Universidade Federal de Minas Gerais
Ciência da Computação

Linguagens de Programação - Haniel Barbosa

Lista de Exercícios 1

Monitor: Matheus Cândido Teixeira

Respostas

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

   input: 3.0
   output: val it = 27.0 : real

```
fun cube x:real = x * x * x;
```

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

```
fun pow (0,e) = 0
| pow (n,0) = 1
| pow (n,1) = n
| pow (n,e) = n * pow(n,e-1);
```

3. Escreva uma função `sumLists: 'a list * 'a list-> 'a 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

```
datatype T = E | I of int | R of real;

fun sum (I(x), I(y)) = I(x+y)
| sum (R(x), R(y)) = R(x+y)
| sum (I(x), R(y)) = E
| sum (R(x), I(y)) = E
| sum (E,_) = E
| sum (_,E) = E;

fun sumLists ([],_) = []
| sumLists (_,[]) = []
| sumLists (h::t,h2::t2) =
```

```
13    sum(h,h2)::sumLists(t,t2);
14
15  fun sumLists ([],_) = []
16  | sumLists (_,[]) = []
17  | sumLists (h1::t1,h2::t2) = (h1+h2)::sumLists(t1,t2);
```

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

```
1   fun maxAux (x, y) = if x >= y then x else y;
2
3   fun max [] = 0
4   | max (x::[]) = x
5   | max (x::xs) =
6     let
7       val y = max(xs)
8     in
9       maxAux(x, y)
10    end;
```

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

```
1   fun cumSum [] = []
2   | cumSum (x::[]) = [x]
3   | cumSum (x::y::xs) = x :: cumSum(x+y::xs);
```

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

```
fun greet x =
  let
    val g = "Hello "
  in
    if String.size x > 0 then g ^ x else g ^ "nobody"
  end;
```

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.

   input: "Bom dia,pra-você"
   output: val it = ["Bom", "dia", "pra", "você"] : string list

```
fun mustSplit c =
  let
    val isComma = (Char.compare(c, #",") = EQUAL)
    val isDot = (Char.compare(c, #".") = EQUAL)
    val isHyphen = (Char.compare(c, #"-") = EQUAL)
  in
    Char.isSpace c orelse isComma orelse isDot orelse isHyphen
  end;

fun split x = String.tokens mustSplit x;
```

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

```
fun allTrue [] = false
  | allTrue (x::[]) = x
  | allTrue (x::xs) = x andalso allTrue(xs);
```

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) : 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

```
datatype dinheiro = Centavos of int | Reais of real |
   Pessoa_Dinheiro of string * real;

fun amount (Centavos(x)) = x
  | amount (Reais(x)) =
    let
      val cast = Real.toInt IEEEReal.TO_NEAREST
    in
      cast (x * 100.0)
    end
  | amount (Pessoa_Dinheiro(x,y)) =
    let
      val cast = Real.toInt IEEEReal.TO_NEAREST
    in
      cast (y * 100.0)
    end;
```

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.

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

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

```
datatype Planeta  = Mercurio | Venus | Terra | Marte | Jupiter
   | Saturno | Urano | Netuno;

fun planetAge (x, y) =
  let
    val year = x div 12;
  in
    case y of
      Mercurio => year * 88
    | Venus => year * 225
    | Terra => year * 365
    | Marte => year * 687
    | Jupiter => year * 4332
    | Saturno => year * 10760
    | Urano => year * 30681
    | Netuno => year * 60190
  end;
```

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 valores nos nós internos desta.

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

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

fun sumAll (Leaf) = 0
  | sumAll (Node(tl,x,tr)) =
  let
    val sumtl = sumAll(tl)
    val sumtr = sumAll(tr)
  in
    x + sumtl + sumtr
  end;
```

12. Escreva uma função `multiPairs: (int * int) list -> int list` que receba duas listas e retorne uma lista contendo o produto de cada par de elementos das listas de entrada. Sua solução deve conter apenas **uma** linha de código.

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

```
fun multiPairs (x,y) = ListPair.map (fn (a,b) => a*b) (x,y);

(* fun multiPairs (x,y) = ListPair.map op * (x,y); *)
```

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

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

```
fun square x = x * x;
fun sum (x,y) = x + y;
fun compose (f,g,x,y) = f (g (x,y));

compose(square, sum, 4, 5);
```

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 associativiade determina a ordem de aplicação de operadores de mesma precedência.

    (a) `i+r`
    real

    (b) `i+r+i`
    real e real

    (c) `i+(r+i)`
    real e real

    (d) `i+i+r+(r+i)`
    int, real, real e real

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
```

Diferente de linguagens imperativas, onde `if` representa um *comando*, em ML `if` é uma *expressão*. Isso implica que `if` deve retornar algum valor. No exemplo acima, não temos um valor de retorno definido para o caso de a expressão condicional do `if` for avaliada como falsa.

(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
```

Apesar de o operador `*` ser sobrecarregado em SML, quando usado em uma expressão os dois operandos devem ter o mesmo tipo. No exemplo acima temos um inteiro e um real sendo utilizados.

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

O erro acima indica que definidos um caso de casamento de padrão redundante. Pois quando definimos a função para um número arbitrário `n`, o que inclui 0. O algoritmo de casamento de padrões avalia tais padrões na ordem em que foram definidos, por isso para a função acima funcionar da forma esperada devemos definir o caso de `n=0` antes do caso de uma entrada genérica.