# Lista de exercícios para a avaliação da 1ª unidade

1. Escreva códigos na linguagem C que inicializam uma matriz A quadrada de 100000 linhas em um espaço de endereços contíguo utilizando os 4 tipos de código abaixo. Compile os códigos com seu compilador favorito e temporize a execução dos programas resultantes com o comando `time` do Linux. Observe se há diferenças de desempenho e explique a importância das suas observações no projeto de compiladores.

   a.
   ```
   for (i=0; i<n; i++)
       for (j=0; j<n; j++)
           A[i][j] = 0;
   ```

   b.
   ```
   for (i=0; i<n; i++)
       for (j=0; j<n; j++)
           A[j][i] = 0;
   ```

   c.
   ```
   p = &A[0][0];
   t = n * n;
   for (i=0; i<t; i++)
       *p++ = 0;
   ```

   d.
   ```
   bzero((void*) &A[0][0],(size_t) n*n*sizeof(int))
   ```

2. Exercício 3 do capítulo 1 do livro Engineering a Compiler.

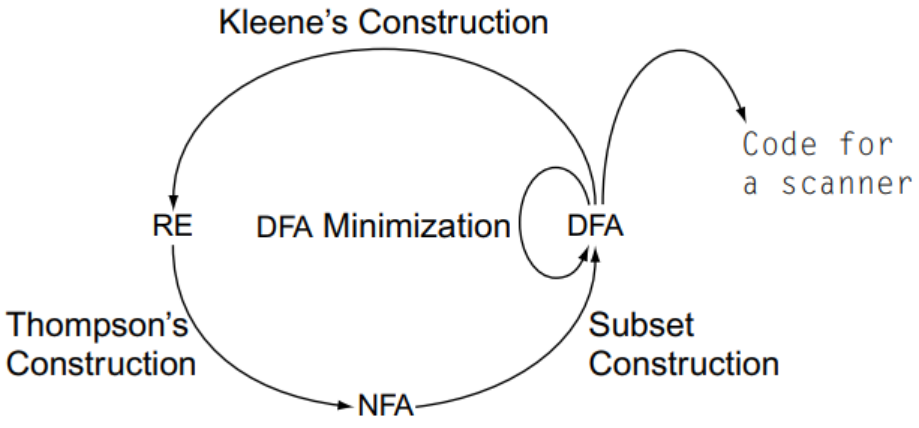
3. Quais são as principais etapas no processo de compilação? Elabore a sua resposta com um diagrama de blocos. Explique qual a função de cada etapa.

4. Dado o seguinte código reconhecedor de nome de registradores abaixo e as tabelas ACTION e STATE sobrepostas abaixo do código, explique o processo de reconhecer ou não as seguintes palavras:
   a. r100
   b. r3
   c. r15
   d. r07

```
Char <- next character
State <- s0
while (Char ≠ EOF)
    Next   <- STATE(State,Char)
    Act <- ACTION(State,Char)
    perform action Act
    State <- Next
    Char <- next character
if (State is a final state )
    then report success
    else  report failure
```

| State / Action | r | 0,1 | 2 | 3 | 4,5,6 7,8,9 | other |
|---|---|---|---|---|---|---|
| 0 | 1 *start* | e | e | e | e | e |
| 1 | e | 2 *add* | 2 *add* | 5 *add* | 4 *add* | e |
| 2 | e | 3 *add* | 3 *add* | 3 *add* | 3 *add* | e *exit* |
| 3,4 | e | e | e | e | e | e *exit* |
| 5 | e | 6 *add* | e | e | e | e *exit* |
| 6 | e | e | e | e | e | x *exit* |
| e | e | e | e | e | e | e |

5. Discorra sobre a figura abaixo enfatizando o papel e a importância de cada uma das construções mencionadas para o projeto de um *scanner*.




6. Apresente o algoritmo de Construção de Subconjuntos para transformação de um NFA em um DFA e explique seu funcionamento com um exemplo.

7. Explique, com um exemplo, como funciona a abordagem de minimização de um AFD.

8. Apresente o algoritmo de *scanner* controlado por tabela e codificado diretamente para o reconhecimento de um fluxo de palavras e explique seu funcionamento com um exemplo.

9. Explique as duas principais causas de ambiguidade em gramáticas livre de contexto. Por que é importante para um *parser* que a gramática livre de contexto empregada não seja ambígua? Quais as formas de resolver esse problema?

10. Explique o que pode acontecer com um *parser top-down* se a gramática utilizada tiver recursão à esquerda. Como isso pode ser resolvido?

11. Para que servem os conjuntos FIRST(β), FOLLOW(A), e $FIRST^{+}$(A->β) no contexto de *parsers top-down*?

12. Dada a gramática da expressão clássica na sua forma recursiva à esquerda abaixo, apresente um parse *top-down* para a expressão x * 5 + y para uma derivação à direita.

| | | | |
|---|---|---|---|
| 0 | *Goal* | → | *Expr* |
| 1 | *Expr* | → | *Expr* + *Term* |
| 2 | | \| | *Expr* - *Term* |
| 3 | | \| | *Term* |
| 4 | *Term* | → | *Term* * *Factor* |
| 5 | | \| | *Term* / *Factor* |
| 6 | | \| | *Factor* |
| 7 | *Factor* | → | number |
| 8 | | \| | id |
| 9 | | \| | ( *Expr* ) |

13. Explique o algoritmo do parsing de descida recursiva (Recursive Descent Parsing) e apresente um exemplo.

14. Dada a gramática da expressão clássica na sua forma recursiva à esquerda abaixo, apresente um parse *bottom-up* para a expressão x * 5 + y com os seus respectivos *handles* para uma derivação à direita.

| | | | |
|---|---|---|---|
| 0 | *Goal* | → | *Expr* |
| 1 | *Expr* | → | *Expr* + *Term* |
| 2 | | \| | *Expr* - *Term* |
| 3 | | \| | *Term* |
| 4 | *Term* | → | *Term* * *Factor* |
| 5 | | \| | *Term* / *Factor* |
| 6 | | \| | *Factor* |
| 7 | *Factor* | → | number |
| 8 | | \| | id |
| 9 | | \| | ( *Expr* ) |

15. Dada a seguinte gramática:

| | | | |
|---|---|---|---|
| 0 | *Goal* | → | *List* |
| 1 | *List* | → | *List Pair* |
| 2 | | \| | *Pair* |
| 4 | *Pair* | → | *(Pair)* |
| 5 | | \| | *()* |

Apresente um parse bottom-up para:

a) “()(())”
b) “)(“

**Obs**: É obrigatório apresentar as tabelas: ACTION e GOTO para a gramática assim como as pilhas de execução para os itens requisitados.