Introduction to Programming 1: Assignment 2

Due: October 1, 2015. 11 pm

Important Instructions: Submit your solution in a single file named *loginid.2.hs* on Moodle. For example, if I were to submit a solution, the file would be called *spsuresh.2.hs*. You may define auxiliary functions in the same file, but the solutions should have the function names specified by the problems.

---

1. Define a function *segments* which takes a finite list *xs* as its argument and returns the list of all the segments of *xs*. (A segment of *xs* is a selection of adjacent elements of *xs*.)

   Sample cases:

   ```
   segments []      = [[]]
   segments [1,2,3] = [[1,2,3], [1,2], [2,3], [1], [2], [3]]
   ```

2. A *partition* of a positive integer *n* is a representation of *n* as the sum of any number of positive integral parts. Define a function *parts* which returns the list of distinct partitions of an integer *n*. Each partition of *n* is represented as a non-decreasing list of positive integers that sum up to *n*. The various partitions can themselves be listed in any order in the output.

   Sample cases:

   ```
   parts 1 = [[1]]
   parts 4 = [[1,1,1,1],[1,1,2],[1,3],[2,2],[4]]
   parts 5 = [[5],[2,3],[1,4],[1,2,2],[1,1,3],[1,1,1,2],[1,1,1,1,1]]
   ```

3. A list of numbers is said to be *steep* if each element of the list is at least as large as the sum of the preceding elements. Define a function *llsg* such that *llsg xs* is the length of the longest steep segment of *xs*.

   Sample cases:

   ```
   llsg [] = 0
   llsg [0] = 1
   llsg [225] = 1
   llsg [1,2] = 2
   llsg [1,2,3,5,12,17] = 4
   llsg [1,2,3,6,12,17] = 5
   ```

4. Consider strings composed of the letters *a* and *b*. We say that the string s_2 is *next to* the string s_1 iff one of the following conditions hold:

   (a) s_1 is the all-*b*'s string of length n and s_2 is the all-*a*'s string of length $n + 1$, for some $n \geq 0$.

(b) s_1 and s_2 can be split into $s_1' x s_1''$ and $s_2' y s_2''$ respectively, such that

- $s_1' = s_2'$,
- x and y are strings of length 1, with $x = a$ and $y = b$,
- s_1'' is the all-b's string of some length $m \geq 0$, and s_2'' is the all-a's string of the same length m.

Define a Haskell function *isnext* that takes two strings as inputs and checks if the second is next to the first.

Sample cases:

```
isnext "" "a"             = True
isnext "bbb" "aaaa"       = True
isnext "bbabbb" "bbbaaa"  = True
isnext "bbb" "aaaaa"      = False
isnext "baabbb" "bbbaaa"  = False
```

5. Define a function *next* that takes a string (involving the letters a and b) and outputs the next string.

   Sample cases:

```
next ""        = "a"
next "bbb"     = "aaaa"
next "bbabbb"  = "bbbaaa"
```

6. Define a function *abundant* that takes a string s_1 (involving the letters a and b) as input and outputs *True* when s_1 has at least two occurrences of the substring ab.

   Sample cases:

```
abundant ""             = False
abundant "bbb"          = False
abundant "bbabbb"       = False
abundant "abab"         = True
abundant "abbababbaba"  = True
```

7. Define a function *abundants* that outputs the list of all abundant strings in the order defined by our function *next*. For example, `take 10 abundants` is the following list.

```
["abab","aabab","abaab","ababa","ababb","abbab","babab","aaabab","aabaab","aababa"]
```