Matemática Discreta
assunto desta Analise combinatória
T. Praciano-Pereira

Lista numero 01
tarcisio.praciano@gmail.com
Dep. de Computação

alun@:

5 de março de 2013 **Univ. Estadual Vale do Aca**

Documento escrito com LaTeX - sis. op. Debian/Gnu/Linux
http://www.multivariado.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.

Exercícios 1 *Conjuntos* *objetivo: Objetivo aqui programas sobre conjuntos*
palavras chave: *Conjuntos, conjuntos das partes, analise combinatória*

1. *Combinatória*

   Suponha que o conjunto $B = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 0\}$ *e que você tenha executado*

   ```
   (setq L '(1 2 3 4 5 6 7 8 9 0 ))
   ```

   (a) *(V)[ ](F)[ ] Uma das expressões LISP abaixo está errada (produz uma mensagem de erro)*

   ```
   (member 3 L)
   (member 3 (1 2 3 4 5 6 7 8 9 0 ))
   ```

   (b) *(V)[ ](F)[ ] O resultado de*

   ```
   (member 3 L)
   ```

   é uma lista cujo `car` *é 3.*

   (c) *(V)[ ](F)[ ]* $4 \subset B$

   (d) *(V)[ ](F)[ ]* $4 \in B$

   (e) *(V)[ ](F)[ ]* $\{4\} \subset B$

2. *Combinatoria Considere a definição dos conjuntos*

   (a) $A = \{0, 2, 4, 6, 8\}$

   (b) $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} = B$

   (c) $C = \{x; x \in \mathbf{Z}; x < 3\}$

   e que o símbolo $\mathbf{N}$ *represente o conjunto de todos os números naturais.*

   (a) *(V)[ ](F)[ ]* $C \subset \mathbf{N}$

   (b) *(V)[ ](F)[ ]* $C \cap A \subset \mathbf{N}$

   (c) *(V)[ ](F)[ ]* $C \cup A \subset \mathbf{N}$

   (d) *(V)[ ](F)[ ]* $C \subset B$

   (e) *(V)[ ](F)[ ]* $B \subset C$

3. *Combinatoria*

   Considere a definição dos conjuntos

   (a) $A = \{0, 2, 4, 6, 8\}$

   (b) $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} = B$

   (c) $C = \{x; x \in \mathbf{Z}; x < 3\}$

   que você tenha executado

   ```
   (setq L '(1 2 3 4 5 6 7 8 9 0 ))
   ```

   e que o símbolo $\mathbf{N}$ *represente o conjunto de todos os números naturais.*

   (a) *(V)[ ](F)[ ] Podemos dizer que* `L` *é equivalente A.*

   (b) *(V)[ ](F)[ ] Os objetos* `L` *A. não são comparáveis porque pertencem, cada um deles, a duas estruturas sintáticas diferentes. Por exemplo, é possível construir 10! listas diferentes usando os elementos de A.*

   (c) *(V)[ ](F)[ ] Os objetos* `L` *A. não são comparáveis porque pertencem, cada um deles, a duas estruturas sintáticas diferentes. Por exemplo, é possível construir 9! listas diferentes usando os elementos de A.*

   (d) *(V)[ ](F)[ ] A definição da função* `segundo` *que, dada uma lista* `L`*, exiba o segundo elemento de* `L`*, seria*

   ```
   (defun segundo (x)
        (cdr (car x))
   )
   ```

   (e) *(V)[ ](F)[ ]*

   ```
   (defun segundo (x)
        (car (cdr x))
   )
   ```

4. *A macro* `let` *atribue valores, localmente, a uma lista de variáveis com o formato*

```
(let       (
           (a1 valor1)
.....
           (aN valorN)
      )
)
```

e

```
(let*      (
           (a1 valor1)
.....
           (aN valorN)
      )
)
```

é a macro `let*` *que é conhecida como* `let paralelo` *em que todas as variáveis do* `let` *são atribuidas em paralelo então o valor de* `aK` *pode ser usado para o cálculo de* `aP; P > K`.

Combinatoria

(a) (V)[ ](F)[ ] *A função* `minha-equacao`

```
(defun minha-equacao (n)
      (let
            (
                  (x (sin n))
                  (y (cos n))
                  (z (* x y))
            )
            (+ n z)
      )
)
```

produz um erro.

(b) (V)[ ](F)[ ] *Esta versão da função* `minha-equacao` *calcula corretamente o valor final* `(+ n x y)`

```
(defun minha-equacao (n)
      (let
            (
                  (x (sin n))
                  (y (cos n))
            )
            (+ n x y)
      )
)
```


(c) (V)[ ](F)[ ] *Esta versão de* `minha-equacao` *usa a macro* `let*` *então o cálculo* `(z (* x y))` *é possível.*

```
(defun minha-equacao (n)
      (let*
            (
                  (x (sin n))
                  (y (cos n))
                  (z (* x y))
            )           (+ n z)
      )
)
```

(d) (V)[ ](F)[ ] *Querendo testar se as duas expressões*

```
(defun equacao (n)
(let*
      (
            (x (sin n))
            (y (cos n))
            (z (* x y))
      )
      (+ n z) ) )

(defun Equacao (n)
(+ n (* (cos n) (sin n))
)
```

são equivalentes, basta evaluar as expressões para um número finito de valores de n, e obtendo sempre o mesmo valor, elas serão equivalentes.

(e) (V)[ ](F)[ ] *Suponha que* `P` *representa uma função polinômial do grau 3 em LISP e que a avaliação* `(P n1) ...(P n3)` *resulte sempre em zero. Podemos concluir que foram encontradas as três possíveis raízes do polinômio* `P` *assim definido.*

5. LISP

(a) (V)[ ](F)[ ] *Quero definir um polinômio do terceiro grau. Preciso de uma lista com tamanho 4 para representar os coeficientes.*

(b) (V)[ ](F)[ ] *Quero definir um polinômio do terceiro grau. O polinômio fica bem definido com uma lista de tamanho 4.*

(c) (V)[ ](F)[ ] *Quero definir um polinômio do terceiro grau. Preciso de uma lista* `L` *com tamanho 4 para representar os coeficientes mas devo definir se é* `(car L)` *ou* `(last L)` *que representa o coeficiente de grau 3.*

(d) (V)[ ](F)[ ] `cond` *é a função equivalente ao* `case` *de* `C++`. *O resultado desta expressão será* `NIL`

```
(cond
     (() "não vazia")
     (() "vazia")
     (() "será ignorada")
     (() 3) )
```

(e) (V)[ ](F)[ ] `cond` *é a função equivalente ao* `case` *de* `C++`*. O resultado desta expressão será 3*

```
(cond
     (() "não vazia")
     (() "vazia")
     (() "será ignorada")
     ( 3) )
```

6. *A sequência de Fibonacci é definida pelas equações*

$$f(n) = \begin{cases} n = 0 & 1 \\ n = 1 & 1 \\ n > 1 & f(n-1) + f(n-2) \end{cases} \quad (1)$$

Qual das expressões LISP define a Fibonacci?

LISP

(a) (V)[ ](F)[ ] *(defun fib (n) (if (zerop n) 1) (if (= n 1) 1) (+ (fib (- n 2)) (fib (- n 1))))*

(b) (V)[ ](F)[ ] *(defun fib (n) (if (zerop n) 1 (if (= n 1) 1)) (+ (fib (- n 2)) (fib (- n 1))))*

(c) (V)[ ](F)[ ] *(defun fib (n) (cond ((= n 0) 1) ((= n 1) 1) (1 (+ (fib (- n 2)) (fib (- n 1))))))*

(d) (V)[ ](F)[ ] *(defun fib (n) (if (¡ n 2) 1 (+ (fib (- n 2)) (fib (- n 1)))))*

(e) (V)[ ](F)[ ] *(defun fib (n) (if (zerop n) 1 (if (= n 1) 1 (+ (fib (- n 2)) (fib (- n 1))))))*

7. *Usando a função* `member` *decida qual é a saída de dados.*

LISP

(a) (V)[ ](F)[ ]

```
(member rápido
'(antigamente computadores eram lentos e computador
   rápido é agradável)
)
```

responde `(é agradável)`

(b) (V)[ ](F)[ ]


```
(member 'rápido
'(antigamente computadores eram lentos e computador
   rápido é agradável)
)
```

responde `(rápido é agradável)`

(c) (V)[ ](F)[ ]

```
(member 'antigamente
'(antigamente computadores eram lentos e computador
   rápido é agradável)
)
```

responde `(antigamente computadores eram lentos e computador rápido é agradável).`

(d) (V)[ ](F)[ ]

```
(let ((n 2) (m 3))
   (cond
      ((< n m)
         (member 'antigamente
            '(antigamente computadores eram lentos e
      computador rápido é agradável)
         )
      )
      ((> m n)
         (member 'rapido
            '(antigamente computadores eram lentos e
      computador rápido é agradável)
         )
         )
   )
)
```

responde `(antigamente computadores eram lentos e computador rápido é agradável)` .

(e) (V)[ ](F)[ ]

```
(let* ((n 2) (m 3) (m n))
   (cond
      ((> n m) (format t "~a ~%" n))
      ((> m n) (format t "~a ~%" m))
   )
)
```

responde `nil`.

8.
```
(defun revapp (x L)
     (reverse (cons x (reverse L))) )
```

LISP

(a) (V)[ ](F)[ ] `(revapp 3 '(1 2 4))`
responde 3.

(b) (V)[ ](F)[ ] `(revapp 3 '(1 2 4))`
responde `(3 1 2 4)`

(c) (V)[ ](F)[ ] `(revapp 3 '(1 2 4))`
responde `(1 2 4 3)`

(d) (V)[ ](F)[ ]

(e) (V)[ ](F)[ ] `(revapp 3 '(1 2 4))`
responde `nil`

9. LISP

Quero definir uma função, `swap`*, que troque os elementos de um par, a função que desejo é*

(a) (V)[ ](F)[ ]

```
(defun swap (x)
     (revapp (car x) (cdr x))
)
```

(b) (V)[ ](F)[ ]

```
(defun swap (x y)
     (let*
          ((x y) (y x))
          (format t "~a ~% "'(x y))
     )
)
```

(c) (V)[ ](F)[ ]

```
(defun swap (x y)
     (psetq
          x y y x
     (
     (format t "~a ~% "'(x y))
)
```

(d) (V)[ ](F)[ ]

```
(defun swap (x y)
     (setq
          x y y x
     (
     (format t "~a ~% "'(x y))
)
```

(e) (V)[ ](F)[ ]


```
(defun swap (x y)
     (let
          ((x y) (y x))
          (format t "~a ~% "'(x y))
     )
)
```

10. LISP *Definindo:*

```
(defun swap2 (x)
     (psetq x y y x)
)
```

(a) (V)[ ](F)[ ] *Está mal definida.*

(b) (V)[ ](F)[ ] *Inverte listas*

(c) (V)[ ](F)[ ] *Inverte pares.*

(d) (V)[ ](F)[ ] *Sempre responde* `nil`

(e) (V)[ ](F)[ ] *Sempre responde* `t`