Matemática Discreta
Revisão
T. Praciano-Pereira
alun@:

Lista numero 06
tarcisio.praciano@gmail.com
Dep. de Computação

11 de junho de 2013 — **Univ. Estadual Vale do Acaraú**

LaTeX — sis. op. Debian/Gnu/Linux

www.matem-discreta.sobralmatematica.org/

Se entregar em papel, por favor, prenda esta *folha de rosto* na sua solução desta lista, deixando-a em branco. Ela será usada na correção.

Esta lista de exercícios contém 8 questões e sua nota será multiplicada por $\frac{10}{8}$.

Exercícios 1 *Revisão*

1. *LISP Considere o programa* `(pol x lista soma)` *que se encontra no arquivo meu_lisp.lsp na página da discplina*

   (a) *(V)[ ](F)[ ] O parâmetro* `lista` *sempre recebe um número.*

   (b) *(V)[ ](F)[ ] O parâmetro* `x` *sempre recebe um número.*

   (c) *(V)[ ](F)[ ] Ao executar a função o parâmetro* `soma` *recebe a lista de coeficientes.*

   (d) *(V)[ ](F)[* `progn` *marca um bloco de comandos numa função LISP. Todas as funções neste bloco serão executadas.*

   (e) *(V)[ ](F)[ ] A função* `(pol)` *tem três* `if` *encaixados (um é a parte "*`else`*" do anterior).*

2. *LISP*

   Considere o programa `(pol x lista soma)` *que se encontra no arquivo meu_lisp.lsp na página da discplina.*

   (a) *(V)[ ](F)[ ] A função* `mapcar` *serve para chamar outras funções.*

   (b) *(V)[ ](F)[ ] A função* `mapcar` *tem como um dos seus parâmetros uma função.*

   (c) *(V)[ ](F)[ ] Suponha que* `polseg` *esteja definida e seja o polinômio do segundo grau* x^2.
   `(mapcar polseg '(1 2 3 4 5))`
   resulta na lista `(1 4 9 16 25)`.

   (d) *(V)[ ](F)[ ] A expressão* `lambda`
   `(lambda (z) (* z z))`
   representa x^2.

   (e) *(V)[ ](F)[ ] A expressão* `lambda`
   `(lambda (z) (* 3 z))`
   representa $3x$.

3. *LISP*

   Considere no arquivo `meu_lisp.lsp` *que se encontra no link "programas" da página da discplina, a função* `(fibonacci)`.

   (a) *(V)[ ](F)[* `(fibonacci)` *é uma função recursiva.*

   (b) *(V)[ ](F)[ ]* `(fibonacci 0)` *está definida e vale 1*

   (c) *(V)[ ](F)[ ]* `(fibonacci 1)` *está definida e vale 1*

   (d) *(V)[ ](F)[ ]* `(fibonacci)` *sempre chama dois valores consecutivos a partir de 3.*

   (e) *(V)[ ](F)[ ]* `(fibonacci)` *sempre chama dois valores consecutivos a partir de 2.*

4. *LISP Considere no arquivo* `meu_lisp.lsp` *que se encontra no link "programas" da página da discplina, a função* `(triangular)`.

   (a) *(V)[ ](F)[ ]* `(triangular)` *é uma função recursiva que resolve uma questão da lista 05.*

   (b) *(V)[ ](F)[ ] A função* `(ListaTriangular)` *lista os valores de* `(triangular)` *em ordem crescente.*

   (c) *(V)[ ](F)[ ] A função* `(ListaTriangular)` *lista os valores de* `(triangular)` *entre dois valores inteiros dados pelo usuário.*

   (d) *(V)[ ](F)[ ] A função* `(ListaTriangular)` *lista os valores de* `(triangular)` *entre dois valores inteiros, um dado pelo usuário e outro é zero.*

   (e) *(V)[ ](F)[ ] A função* `format` *se "assemelha" ao* `printf` *da linguagem* `C`.

5. *LISP Considere o arquivo* `meu_lisp.lsp` *que se encontra no link "programas" da página da discplina.*

   (a) *(V)[ ](F)[ ] A função* `soma` *calcula a soma dos elementos de uma lista dada.*

(b) (V)[ ](F)[ ] *A função* `soma` *não existe.*

(c) (V)[ ](F)[ ] *A função* `Soma` *calcula a soma dos elementos de uma lista dada.*

(d) (V)[ ](F)[ ] *Como* `Common LISP` *ignora diferença entre letras maiúsculas e minúsculas, tanto faz* `soma` *como* `Soma`.

(e) (V)[ ](F)[ ] `clisp` *pode ser "programado" para reconhecer a diferença entre letras maiúsculas e minúsculas sendo chamado, na linha de comandos com a* bandeira `-modern`.

6. *LISP*

(a) (V)[ ](F)[ ] *A sequência de operações*

```
(setq L '(1 2 3 4 5))
(pop L)
(push 10 L)
```

resulta na lista `(10 2 3 4 5)`

(b) (V)[ ](F)[ ] *Suponha que se tenha feito a inicialização* `(setq L '(1 2 3 4 5))` . *As duas operações*

```
(cons 10 L)
(push 10 L)
```

são equivalentes.

(c) (V)[ ](F)[ ] *Suponha que se tenha feito a inicialização* `(setq L '(1 2 3 4 5))` .
As duas operações

```
(cons 10 L)
(push 10 L)
```

não são equivalentes porque uma delas altera `L`.

(d) (V)[ ](F)[ ] *Suponha que se tenha feito a inicialização* `(setq L '(1 2 3 4 5))` .
O resultado de

```
(cdr (car L))
```

é 2.

(e) (V)[ ](F)[ ] *Suponha que se tenha feito a inicialização* `(setq L '(1 2 3 4 5))` .
A sequência de operações

```
(cdr (car L))
```

resulta num erro.

7. *Sempre considerando o arquivo* `meu_lisp.lsp` *que se encontra na página da disciplina.*

LISP

(a) (V)[ ](F)[ ] *A função* `Produto` *devolve 0 quando lhe for passada uma lista* `vazia`.

(b) (V)[ ](F)[ ] *A função* `Produto` *devolve 1 quando lhe for passada uma lista* `vazia` *e isto está consequente com fatorial em que um produto vazio corresponde ao 1.*

(c) (V)[ ](F)[ ] *A função* `Soma` *devolve 0 quando lhe for passada uma lista* `vazia`.

(d) (V)[ ](F)[ ] *A função* `swap` *permuta dois elementos de uma lista com dois elementos.*

(e) (V)[ ](F)[ ] *A função* `swap` *permuta o primeiro e o último elementos de uma lista qualquer com no mínimo dois elementos.*

8. *LISP*

Sempre considerando o arquivo `meu_lisp.lsp` *que se encontra na página da disciplina.*

(a) (V)[ ](F)[ ] `ultimo` *equivale a* `last` *que é uma função padrão do* `Common LISP`.

(b) (V)[ ](F)[ ] *A função* `inclui` *equivale a* `cons`.

(c) (V)[ ](F)[ ] *A função* `inclui` *não equivale a* `cons` *porque verifica se o parâmetro a ser inclúido pertence à lista.*

(d) (V)[ ](F)[ ] *A função* `segundo` *equivale a* `second`.

(e) (V)[ ](F)[ ] *Suponha que se tenha feito a inicilização* `(setq L '(1 2 3 4 5))`
A função `nth` *recebe dois parâmetros, o primeiro numérico (inteiro) e o segundo uma lista e* `(nth 20 L)` *equivale a* `(ultimo L)`.