Paradigma de Programação Funcional

3.0 - Dados compostos - Exercícios

3.0.1) Defina uma função que verifique se um determinado elemento está em uma lista.

3.0.2) 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)
```

3.0.3) 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)
```

3.0.4) Defina uma função que que rebeba 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)
```

3.0.5) Defina uma função que determine se uma lista é palíndromo. Dica: use a função `inverte`.

3.0.6) Defina uma função que receba como entrada uma lista `lst` de números e um número `n` e devolva uma lista com cada elemento de `lst` somado com `n`.

```
> (lista-add-num (list 2 6 1 4 10) 3)
'(5 9 4 7 13)
```

3.0.7) Defina uma função que receba como entrada uma lista `lst` de número naturais e devolva uma lista que é como `lst` mas sem números pares.

3.0.8) 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.

3.0.9) Defina uma função que encontre o valor máximo de uma lista de números.

3.0.10) Defina uma função que receba como entrada uma lista `lst` de números em ordem crescente e um número `n` e devolva uma lista com os elementos de `lst` e com `n` em ordem crescente.

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

3.0.11) Defina uma função que receba como entrada uma lista de números e devolva uma lista como os mesmos valores de entrada mas em ordem crescente. (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`.

3.0.12) [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)
```

3.0.13) 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))
```

3.0.14) Defina uma função que avalie uma expressão aritmética em Racket que contenha apenas constantes. Cada operação precisa de exatamente 2 parâmetros. Dica: crie uma definição (semelhante a de lista aninhada) que represente expressões aritméticas, crie um template baseado na definição do tipo e use o template para resolver o exercício.

```
> (avalie (list + (list * 3 (list - 4 5)) (list / 10 2))
2
```

3.0.15) 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.

3.0.16) 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.

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

Referências

- [pp99]. 99 problemas para resolver em (Prolog) Racket

Licença