Autorreferência

Marco A L Barbosa malbarbo.pro.br

1. Defina uma função que calcule a quantidade de elementos de uma lista.
2. Defina uma função que receba com entrada uma lista `lst` e um elemento `a` e devolva uma lista que é como `lst` mas sem as ocorrências de `a`.

   ```
   > (remove-todos (list 3 2 3 5 8 3) 3)
   '(2 5 8)
   ```

3. Defina uma função que receba como entrada uma lista `lst` e um elemento `a` e devolva uma lista que é como `lst` mas com `a` no final.

   ```
   > (cons-fim (list 5 2 8) 3)
   '(5 2 8 3)
   ```

4. Defina uma função que que receba como entrada uma lista `lst` e devolva uma lista com os mesmos elementos de `lst` mas em ordem reversa. Dica: use a função `cons-fim`.

   ```
   > (inverte (list 1 2 3 4 5))
   '(5 4 3 2 1)
   ```

5. Defina uma função que determine se uma lista é palíndromo. Dica: use a função `inverte`.
6. Defina uma função que receba como entrada uma lista `lst` de números naturais e devolva uma lista que é como `lst` mas sem os números pares.
7. Defina uma função que devolva o último elemento de uma lista. Use a função `error` (com uma string de mensagem como argumento) para indicar erro se a lista for vazia.
8. Defina uma função que encontre o valor máximo de uma lista de números.
9. Defina uma função que receba como entrada uma lista `lst` de números em ordem não decrescente e um número `n` e devolva uma lista com os elementos de `lst` e com `n` em ordem não decrescente.

   ```
   > (insere-ordenado (list 2 8 10) 5)
   '(2 5 8 10)
   ```

10. Defina uma função que receba como entrada uma lista de números e devolva uma lista com os mesmos valores de entrada mas em ordem não decrescente. (Lembre-se de aplicar a receita de projeto, não tente implementar um método de ordenação qualquer, a receita te levará a implementar um método específico). Dica: use a função `insere-ordenado`.
11. [pp99 1.10] Defina uma função que receba como entrada uma lista `lst` e devolva uma nova lista que é como `lst` com apenas uma ocorrência dos elementos repetidos consecutivos.

    ```
    > (remove-duplicates (list 1 1 1 1 2 3 3 4 4 5 5 5))
    '(1 2 3 4 5)
    ```

12. Defina uma função que calcule o fatorial de um número.
13. Utilizando apenas as funções primitivas `zero?`, `add1` e `sub1`, escreva as funções `+`, `-` e `*`. Cada função deve receber como parâmetro dois números naturais e executar a operação aritmética apropriada.

14. [tspl 2.8.6] Recursão indireta é quando dois (ou mais) procedimentos usam um ao outro. Defina duas funções `impar?` e `par?`, uma em termos da outra. (Dica: O que cada uma deve retornar quando o argumento for 0?)

15. Defina uma função que receba como entrada um número qualquer a e um número natural n e calcule o valor a^n.

16. Projete um algoritmo que receba como entrada dois números naturais maiores que zero, `n` e `x`, e devolva uma lista com os divisores de `x` que são menores ou iguais a `n` (não se preocupe com a ordem dos valores na resposta).

17. Um número natural é perfeito se a soma dos seu divisores, exceto ele mesmo, é igual a ele. Por exemplo, o número 6 é perfeito porque $6 = 1 + 2 + 3$. Projete uma função que verifique se um número natural é perfeito. (Essa função não é recursiva! Use a função que produz a lista de divisores e a função que soma os valores de uma lista para implementar essa função)

18. Defina uma função que receba como entrada uma lista aninhada `lst` e devolva uma nova lista aninhada como os mesmo elementos de `lst` mas em ordem reversa.

```
> (reverse* (list (list 2 3) 8 (list 9 (list 10 11) 50) (list 10) 70))
'(70 (10) (50 (11 10) 9) 8 (3 2))
```

19. Defina uma função que receba como entrada uma árvore binária `t` e um número `n` e devolva uma nova árvore binária que é como `t` mas com `n` somado a cada elemento.

20. Defina uma função que verifique se uma árvore binária é uma árvore binária de busca. Uma árvore binária de busca tem as seguintes propriedades: 1) A subárvore a esquerda contém valores nos nós menores que o valor no nó raiz. 2) A subárvore a direita contém valores nos nós maiores que o valor no nó raiz. 3) As subárvores a esquerda e a direita também são árvores binárias de busca.

21. Defina uma função que verifique se um elemento está em uma árvore binária de busca.

Referências

- [tspl]. The Scheme Programming Languague
- [pp99]. 99 problemas para resolver em (Prolog) Racket