regra 'r1/r2' reconhece r1 somente se r2 aparece após (aceitação condicional).
  - Quando se encontra o lexema r1r2, o r2 é descartado e colocado de volta no buffer de entrada. r1 é reconhecido.

12

## Conclusão

- O (F)Lex é a ferramenta preferencial para gerar a primeira fase de um compilador.
- Na verdade, 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 yylex().
- Basta definir suas ERs (i.e., a linguagem) para obter automaticamente um reconhecedor e poder tomar ações em função dos tokens reconhecidos.

13

## Semana que vem...

### Exercício

- Implementar um analisador léxico para uma calculadora simples, capaz de realizar as quatro operações aritméticas básicas.
  - Entrada válida: número operador número.
  - Não é necessário tratamento de erros: descarte o que não for válido.
  - Objetivo do trabalho:
    - Explorar as potencialidades do (f)lex e familiarização com este ambiente.

14