Processamento de Linguagens

2ª Ficha para as Aulas Práticas

Objectivos

Em complemento à ficha 1, esta ficha prática contém exercícios para serem resolvidos nas aulas teórico-práticas com vista a sedimentar os conhecimentos relativos a:

- uso de Expressões Regulares para definir (gerar) Linguagens Regulares;
- uso de Expressões Regulares para desenvolver programas eficientes, baseados em algoritmos standard guiados por Autómatos Finitos Deterministas, para reconhecer Linguagens Regulares;
- uso de Autómatos Deterministas Reactivos, para processar Linguagens Regulares, isto é para desencadear Acções específicas ao reconhecer frases que derivam de Padrões (definidos com base em ERs) —princípio da Programação baseada em regras *Condição-Reacção*;
- geração automática de programas a partir de especificações formais;
- uso da ferramenta Flex, disponível em ambiente Linux, para geração automática de *Analisadores Léxicos*.

Complementarmente, servem os exercícios desta ficha para introduzir as Gramáticas Independentes de Contexto (GIC) como formalismo para definição de linguagens de programação.

1 Desenvolvimento de Analisadores Léxicos

No contexto do desenvolvimento de Compiladores, ou mais genericamente de Processadores de Linguagens, o primeiro nível, ou camada a implementar, é a **análise léxica** que tem por missão *ler o texto fonte*, escrito numa determinada linguagem, *e converter todas as palavras correctas em* ***símbolos terminais*** *dessa linguagem.*
Por questões pragmáticas (de eficiência e de espaço em memória), o analisador léxico não lê o texto todo de uma só vez porque isso obrigaria a criar uma estrutura interna ou um ficheiro para armazenar os códigos dos símbolos terminais que vão sendo reconhecidos no texto; em vez disso, o analisador léxico, cada vez que encontra e identifica um símbolo terminal retorna de imediato o seu código ao programa/função que o invocou.

Com esse fim em vista, propõe-se para esta aula o recurso à ferramenta Flex para gerar o **Analisador Léxico** (AL) para uma dada linguagem especificada por uma GIC (dada ou a definir na aula), a partir da descrição dos símbolos terminais dessa gramática e da respectiva associação aos códigos internos.
Para que se veja o resultado, o programa principal deve invocar sucessivamente o dito Analisador Léxico até consumir todo o texto fonte, isto é, até ser encontrado o fim-de-ficheiro (caso em que o código retornado é 0), imprimindo sucessivamente o código de cada símbolo terminal reconhecido.

1.1 Linguagem SQL

Como sabe a linguagem standard para interrogação e gestão de Bases de Dados é a SQL. Esta linguagem, criada para interrogar BDs, permite escrever instruções da forma:

```
SELECT * FROM table WHERE ((chv = 1) AND (tipo="AAA"))
ORDER BY nome GROUP BY regiao
```

No contexto desta aula—ignorando todos os outros comandos que podem ser escritos para inserir, remover e actualizar bases de dados, tabelas e registos—pretende-se que desenvolva um AL para reconhecer todos os símbolos terminais das frases interrogação da linguagem SQL acima exemplificada e devolver os respectivos códigos.
Melhore o seu AL suportando cada *palavra-chave*, ou *palavra-reservada*, da linguagem em *maiúsculas* ou *minúsculas* e permitindo a inserção de *linhas de comentário* no meio de uma frase válida.

1.2 Linguagem genérica de Listas

No contexto desta aula pretende-se que desenvolva um AL para reconhecer todos os símbolos terminais da linguagem para escrita de listas de palavras e números introduzida nas aulas teóricas e definida pela GIC a seguir relembrada:

```
1: Frase --> INICIO Lista FIM
2: Lista --> Elem
3:         |  Elem "," Cauda
4: Cauda --> Lista
5: Elem  --> pal
6:         |  num
```

Para poder ligar este analisador léxico ao Parser RD também apresentado nas aulas teóricas (e acessível em código C a partir do respectivo sumário) use os códigos seguintes:

```
#define EOF    0
#define Frase  1
#define Lista  2
#define Cauda  3
#define Elem   4
#define INIC   5
#define FIM    6
#define Virg   7
#define pal    8
#define num    9
```

1.3 Linguagem Lisp

Adapte o Analisador Léxico anterior para reconhecer todos os símbolos terminais da linguagem Lisp, também estudada nas aulas teóricas (ver sumários respectivos) e definida pela GIC a seguir relembrada:

```
1: Lisp --> SExp
2: SExp --> pal
3:        |  num
4:        | "(" SExpLst ")"
5: SExpLst --> SExp SExpLst
6:           |  &
```

Escolha o código dos símbolos a seu gosto e adapte o RD da linguagem de Listas de modo a obter um parser recursivo-descendente, em C, para a linguagem Lisp.

1.4 Documento anotado em XML

Como sabe um Documento XML é um texto vulgar semeado de anotações, ou marcas, que são identificadores especiais (designados por *elementos XML*) intercalados entre os caracteres "<" e ">". Num documento XML bem formado, a cada *marca de abertura* corresponderá uma *marca de fecho*, que tem o mesmo identificador, mas que começa por "</" terminando na mesma em ">".
Dentro de cada *marca de abertura*, além do identificador do elemento, ainda podem aparecer triplos formados por um outro identificador (de atributo), pelo sinal "=" e pelo respectivo valor que é qualquer texto entre aspas.
Cada fragmento do documento (texto livre) entre marcas deve ser considerado em bloco como sendo o símbolo PCDATA.
Desenvolva então um AL que receba um documento XML e devolva todos os símbolos terminais encontrados, a seguir resumidos: "<", ">", "</", "=", identificador (qualquer palavra formada por letras), valor, PCDATA.

1.5 Máquina de Venda de Chocolates

Considere uma situação em que se pretende simular o funcionamento de uma máquina de vender chocolates. Dado o stock no início do dia (nome, preço e quantidade de cada produto disponível), a quantia inicial de trocos e os registos das vendas diárias (nome do chocolate escolhido e a quantia introduzida), o objectivo é calcular *a evolução do stock ao longo do dia e o dinheiro acumulado.* A animação pretendida deverá mostrar através de desenhos o estado da máquina (stock existente e dinheiro ganho) após cada movimento. No contexto desta aula o que se pretende é que:

- defina uma linguagem para descrever o estado inicial da máquina e os registos de vendas efectuadas durante o dia;
- desenvolva um AL para reconhecer todos os símbolos terminais dessa linguagem e devolver os respectivos códigos.
  Melhore o seu AL suportando cada *palavra-chave*, ou *palavra-reservada*, da linguagem em *maiúsculas* ou *minúsculas* e permitindo a inserção de *linhas de comentário* no meio de uma frase válida.

1.6 Anuário dos Medicamentos brancos

Considere agora uma outra situação em que, para auxiliar o Instituto Farmacêutico do Ministério da Saúde na gestão do novo lote de medicamentos brancos, se pretende criar um sistema de consulta a esses medicamentos acessível a qualquer farmácia via um browser HTML. Esse sistema deve mostrar a informação agrupada por: classe de medicamentos no Symposium Terapêutico (uma página por classe, com os medicamentos ordenados alfabeticamente); ou por fabricante (uma página única, com os medicamentos agrupados por fabricante).
Sobre cada medicamento é fornecida a seguinte documentação: nome, código, classe, composição química, preço recomendado, fabricantes disponíveis e lista de medicamentos de marca equivalentes (respectivo nome e fabricante). No contexto desta aula o que se pretende é que:

- defina uma linguagem para descrever a informação envolvida no lote de medicamentos a considerar (essa linguagem terá que permitir definir inicialmente o ano a que o Symposium Terapêutico diz respeito e a lista das classes de medicamentos);
- desenvolva um AL para reconhecer todos os símbolos terminais dessa linguagem e devolver os respectivos códigos.
  Melhore o seu AL suportando cada *palavra-chave*, ou *palavra-reservada*, da linguagem em *maiúsculas* ou *minúsculas* e permitindo a inserção de *linhas de comentário* no meio de uma frase válida.