Programming Assignment #5 (SML)

CS–671

due 8 April 2014 11:59pm

1. Write a function `last` that returns the last element of a list. The function raises an exception (of any type) when applied to the empty list. 5 pts

2. Write a function `natList` such that `natList(n)` returns the list `[1,2,3,...,n]`. `natList(0)` is `[]` and `natList(1)` is `[1]`. Negative numbers are invalid inputs. 5 pts

3. Write a function `sum` that computes the sum of a list of integers. The empty list is a valid input. 5 pts

4. Write a function `prod` that computes the product of a list of integers. The empty list is a valid input. 5 pts

5. Write a function `sum2` that computes the sum of a list of lists of integers. The empty list is a valid input. 10 pts

6. Write a function `isLetter` that returns true if a character is a letter, false otherwise. 5 pts

7. Write a function `toLower` that returns a lowercase character from an uppercase character. Parameters other than `#"A"`, `#"B"`, ..., `#"Z"` are left unchanged. 5 pts

8. Write a function `palindrome` that returns true if a string is a palindrome. A string is a palindrome if its letters read the same from left to right and from right to left. For instance, the following strings are palindromes and should make the function return true: 10 pts

   - `"A man, a plan, a canal, Panama!"`
   - `"Won't cat lovers revolt? Act now!"`

9. Write a function `hanoi` that builds a list of moves to solve the *Towers of Hanoi* problem. Each move is represented as a pair *(source, destination)*. A call to `hanoi(n,A,B,C)` should solve the problem of moving n discs from peg A to peg C using peg B as an intermediate. The solution must have minimal length (i.e., smallest number of moves), which is $2^n - 1$. 10 pts

10. Write a function `factor` that factorizes an integer ($\geqslant 2$) into its constituent primes. Each factor is represented as a pair *(value, exponent)* and the function returns a list of such pairs. No factor should appear more than once in the list and the list is sorted in order of increasing factor values. For instance, `factor(1111104)` should return `[(2,6),(3,3),(643,1)]` ($1111104 = 2^6 \times 3^3 \times 643$). 10 pts

11. Write a function `multiply` that is the inverse of `factor`: it reconstructs a number from its list of prime factors. For any integer $n \geqslant 2$, `n=multiply(factor(n))` should always be true. *This problem must be solved using tail recursion only*: `multiply` and any intermediate function used to solve the problem must be tail recursive (or not recursive at all). 10 pts

12. Write a function `printFact(n)` that prints the prime factors decomposition of `n` in a human-readable form. For instance, `printFact(1776)` prints `1776 = 2^4 * 3 * 37`, followed by a newline. When the number is prime, the function prints a message instead. For instance, `printFact(1789)` prints `1789 is prime`, followed by a newline. 10 pts

13. Write a function `isPerm` that checks that two lists are permutations of each other. Be careful with duplicates: 10 pts

```
isPerm ([1,2,3,2], [2,2,3,1]) = true
isPerm ([1,2,3,2], [2,1,3,1]) = false
```

Notes:

- This assignment must be submitted in a file named `5.sml` inside a `sml` subdirectory of the repository. This file can load other files using function `use` on relative paths, if necessary.
- Except for question 11, recursive functions don't have to be tail recursive. Functions can rely on the correctness of their input and don't have to deal with exceptional cases due to erroneous input, overflow, etc.
- The warning `calling polyEqual` can be ignored (it tells you that you are using a costly comparison operator, which is fine). *Do not* ignore other warnings.
- Functions don't have to use pattern matching. In particular, functions that work on lists can be written using `hd`, `tl`, `null`, etc. However, you are free to use pattern matching if you want to.
- Functions can use previously defined functions, as well as other intermediate functions that are not part of the assignment. It is not required to hide intermediate functions in `let` and `local` blocks.
- Functions `Char.toLower`, `Char.isAlpha`, `Int.min`, `List.length`, `List.last`, `List.nth`, `List.take`, `List.drop`, `List.foldl`, `List.foldr`, ... from the standard library cannot be used. Functions have to be programmed explicitly (for this assignment). Functions `Char.ord` and `Char.chr` can be used.
- Code of the form: `"if ... then true else false"` or `"if ... then false else true"` should never be used.
- Below is a simple algorithm to calculate the prime factors of `n`, expressed in Java. You are not required to use this particular algorithm.

```
1     int i = 2;
2     double s = Math.sqrt(n);
3     while (i <= s) {
4       if (n % i == 0) {
5         System.out.printf("%d ", i);
6         n /= i;
7         s = Math.sqrt(n);
8       } else {
9         i += 1;
10      }
11    }
12    System.out.println(n);
13  }
```

- Functions must have the following types:

```
val last = fn : 'a list -> 'a
val natList = fn : int -> int list
val sum = fn : int list -> int
val prod = fn : int list -> int
val sum2 = fn : int list list -> int
val isLetter = fn : char -> bool
val toLower = fn : char -> char
val palindrome = fn : string -> bool
val hanoi = fn : int * 'a * 'a * 'a -> ('a * 'a) list
val factor = fn : int -> (int * int) list
val multiply = fn : (int * int) list -> int
val printFact = fn : int -> unit
val isPerm = fn : ''a list * ''a list -> bool
```

Be careful that the type parameter in `last` and `hanoi` is `'a`, not `''a`.

If a function is not implemented, it should be replaced with a dummy function of type `'a -> 'b` so compilation can proceed:

```
(* Function printFact is not implemented *)
fun printFact _ = raise Fail "not implemented"
```