Prova de recuperação de LISP

META:
- Preciso tirar 7;
- Preciso fazer todos os exercícios listados abaixo:

LISP

1. Determine o valor das seguintes expressões em Lisp:

   (a) (and (or (> 2 3) (not (= 2 3))) (< 2 3))
   (b) (not (or (= 1 2) (= 2 3)))
   (c) (or (< 1 2) (= 1 2) (> 1 2))
   (d) (and 1 2 3)
   (e) (or 1 2 3)
   (f) (and nil 2 3)
   (g) (or nil nil 3)

2. (1.0 ponto) Converta as seguintes expressões da notação infixa da aritmética para a notação prefixa do Lisp:

   (a) 1 + 2 - 3
   (b) 1 - 2 * 3
   (c) 1 * 2 - 3
   (d) 1 * 2 * 3
   (e) (1 - 2) * 3
   (f) (1 - 2) + 3
   (g) 1 - (2 + 3)
   (h) 2 * 2 + 3 * 3 * 3

3. Quando tentamos avaliar em Lisp se o símbolo xpto é atômico, e escrevemos a expressão `(atom xpto)` recebemos uma mensagem de erro como retorno. Explique porque isso acontece.

4. Implemente uma função em Lisp que receba dois inteiros a e b e determine a lista de todos os valores inteiros desde a até b.

5. Implemente uma função em Lisp que determina se uma lista é palíndromo.

Questão 1: (2.0 pontos)
Defina (em **Lisp**) uma função recursiva chamada `compress` que elimina duplicação consecutiva de elementos em uma lista.

```
Exemplo:
(write(compress '(a a a a b c c a a d e e e e)))
(A B C A D E)
(write(compress '(1 2 3 3 2 4 5 5 7 4 4 4 9 8)))
(1 2 3 2 4 5 7 4 9 8)
```

P

Questão 1: (2.0 pontos)
Defina (em **Lisp**) uma função recursiva chamada **`recursive-reverse`** que retorna o reverso de uma lista de elementos.

P

Exemplo:

```
(write(recursive-reverse(list 2 3 5 7 11 13)))
(13 11 7 5 3 2)
(write(recursive-reverse(list(list 1 1 2 3 5 8))))
((1 1 2 3 5 8))
```

Haskell

1. Implemente (em Haskell) uma função que determina se um ano é bissexto. Tal função deve retornar `True` ou `False`.

2. Implemente (em Haskell) uma função que determina se um número inteiro positivo é um quadrado perfeito. Tal função deve retornar `True` ou `False`.

3. Implemente (em Haskell) uma função que determina o Máximo Divisor Comum (`MDC`) entre dois números inteiros positivos. Aplique o algoritmo de Euclides.

4. **(1.0 ponto)** Um grupo de n soldados está cercado por uma tropa inimiga e não existe esperança de vitória para eles caso não chegue reforço. Então, decidem que alguém entre eles terá que utilizar o único cavalo que possuem para escapar e buscar ajuda. Para escolher quem será este soldado todos eles formam um círculo, aí escrevem seus nomes em pedaços de papéis que são colocados dentro de um chapéu. Em seguida um dos nomes dentro do chapéu é sorteado e inicia-se uma contagem no sentido horário a partir do soldado cujo nome estava no papel. Quando a contagem atinge o valor k o soldado correspondente é retirado do círculo e a contagem recomeça a partir do soldado seguinte. Novamente é considerado k passos até que mais um soldado seja retirado do círculo. Este processo se repete até que sobre apenas um único soldado, que será o responsável por buscar ajuda para o grupo. Tal situação pode ser representada por uma recorrência matemática famosa que é atribuída a Flavius Josephus. Desta forma, se os soldados estão numerados de 1 até n e k é o número de passos, o último soldado restante pode ser determinado por meio da fórmula:

$$f(n,k) = ((f(n-1,k) + k - 1) \bmod n) + 1, \text{ com } f(1,k) = 1$$

P

Implemente (em Haskell) uma função josephus que resolve o problema de escolher o soldado para buscar ajuda para o grupo.

5. A conjectura de Collatz existe devido ao matemático alemão Lothar Collatz e também é conhecida como problema $3n+1$. A conjectura apresenta uma regra afirmando que, qualquer número natural, quando aplicado a esta conjectura, no final dará sempre 1. Tal regra consiste em dividir um número n por 2 se ele for par, ou multiplicar por 3 e somar 1 se for ímpar. A sequência de Collatz é obtida aplicando-se esta regra a partir do primeiro elemento da sequência até que seja atingido o valor 1. Por exemplo, se o valor de n for 5 será produzido a sequência 5, 16, 8, 4, 2, 1. Implemente (em Haskell) uma função collatzSequence que recebe um valor n e devolve a correspondente sequência de Collatz.

1. Implemente (em Haskell) uma função `dropEvery` que receba como argumentos uma lista e um valor inteiro `N` não negativo e exclua repetidamente cada `N`-*ésimo* elemento desta lista.

2. Implemente (em Haskell) uma função `ehDecrescente` que recebe uma lista (contendo apenas números inteiros) e retorna `True` caso ela esteja em ordem decrescente, ou `False` caso contrário. Os casos especiais de lista vazia e lista unitária podem ser considerados decrescente.

3. Implemente (em Haskell) uma função `rotate` que receba como argumentos uma lista e um valor inteiro N e rotacione (circularmente) os elementos desta lista N posições para a esquerda.

4. Implemente (em Haskell) uma função `num2digits` que transforma um inteiro em uma lista de dígitos correspondente ao número. (**Dica:** use as funções `mod` e `div`)

5. (1.0 ponto) Um número é dito *chic* se o dígito resultante da soma de seus dígitos ocorre no número. Se o resultado da soma dos dígitos for um número com mais de um dígito, então o processo deve ser repetido até que se obtenha um único dígito. Por exemplo, 1276 é *chic*, pois $1+2+7+6=16$, $1+6=7$. Por outro lado, 123 não é *chic* uma vez que $1+2+3=6$. Implemente (em Haskell) uma função `chic` que retorna `True` se o número for *chic* e `False` caso contrário.

1. Implemente (em Haskell) uma função para calcular o valor de $f(4,3)$, com $f(x,y)=x^2+y^2$, utilizando expressão lâmbda. (**Dica:** $\lambda xy \equiv \lambda x\ \lambda y$)

2. Implemente (em Haskell) uma função `mergeSort` que ordena uma lista aplicando o algoritmo de ordenação por intercalação. (O algoritmo Merge Sort baseia-se no princípio de que para juntar duas listas já ordenadas basta sempre olhar o primeiro elemento de cada lista e tomar o menor dos dois, até que todos os elementos das listas tenham sido consumidos. Assim, para ordenar uma lista desordenada, separam-se os elementos da lista em duas sublistas, ordenam-se as sublistas recursivamente, e juntam-se as duas sublistas ordenadas. A recursão termina porque uma lista de 0 ou 1 elemento já está ordenada.)

3. Se A é uma matriz $m \times n$ e B é uma matriz $n \times p$, então seu produto é uma matriz $m \times p$ definida como $(AB)_{ij} = \displaystyle\sum_{r=1}^{n} a_{ir}b_{rj}$, com $1 \le i \le m$ e $1 \le j \le p$. Sabendo disso, implemente (em Haskell) uma função `produto` que efetue a multiplicação das duas matrizes A e B.

4. (1.0 ponto) Sabemos que a série de Taylor para a função $\cos(x)$ é expressa como $\displaystyle\sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!}x^{2n}$. Implemente (em Haskell) uma função para calcular $\cos(1)$ considerando n termos da série de Taylor.

5. A conjectura de Goldbach diz que todo número inteiro par positivo maior que 2 é a soma de dois números primos. Por exemplo, $12 = 5 + 7$. Implemente (em Haskell) uma função `goldbach` que determina quais são os dois valores primos.

Questão 2: (1.5 pontos)
Implemente (em **Haskell**) uma função `rotate` que receba como argumentos uma lista e um valor inteiro N e rotacione (circularmente) os elementos desta lista N posições para a esquerda.

```
Exemplo:
Main> rotate [1,3,5,7,9,11] 3
[7,9,11,1,3,5]
Main> rotate "abcdefghijk" 2
"cdefghijkab"
```

P

Questão 3: (1.5 pontos)
Implemente (em **Haskell**) uma função `num2digits` que transforma um inteiro em uma lista de dígitos correspondente ao número. (**Dica:** use as funções `mod` e `div`)

```
Exemplos:
Main> num2digits 3701
[3,7,0,1]
Main> num2digits 0042
[4,2]
```

P

Questão 4: (1.5 pontos)
Um número é dito *chic* se o dígito resultante da soma de seus dígitos ocorre no número. Se o resultado da soma dos dígitos for um número com mais de um dígito, então o processo deve ser repetido até que se obtenha um único dígito. Por exemplo, 1276 é *chic*, pois $1+2+7+6=16$, $1+6=7$. Por outro lado, 123 não é *chic* uma vez que $1+2+3=6$. Implemente (em **Haskell**) uma função `chic` que retorna `True` se o número for *chic* e `False` caso contrário.

```
Exemplo:
Main> chic 436
True
Main> chic 571
False
```

P

Questão 1: (2.0 pontos)
Implemente (em **Haskell**) uma função `slice` que receba como argumentos uma lista e dois índices inteiros, *i* e *j*, e retorne uma lista contendo os elementos entre a *i-ésima* e *j-ésima* posições da lista de entrada original (incluindo os limites). (**Dica:** comece a contagem dos elementos com 1)

```
Exemplo:
Main> slice [3,5,7,11,13,17] 3 5
[7,11,13]
Main> slice "qwertyuiop" 7 8
"ui"
```

P - Última

Questão 2: (1.5 pontos)
Implemente (em **Haskell**) uma função `dropEvery` que receba como argumentos uma lista e um valor inteiro `N` não negativo e exclua repetidamente cada `N`-*ésimo* elemento desta lista.

```
Exemplo:
Main> dropEvery [1,3,5,7,9,11] 3
[1,3,7,9]
Main> dropEvery "abcdefghijk" 2
"acegik"
```

P - Última

Questão 3: (1.5 pontos)
A conjectura de Goldbach diz que todo número inteiro par positivo maior que 2 é a soma de dois números primos. Por exemplo, $12 = 5 + 7$. Implemente (em Haskell) uma função `goldbach` que determina quais são os dois valores primos.

```
Exemplo:
Main> goldbach 28
(5,23)
Main> goldbach 1000
(3,997)
```

P - Última

***Questão 4*:** (1.5 pontos)

Um grupo de n soldados está cercado por uma tropa inimiga e não existe esperança de vitória para eles caso não chegue reforço. Então, decidem que alguém entre eles terá que utilizar o único cavalo que possuem para escapar e buscar ajuda. Para escolher quem será este soldado todos eles formam um círculo, aí escrevem seus nomes em pedaços de papéis que são colocados dentro de um chapéu. Em seguida um dos nomes dentro do chapéu é sorteado e inicia-se uma contagem no sentido horário a partir do soldado cujo nome estava no papel. Quando a contagem atinge o valor k o soldado correspondente é retirado do círculo e a contagem recomeça a partir do soldado seguinte. Novamente é considerado k passos até que mais um soldado seja retirado do círculo. Este processo se repete até que sobre apenas um único soldado, que será o responsável por buscar ajuda para o grupo. Tal situação pode ser representada por uma recorrência matemática famosa que é atribuída a **Flavius Josephus**. Desta forma, se os soldados estão numerados de 1 até n e k é o número de passos, o último soldado restante pode ser determinado por meio da fórmula:

$$f(n,k) = ((f(n-1,k) + k - 1) \bmod n) + 1, \text{ com } f(1,k) = 1$$

Implemente (em **Haskell**) uma função `josephus` que resolve o problema de escolher o soldado para buscar ajuda para o grupo.

```
Exemplo:
Main> josephus 10 2
5
Main> josephus 100 7
50
```

P - Última

Prolog

1. Implemente (em Prolog) um predicado `primo(P)` que é satisfeito quando `P` é um número primo. Lembre-se que um número é primo quando ele é divisível apenas por 1 e ele próprio.

2. Implemente (em Prolog) um predicado `mdc(A, B, M)` que é satisfeito quando `M` é o máximo divisor comum de `A` e `B`. (**Dica:** use o algoritmo de Euclides)

3. Implemente (em Prolog) um predicado `perfeito(N)` que é satisfeito quando `N` é um número inteiro perfeito, isto é, `N` é igual à soma de todos os seus divisores menores que o próprio `N`. Como exemplo, temos que 6 é perfeito, pois $6 = 1 + 2 + 3$, mas 10 não é perfeito uma vez que $10 \neq 1 + 2 + 5$.

4. Implemente (em Prolog) um predicado `remover(LA, E, LB)` que remove apenas a primeira ocorrência do elemento `E` da lista `LA`, produzindo a lista `LB`. O predicado não deve falhar se o elemento não estiver na lista, neste caso ele deve retornar a lista de entrada `LA`.

5. (1.0 ponto) A sequência de Padovan é uma sequência de naturais $P(n)$ definida pelos valores iniciais $P(0) = P(1) = P(2) = 1$ e a seguinte relação recursiva

$$P(n) = P(n-2) + P(n-3) \text{ se } n > 2$$

Alguns valores da sequência são $1, 1, 1, 2, 2, 3, 4, 5, 7, 9, 12, 16, 21, 28, \ldots$ Implemente (em Prolog) um predicado `padovan(N, P)` que é verdadeiro sempre que `P` é o `N`-*ésimo* número da sequência de Padovan.

1. Implemente (em Prolog) um predicado `maximo(L, M)` que é satisfeito quando `M` é o maior elemento da lista numérica `L`.

2. Implemente (em Prolog) um predicado `produto(U, V, P)` que é satisfeito quando `P` é o produto escalar dos vetores `U` e `V` de $\mathbb{R}^n$.

3. (1.0 ponto) Implemente um predicado `superset(A, B)`, em Prolog, que recebe duas listas `A` e `B`, e que é satisfeito quando a lista `A` contém a lista `B`, isto é, todos os elementos de `B` são membros de `A`.

4. Implemente um predicado `insere(E, L1, L2)`, em Prolog, que recebe um elemento `E` e duas listas `L1`, `L2`, e que é satisfeito quando `L2` é o resultado da inserção de `E` em `L1` como primeiro elemento.

5. Desenhe a "árvore de execução" (*backtracking*) do código Prolog apresentado a seguir, para as consultas `?- b([1, 0, 1])` e `?- b([1, 0, 2])`. (**Dica:** utilize o comando `trace` do SWI-Prolog)

```
d(0).
d(1).
b([A,B,C]) :- d(A), d(B), d(C).
```

1. Implemente (em Prolog) um predicado `fibonacci(N, F)` que determina o N-*ésimo* valor na sequência de Fibonacci. Lembre-se que $F(0) = 0$, $F(1) = 1$ e $F(N) = F(N-1) + F(N-1)$, para $n > 1$.

2. Mostre como é possível simular/emular, em Prolog, os laços (ou *loops*) **while** e **for** de linguagens imperativas como C. Exemplifique sua solução com um código que some os n primeiros números naturais, ou some os n primeiros números naturais pares/ímpares.

3. (1.0 ponto) O Quicksort é um algoritmo de ordenação que adota a estratégia de divisão e conquista. Ele pode ser definido, de forma sucinta, em 3 etapas: (i) Escolha de um elemento da lista denominado pivô; (ii) Rearranjo da lista de forma que todos os elementos anteriores ao pivô sejam menores que ele, e todos os elementos posteriores ao pivô sejam maiores que ele; (iii) Ordenação, recursiva, da sublista dos elementos menores e da sublista dos elementos maiores. Escreva um predicado `quicksort(L, M)`, em Prolog, que recebe uma lista `L` e é satisfeito quando a lista `M` é formada pelos elementos de `L` ordenados.

4. Diferenciação de funções matemáticas é um exemplo de aplicação de manipulação simbólica em que a linguagem Prolog é útil, pois não é preciso empregar computação numérica. Então defina, em Prolog, regras de diferenciação para funções $f(x) = x^n$, com n natural, de maneira que o predicado `derivada(U,V)` é satisfeito quando `V = dU/dx`. Lembre-se que $\dfrac{du^n}{dx} = nu^{n-1}\dfrac{du}{dx}$, para inteiros $n > 0$.

5. Um retângulo é representado pela estrutura `retangulo(A,B,C,D)`, cujos vértices são os pontos `A`, `B`, `C` e `D`, nesta ordem.

   (a) Implemente (em Prolog) o predicado `regular(R)` que resulta em `true` apenas se `R` for um retângulo cujos lados sejam verticais e horizontais.

   (b) Implemente (em Prolog) o predicado `quadrado(R)` que resulta em `true` apenas se `R` for um retângulo cujos lados tem a mesma medida.

   (c) Implemente (em Prolog) o predicado `diagonal(R, D)` que é satisfeito se `R` é um quadrado e `D` retorna o valor da sua diagonal.

***Questão 4*:** (1.5 pontos)
Implemente (em **Prolog**) um predicado `mdc(A, B, M)` que é satisfeito quando `M` é o máximo divisor comum de `A` e `B`. (**Dica:** use o algoritmo de Euclides)

```
Exemplo:
?- mdc(15, 5, M).
M = 5.
?- mdc(19, 3, M).
M = 1.
```

P - Última

***Questão 5*:** (1.5 pontos)
Implemente um predicado `superset(A, B)`, em **Prolog**, que recebe duas listas `A` e `B`, e que é satisfeito quando a lista `A` contém a lista `B`, isto é, todos os elementos de `B` são membros de `A`.

```
Exemplo:
?- superset([a,b,c], [c,a]).
true .
?- superset([a,b,c], [c,a,d]).
false.
```

P - Última

***Questão 6*:** (2.0 pontos)
Suponha que números complexos sejam implementados em **Prolog** da seguinte forma: o número $a+bi$ é representado pela estrutura `complexo(a, b)`. Escreva um predicado `potencia(X, N, P)` que seja satisfeito quando `P` é o produto de `N` complexos iguais a `X`. (**Dica:** $(a+bi)(c+di) = (ac-bd)+(bc+ad)i$)

```
Exemplo:
?- potencia(complexo(2, 1), 2, P), potencia(complexo(2, 1), 3, Q).
P = complexo(3, 4),
Q = complexo(2, 11) .
```

P - Última

Questão 5: (1.5 pontos)
Implemente (em **Prolog**) um predicado `primo(P)` que é satisfeito quando `P` é um número primo. Lembre-se que um número é primo quando ele é divisível apenas por 1 e ele próprio.

```
Exemplo:
?- primo(7).
true.
?- primo(10).
false.
```

P

Questão 6: (2.0 pontos)
Implemente (em **Prolog**) um predicado `remover(LA, E, LB)` que remove apenas a primeira ocorrência do elemento `E` da lista `LA`, produzindo a lista `LB`. O predicado não deve falhar se o elemento não estiver na lista, neste caso ele deve retornar a lista de entrada `LA`.

```
Exemplo:
?- remover([1,3,5,1,3,11,13], 3, L).
L = [1, 5, 1, 3, 11, 13].
?-  remover([2,4,6,8,10], 5, L).
L = [2, 4, 6, 8, 10].
```

P

Questão 5: (1.5 pontos)
Implemente (em **Prolog**) um predicado `mdc(A, B, M)` que é satisfeito quando `M` é o máximo divisor comum de `A` e `B`. (**Dica:** use o algoritmo de Euclides)

```
Exemplo:
?- mdc(15, 5, M).
M = 5.
?- mdc(19, 3, M).
M = 1.
```

P

Questão 6: (2.0 pontos)
A sequência de Padovan é uma sequência de naturais $P(n)$ definida pelos valores iniciais $P(0) = P(1) = P(2) = 1$ e a seguinte relação recursiva

$$P(n) = P(n-2) + P(n-3) \text{ se } n > 2$$

Alguns valores da sequência são $1, 1, 1, 2, 2, 3, 4, 5, 7, 9, 12, 16, 21, 28, \ldots$ Implemente (em **Prolog**) um predicado `padovan(N, P)` que é verdadeiro sempre que `P` é o `N`-*ésimo* número da sequência de Padovan.

```
Exemplo:
?- padovan(5, 3).
true.
?- padovan(7, 4).
false.
```

P