Universidade Federal de Minas Gerais
Ciência da Computação

Linguagens de Programação - Haniel Barbosa

Lista de Exercícios 1

Monitor: Matheus Cândido Teixeira

1. Escreva uma função `cube : real -> real` que retorne o cubo de um número real.

   input: 3.0
   output: val it = 27.0 : real

2. Escreva uma função `pow : int * int -> int` que receba um número **n** e um expoente **e** e retorne o valor de n^e.

   input: (5, 2);
   output: val it = 25 : int

3. Escreva uma função `sumLists: int list * int list-> int list` que receba duas listas de mesmo tamanho e retorne uma lista em que o i-ésimo elemento é a soma do i-ésimo elemento da lista 1 com o i-ésimo elemento da lista 2. Não se preocupe com listas de tamanho diferentes como entrada.

   input: ([2, 5, 10], [1, 15, 4])
   output: val it = [3, 20, 14] : int list

4. Escreva uma função `max: int list -> int` que retorne o maior valor de uma lista.

   input: [2, 1, 7, 3]
   output: val it = 7 : int

5. Escreva uma função `cumSum: int list -> int list` que receba uma lista e retorne uma lista contendo a soma parcial de seus elementos.

   input: [6, 10, 3, 11]
   output: val it = [6, 16, 19, 30] : int list

6. Escreva uma função `greet: string -> string` que receba um nome e retorne um cumprimento para esse nome. Se nenhum nome for passado como entrada, a função deve retornar *Hello nobody*.

   input: "Janis"
   output: val it = "Hello Janis" : string

7. Escreva uma função `split: string -> string list` que receba uma frase **f** e retorne uma lista em que cada elemento é uma palavra de **f**. Considere que cada palavra na frase pode estar separada por espaço, ou pelos caracteres ",", ".", ou "-".

DICA: A biblioteca de SML é muito rica, e apresenta diversas interfaces que assistem o programador dessa linguagem. Pesquisem sobre as interfaces de SML STRING e CHAR. Elas possuem métodos que vão auxiliar nessa questão.

Notem que entradas em unicode (por exemplo contendo carecteres acentuados) podem ter comportamenos diferentes do esperado. Podem supor que não haverão entradas assim.

input: "Good morning to-you"
output: val it = ["Good", "morning", "to", "you"] : string list

8. Escreva uma função `allTrue: bool list -> bool` que receba uma lista de booleanos e retorne **true** apenas se todos os elementos da lista de entrada forem verdadeiros.

   input: [true, true, false, true]
   output: val it = false : bool

   input: [true, true, true]
   output: val it = true : bool

9. Defina um tipo algébrico de dados `dinheiro`, que possa representar quantidades em centavos (tipo `int`), em reais (tipo `real`), ou um par Nome x reais. A partir desse tipo, defina uma função `amount: dinheiro -> int` que recebe um tipo `dinheiro` como entrada e retorne a quantidade em centavos correspondente à entrada.

   input: val d = Real(2.0) : dinheiro
   output: val it = 200 : int

   input: val d = Centavos(2) : dinheiro
   output: val it = 2 : int

   input: val d = Pessoa_Dinheiro("Gene", 2.5)) : dinheiro
   output: val it = 250 : int

10. O nosso planeta demora 365 dias para completar uma órbita em torno do sol, nós marcamos essa forma de tempo como 1 ano. Esse tempo é diferente em outros planetas do nosso sistema solar. Por exemplo, Mercúrio leva 88 dias terrestres para completar sua órbita, enquanto Marte demora 687. Seguido os dados abaixo, crie uma função `planetAge: int * Planeta -> int` que recebe como entrada a idade de alguém em meses e um planeta. A função deve retornar a idade em dias dessa pessoa de acordo com o planeta dado como entrada. Considere que cada ano tem 12 meses, com quantidade igual de dias, independente do planeta.

    input: (24, Jupiter)
    output: val it = 8664 : int

Planeta	Tempo de Órbita (dias terrestres)
Mercúrio	88
Vênus	225
Terra	365
Marte	687
Júpiter	4332
Saturno	10.760
Urano	30.681
Netuno	60.190

11. Considerando a definição de *Binary Search Tree* vista na aula sobre tipos de dados algébricos:

    ```
    datatype btree = Leaf | Node of (btree * int * btree);
    ```

    Escreva uma função `sumAll: btree -> int` que percorra a árvore retorne a soma de todos os valors nos nós internos desta.

    input: val t = Node (Node (Leaf, 1, Leaf), 6, Node (Leaf, 12, Leaf))
    output: val it = 19 : int

12. Escreva uma função `multiPairs: int list * int list -> int list` que receba duas listas de mesmo tamanho e retorne uma lista contendo o produto de cada par de elementos das listas de entrada. Sua solução deve usar combinadores.

    input: ([2, 5, 10], [4, 10, 8]);
    output: val it = [8, 50, 80] : int list

13. Utilizando combinadores, escreva um programa em SML que retorne o quadrado da soma de dois números inteiros.

    input: (4,5)
    output: val it = 81 : int

14. Considere uma linguagem genérica com o operador `+` posssuido associatividade a esquerda e que é sobrecarregado para suportar os seguintes tipos:

    - `int * real -> real`
    - `int * int -> int`
    - `real * int -> real`
    - `real * real -> real`

    Suponha que a variável `i` seja do tipo `int` e a variável `r` seja do tipo `real`. Para cada operador `+` em cada expressão abaixo, diga qual tipo de `+` é utilizado:

    (a) `i+r`
    (b) `i+r+i`
    (c) `i+(r+i)`
    (d) `i+i+r+(r+i)`

15. Explique qual a razão dos erros abaixo ocorrerem em ML:

(a)
```
- if 1 < 2 then 1;
= ;
= ;
= ;
stdIn:5.6-7.2 Error: syntax error: deleting  SEMICOLON
   SEMICOLON SEMICOLON
```

(b)
```
- 1 * 2.0;
stdIn:1.2-10.2 Error: operator and operand do not agree [
   overload - bad instantiation]
  operator domain: 'Z[INT] * 'Z[INT]
  operand:         'Z[INT] * real
  in expression:
    1 * 2.0
```

(c)
```
- fun fact n = n * fact (n-1)
= | fact 0 = 1;
stdIn:31.1-39.13 Error: match redundant
          n => ...
    -->   0 => ...
```