Universidade Federal de São Carlos – Departamento de Computação
Construção de Compiladores e Construção de Compiladores 1
Profa. Helena Caseli

Primeira Lista de Exercícios – Introdução e Análise Léxica

1) O que é um compilador?
R. Um compilador é um programa que lê um programa em uma linguagem-fonte e o traduz em um programa em uma linguagem-alvo (objeto).

2) Quais são as 6 etapas em uma estrutura padrão de tradução realizada por um compilador? Descreva brevemente o que acontece em cada uma delas.
R. Um compilador típico divide a compilação em 6 etapas em que cada uma toma como entrada a saída produzida pela etapa anterior. A primeira etapa, análise léxica, toma como entrada o programa fonte a ser compilado. A última etapa, geração de código, produz um arquivo com o código gerado a partir do programa fonte.

1. *Análise léxica – o fluxo de caracteres que constitui o programa é lido da esquerda para a direita e esses caracteres são agrupados em tokens*
2. *Análise sintática – os caracteres ou tokens são agrupados hierarquicamente em coleções aninhadas com significado coletivo*
3. *Análise semântica – certas verificações são realizadas a fim de se assegurar que os componentes de um programa se combinam de forma significativa*
4. *Geração de código intermediário – alguns compiladores geram uma interpretação intermediária explícita para o programa fonte (ex: código de 3 endereços)*
5. *Otimização de código intermediário – tentativa de melhorar o código intermediário de tal forma que resulte em um código de máquina mais rápido em tempo de execução*
6. *Geração/otimização de código-alvo – as localizações de memória são selecionadas para cada uma das variáveis e as instruções intermediárias são traduzidas numa sequência de instruções de máquina*

3) O que vem a ser "passada" e qual a vantagem e a desvantagem de um compilador que realiza várias passadas em relação a um compilador de apenas uma passada?
R. Passada é cada uma das vezes que o programa-fonte é processado antes de gerar o código final. Geralmente uma passada engloba diversas etapas. Enquanto em um compilador de uma passada (ex: Pascal, C) todas as etapas ocorrem uma única vez, em um de várias passadas as etapas são divididas por exemplo em:

- *Uma passada para análise léxica + análise sintática*
- *Uma passada para análise semântica + otimização intermediária*
- *Uma passada para geração de código + otimização alvo*

Vantagem de um compilador de várias passadas em relação ao de uma passada apenas: o código-alvo é mais eficiente
Desvantagem de um compilador de várias passadas em relação ao de uma passada apenas: a compilação é menos eficiente

4) O que é um interpretador e quais suas diferenças em relação a um compilador?
R. Um interpretador é um programa que traduz programas, assim como um compilador, porém sem mapeá-lo para linguagem de máquina. Assim, o interpretador executa o programa fonte imediatamente ao invés de gerar um código-objeto para ser executado após a tradução.
COMPILADOR X INTERPRETADOR

- *O interpretador dá maior autonomia e controle ao código interpretado (confere acessos a*

memória/hardware) e facilita a criação de depuradores já que a interpretação está sob o controle do int. e não do hardware

- *Interpretadores são mais simples de serem construídos por vários motivos: traduzem para pseudocódigo, não precisam de editores de ligação (linkers) e o sistema de execução do programa está presente no próprio interpretador (em uma ling. de alto nível o de um compilador é pelo menos em parte codificado em assembly ou código de máquina).*
- *O programa interpretado inicia a execução mais rapidamente*
- *Compiladores produzem código que é 10-20 vezes mais rápido que um código interpretado "equivalente"*

5) Qual é a função da etapa de análise léxica no processo de tradução de um compilador?

R. A análise léxica é a etapa que lê o programa fonte como uma sequência de caracteres e os separa em tokens:

- *palavras reservadas: if, while etc.*
- *identificadores*
- *símbolos reservados: +, *, >=, <> etc.*

6) Identifique todos os *tokens* que compõem os programas seguintes informando, para cada um, a cadeia (lexema) correspondente e sua classe (identificador, palavra ou símbolo reservado, número etc.).

a) Pascal

```
function max(i, j: integer): integer;
{ retorna o maior dos inteiros entre i e j}
begin
  if i > j then max := i
  else max := j
end;
```

R.

Cadeia	*Classe*
function	*Palavra reservada*
max	*Identificador*
(	*Símbolo reservado, (*
i	*Identificador*
,	*Símbolo reservado, ,*
j	*Identificador*
:	*Símbolo reservado, :*
integer	*Palavra reservada*
)	*Símbolo reservado,)*
:	*Símbolo reservado, :*
integer	*Palavra reservada*
;	*Símbolo reservado, ;*
begin	*Palavra reservada*
if	*Palavra reservada*
i	*Identificador*

>	*Símbolo reservado, >*
j	*Identificador*
then	*Palavra reservada*
max	*Identificador*
:=	*Símbolo reservado, :=*
i	*Identificador*
else	*Palavra reservada*
max	*Identificador*
:=	*Símbolo reservado, :=*
j	*Identificador*
end	*Palavra reservada*
;	*Símbolo reservado, ;*

b) C

```
int max(i, j) int i, j;
/* retorna o maior dos inteiros entre i e j */
{
  return i > j ? i : j
}
```

R.

Cadeia	*Classe*
int	*Palavra reservada*
max	*Identificador*
(	*Símbolo reservado, (*
i	*Identificador*
,	*Símbolo reservado, ,*
j	*Identificador*
)	*Símbolo reservado,)*
int	*Palavra reservada*
i	*Identificador*
,	*Símbolo reservado, ,*
j	*Identificador*
;	*Símbolo reservado, ;*
{	*Símbolo reservado, {*
return	*Palavra reservada*
i	*Identificador*
>	*Símbolo reservado, >*
j	*Identificador*

?	*Símbolo reservado, ?*
i	*Identificador*
:	*Símbolo reservado, :*
j	*Identificador*
}	*Símbolo reservado, }*

c) Em qual dos dois programas apresentados nas letras acima (a e b) foram identificados mais *tokens*?
R. No a). No a) foram identificados 27 tokens enquanto que em b), apenas 22.

7) Se a análise léxica é feita sob o comando da análise sintática, então quais são os motivos para se separar conceitualmente a análise léxica da sintática? Explique cada um dos motivos.
R. Motivos para dividir (conceitualmente) a análise do programa em análise léxica X análise sintática:

- *Simplificação e modularização do projeto do compilador*
- *Os tokens podem ser descritos usando notação simples (como Expressões Regulares) enquanto a estrutura sintática de comandos e expressões das linguagens de programação requer uma notação mais expressiva (como Gramática Livre de Contexto)*
- *Os reconhecedores construídos a partir da descrição dos tokens são mais eficientes e compactos do que os reconhecedores construídos a partir das gramáticas livres de contexto*
- *A portabilidade do compilador é realçada já que as peculiaridades do alfabeto de entrada e outras anomalias podem ser restringidas ao analisador léxico*

8) Escreva expressões regulares para os conjuntos de caracteres a seguir ou se não for possível escrever uma expressão regular para um determinado conjunto de caracteres, justifique.

a) Cadeias de letras maiúsculas começando e terminando com *a* (minúsculo).
*R. a[A-Z]*a*

b) Cadeias de dígitos que representam números pares.
R. [0-9](0|2|4|6|8)*

c) Cadeias de 0s e 1s com um número par de 0s.
*R. (1*01*0)*1**
*A expressão de dentro da repetição externa à esquerda, (1*01*0)*, gera cadeias terminando com 0 que contêm exatamente dois 0s (qualquer número de 1s pode aparecer antes ou entre os dois 0s). A repetição dessas cadeias resulta em todas as cadeias terminando com 0 cujo número de 0s é um mútiplo de 2, ou seja, par. O acréscimo da repetição 1* no final fornece o resultado desejado.*

d) Cadeias de 0s e 1s nas quais os 0s ocorrem em pares (um 0 seguido de outro 0).
R. (1(00)*)**

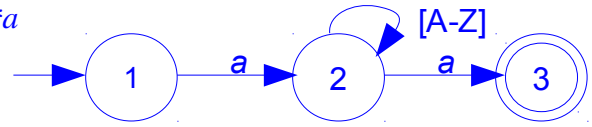
e) Cadeias de 0s e 1s compostas por um único 1 rodeado pelo mesmo número de 0s à esquerda e à direita.
R. 0^n10^n com n diferente de 0
*Esse conjunto não pode ser descrito por uma expressão regular. O motivo é que a única operação de repetição que temos é a operação de fecho *, a qual permite qualquer número de repetições e não garante que o número de 0s e 1s será o mesmo. Expressamos isso dizendo que "expressões regulares não podem contar".*

f) Cadeias de dígitos tais que todos os dígitos ímpares, se ocorrerem, ocorrem antes de todos os dígitos pares (se ocorrerem).
R. (1|3|5|7|9)(0|2|4|6|8)**
OBS.: 0 é par

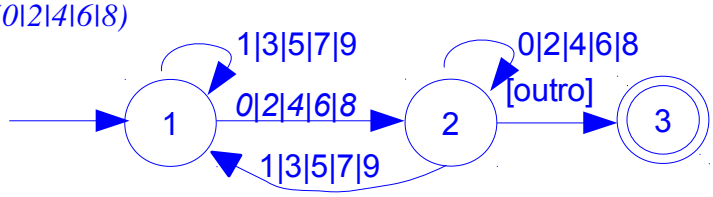
9) Construa os autômatos correspondentes para as expressões regulares descritas no exercício 8. Assuma que os autômatos construídos serão implementados em um sistema de varredura (análise léxica) e que, portanto, a regra de reconhecimento da maior cadeia possível deve ser seguida (cuidado ao definir o estado final).

R.

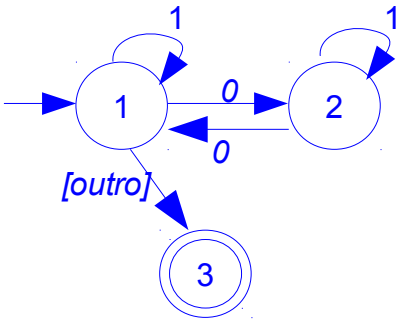
*a) a[A-Z]*a*



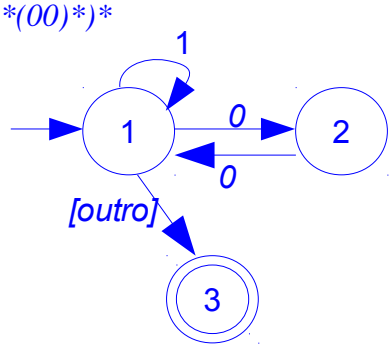
b) [0-9](0|2|4|6|8)*



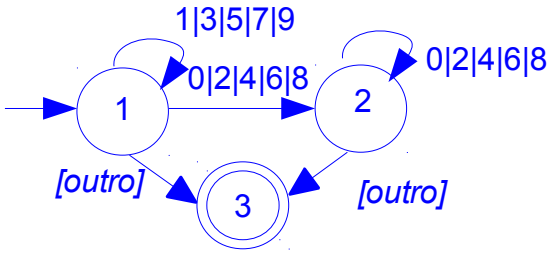
*c) (1*01*0)*1**



d) (1(00)*)**



f) (1|3|5|7|9)(0|2|4|6|8)**



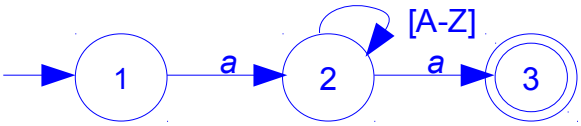
10) Escreva o pseudocódigo (algoritmo) para o reconhecimento de cada conjunto de cadeias do exercício 8, ou seja, faça o mapeamento dos autômatos descritos no exercício 9 em código usando ifs ou cases como apresentado em aula.

R.

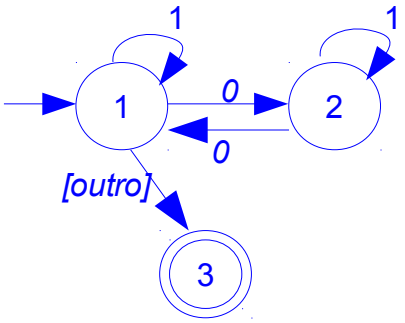
a) usando ifs

{estado 1}

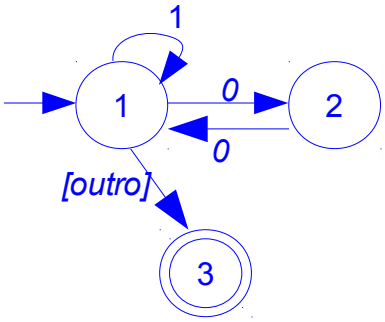
```
if próximo caractere for "a" then
   avance entrada; {estado 2}
   while próximo caractere não for "a" do
     avance entrada; {fica no estado 2}
  end while;
  avance entrada;
  aceitação; {estado 3}
else {outro processamento}
end if;
```




```
c) usando cases
estado := 1;
while estado = 1 ou 2 do
   case estado of
   1: case caractere de entrada of
      "0": avance entrada;
           estado := 2;
      "1": avance entrada;
      else estado := 3;
      end case;
   2: case caractere de entrada of
      "0": avance entrada;
           estado := 1;
      "1": avance entrada;
      else estado := erro;
      end case;
   end case;
end while;
if estado = 3 then aceitação else erro;
```




```
d) usando cases
estado := 1;
while estado = 1 ou 2 do
   case estado of
   1: case caractere de entrada of
      "0": avance entrada;
           estado := 2;
      "1": avance entrada;
      else estado := 3;
      end case;
   2: case caractere de entrada of
      "0": avance entrada;
           estado := 1;
      else estado := erro;
      end case;
   end case;
end while;
if estado = 3 then aceitação else erro;
```



f) usando cases

```
estado := 1;
while estado = 1 ou 2 do
  case estado of
  1: case caractere de entrada of
     "0" ou "2" ou "4" ou "6" ou "8": avance entrada;
           estado := 2;
     "1" ou "3" ou "5" ou "7" ou "9": avance entrada;
     else estado := 3;
     end case;
  2: case caractere de entrada of
     "0" ou "2" ou "4" ou "6" ou "8": avance entrada;
     else estado := 3;
     end case;
  end case;
end while;
if estado = 3 then aceitação else erro;
```