1ª lista de exercícios – Compiladores - Respostas

1 - Descreva o que são os modelos de análise e síntese relacionados ao processo de compilação? Cite as fases que compõe cada um dos modelos.
Análise: analisar a corretude do código fonte em relação a especificação da linguagem de programação. Quebrar o programa fonte em diversas construções criando uma representação intermediária. Composta por: Análse Léxica, Sintática e Semântica.

Síntese: gerar o programa destino a partir de uma representação intermediária. Composta por: gerador de código assembly, otimizadores de código e gerador de código de máquina.

2 - Cite e descreva 2 processos que são de utilização facultativa no projeto de um compilador?

Otimizador de código assembly e otimizador de código de máquina.

3 - Descreva de forma sucinta o que significam os termos abaixo:
- análise léxica: leitura de um fluxo de caracteres do código fonte em linguagem de alto nível mapeando os mesmos em um conjunto de tokens.
- assembly: linguagem de programação de baixo nível
- código relativo: é o código produzido pelo montador, é o código de máquina equivalente às instruções assembly geradas pelo compilador
- código absoluto: é a junção dos códigos relativos, resultado da linkedição, associada às adaptações necessárias para carregar este código para a memória.
- lexeme: conjunto de símbolos terminais de um alfabeto
- token: regra que define um conjunto de lexemas com as mesmas características
- parser: processo responsável pela análise sintática, procura verificar se o conjunto de tokens produzido pelo analizador léxico pode ser gerado a partir das regras da gramática que define a linguagem que esta sendo avaliada pelo compilador
- montador: efetua a tradução do programa assembly em um código de máquina relativo
- loader: realiza a tarefa de carregar o programa na memória, altera os endereços relativos, carregando o programa e os dados na memória, o novo código é denominado absoluto
- tabela de símbolos: estrutura responsável por manter as informações sobre lexemas, tokens e seus respectivos atributos

4 - Cite 2 ou mais utilizações para as técnicas aprendidas na disciplina compiladores.
Formatadores de texto
entrada: seqüência de caracteres
saída: texto formatado (parágrafos, figuras)

Interpretadores de Consultas
entrada: predicados contendo operadores lógicos e relacionais
saída: comandos para pesquisar a base de dados procurando registros que atendam ao predicado

5 - Defina expressões regulares para:
1. Strings sobre {a,b} contendo aa ou bb
2. Strings sobre {a,b} contendo aa e bb
3. Strings sobre {a,b} contendo exatamente dois b's
4. Strings sobre {a,b} contendo pelo menos dois b's
5. Strings sobre {a,b} com número par de b's
6. Strings sobre {a,b} que começam com ba, contém aa e terminam com ab
7. Strings sobre {a,b} que não terminam com aaa
8. Strings sobre {a,b,c} que não contém bc
9. Strings sobre {a,b} que não contém aa

7.1) (a | b)* (aa | bb) (a | b)*
7.2) ((a | b)* aa (a | b)* bb (a | b)*) | ((a | b)* bb (a | b)* aa (a | b)*)
7.3) a* b a* b a*
7.4) (a | b)* b (a | b)* b (a | b)*
7.5) (a | (a* b a* b a*))*
7.6) ba (((a | b)* aa (a | b)*) | a* | ab* | b*a)* ab
7.7) ((a | b)* (b | ba | baa)) | a | aa | λ
7.8) c* (b | (ac*))*
7.9) (b | (a b+))* (a | λ)

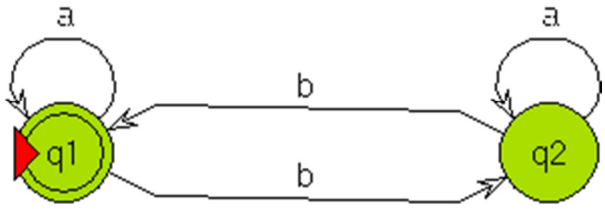
6 – Escreva gramáticas livres de contexto para gerar as linguagens definidas nos itens 2, 3 e 7 da questão anterior.

S -> XaaXbbX | XbbXaaX
X -> aX | bX | λ

S -> XbXbX
X -> aX | λ

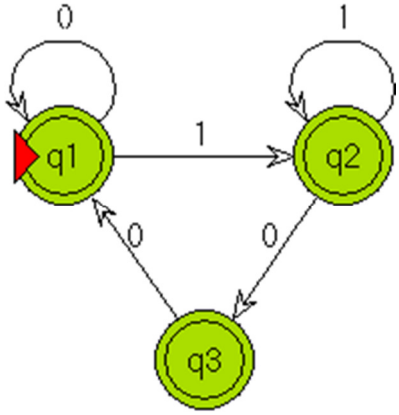
S -> Xb | Xba | Xbaa | a | aa | λ
X -> aX | bX | λ

7 - Construa um autômato finito determinístico e um programa em C, para reconhecer as strings da linguagem do item 5.5.




```
void analisador(String a[]) {
     int p = 0;
     int state = 1;
     while (p < length(a)) {
       switch(state) {
         case 1:
           switch(a[p]) {
             case 'a':
               state = 1;  break;
             case 'b':
               state = 2;  break;
           }
           break;
        case 2:
           switch(a[p]) {
             case 'a':
               state = 2; break;
             case 'b':
               state = 1; break;
           }
           break;
       }
       p++;
     }
     if (state==1)
       System.out.println(”Aceita");
     else
       System.out.println("Nao aceita");
}
```

8 - Construa um autômato finito determinístico e um programa em C, que reconheça sentenças em {0,1}*, as quais não contenham sequências do tipo 101.

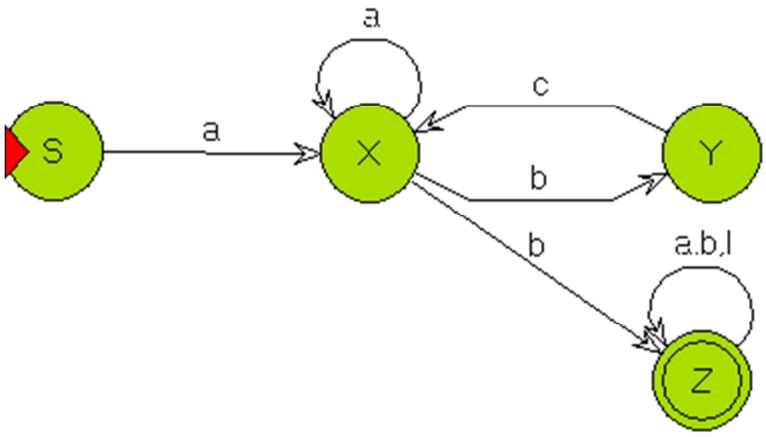



```
void analisador(String a[]) {
     int p = 0;
     int state = 1;
     while (p < length(a)) {
       switch(state) {
         case 1:
           switch(a[p]) {
             case 0:
```

```
            state = 1;  break;
          case 1:
            state = 2;  break;
        }
        break;
      case 2:
        switch(a[p]) {
          case 0:
            state = 3; break;
          case 1:
            state = 2; break;
        }
        break;
      case 3:
        switch(a[p]) {
          case 0:
            state = 1; break;
          case 1:
            state = 4; break;    \\ estado de erro
        }
        break;
    }
    p++;
  }
  if ( (state == 1) || (state == 2) || (state == 3) )
    System.out.println(”Aceita");
  else
    System.out.println("Nao aceita");
}
```

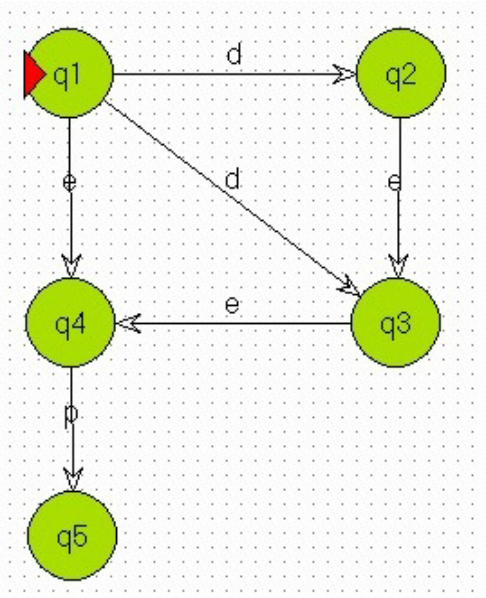
9 - Construa um autômato finito e a gramática regular equivalente à seguinte expressão regular: a (a | bc)* b (a | b)*

S -> aX
X -> aX | bY | bZ
Y -> cX
Z -> aZ | bZ | λ



10 - A partir do alfabeto {[0-9] , . }. Construa uma definição regular e um autômato para reconhecer um número IP “extenso” x.x.x.x sendo que “x” pode ser qualquer número entre 0 e 999. Ex: 127.0.0.0 , 229.120.10.1, 999.128.888.128, etc
Sugestão de definições regulares auxiliares:
d->[1-9]
e->[0-9]

(d e e | d e | e) . (d e e | d e | e) . (d e e | d e | e) . (d e e | d e | e)



11 – Construa uma definição regular para a seguinte linguagem: valores monetários em Reais. Possuem exatamente duas casas decimais depois da vírgula e usam ponto como separador de milhar.
Exemplos: { R$ 2,35 ; R$ 1.546,98 ; R$ 1,00 ; R$ 10.000.000.000,00 }

dig -> [0-9]
digs0 -> [1-9]

R $ digs0 dig? dig? (. dig dig dig)* , dig dig | R $ 0 , dig dig

12 - Construa uma definição regular para produzir a seguinte linguagem: todas as datas dos anos 2017, 2018 e 2019 no formato DD/MM/AAAA. Obs: nenhum destes anos é bissexto. Ex: 28/02/2017, 31/12/2018, 06/09/2019, etc

Meslongo -> 01 | 03 | 05 | 07 | 08 | 10 | 12
Mescurto -> 04 | 06 | 09 | 11
Ano -> [2017-2019]

Datas -> [1-31] / Meslongo / Ano | [1-30] / Mescurto / Ano | [1-28] / 02 / Ano