CS 403, Assignment 1

Due on 3 October at 11:59 pm

In this assignment we will represent arbitrary precision integers in three ways as follows, where a digit is a number between 0 and 9, and a strictly positive digit is a number between 1 and 9.

- A *native integer* is a value of the HASKELL type `Integer`.
- A *long integer* representation is a list of all the decimal digits of an integer, starting with the least significant digit. For example the list `[1,2,3,4]` is the long integer representation of the number 4321. You may take `[]` (the empty list) as an alternative long representation of the number 0.
- For some $k > 0$, k strictly positive integers x_i, and k strictly positive digits n_i, $1 \leq i \leq k$, a *sparse integer* representation is a list of pairs $[(x_1, n_1), (x_2, n_2), \ldots, (x_k, n_k)]$ that represents the number $\sum_{i=1}^{k} n_i \times 10^{x_i}$. Note in passing that `[]` is the sole possible sparse representation of 0.

We start the assignment by providing conversion functions between the three representations of integers. For this purpose:

1. Define a function `intToLong` that converts a native integer into its long representation.
2. *Use a fold* to define a function `longToInt` that converts a long representation of that integer into a native integer.
3. Modify the Quicksort implementation presented in class so that it sorts lists of pairs `(x,n)` with respect to the first component `x`, thus obtaining the function `longIntSort`.
4. Define a function `longIntExpand` that adds the missing (null) digits to a number in the sparse integer representation. You can assume that the number is already sorted in increasing order of its coefficients. For example

   ```
   longIntExpand [(1,3),(4,2),(6,1)]
   ```

   should evaluate to `[(0,0),(1,3),(2,0),(3,0),(4,2),(5,0),(6,1)]`.
5. *Use a map* to define a function `longIntList` that receives the output of `longIntExpand` and drops the exponents, thus obtaining the long integer representation of the number. For example:

   ```
   longIntList [(0,0),(1,3),(2,0),(3,0),(4,2),(5,0),(6,1)]
   ```

   should evaluate to `[0,3,0,0,2,0,1]`.

6. Put all the functions above together to define the function `sparseToLong` that receives a sparse representation of an integer and evaluates to the long representation of that integer. For example:

   ```
   sparseToLong [(1,3),(4,2),(6,1)]
   ```

   should evaluate to `[0,3,0,0,2,0,1]`.

7. *Use a zip and a filter* to define one more function `longToSparse` that receives the long representation of an integer and returns a sparse representation of that integer. For example:

   ```
   longToSparse [0,3,0,2,0,1]
   ```

   should evaluate to `[(1,3),(3,2),(5,1)]` or any permutation thereof (at your discretion).

We will also implement a few operations on long integer representations. The following functions *must* operate directly on that representation, without any conversion to the `Integer` HASKELL type.

8. Define a function `longAdd n m` that returns the sum of `n` and `m`. For example,

   ```
   longAdd [1,5,3] [9,6,7]
   ```

   should evaluate to `[0,2,1,1]` (since 351 + 769 = 1120).

9. Define a function `LongScale n d` that multiplies the long representation `n` with the single digit `d`. For example,

   ```
   longScale [1,2,3,4] 4
   ```

   should evaluate to `[4,8,2,7,1]`.

10. *Use a fold* to define a function `longMul n m` that returns the product of n and m. For example:

    ```
    longMul [1,2,3] [4,5,6]
    ```

    should evaluate to `[4,3,9,9,0,2]` (since $321 \times 654 = 209934$).

    Hint. It may be easier to start by writing a plain recursive function `longMul` and use that as an inspiration for your folding implementation.

Important notes

The use of programming techniques (folds, maps, and so on) is mandatory whenever specified in the handout. For your convenience these requirements are emphasized throughout the handout.

Provide suitable type definitions for all your top level functions. Use the type `[Integer]` for long integer representations and `[(Integer,Integer)]` for the sparse representation (I know that it can be argued that `Int` would make more sense in two instances, but such a type definition can be realized with fewer headaches). Your type definitions must be as specific as you can make them. In particular just copying the output of `:type` commands into your script will be penalized.

Submission guidelines

Submit a single plain text file that can be loaded in the Haskell *interpreter*. It would be nice if you can submit a literate script (`.lhs` extension), but this is not required. Your script will only be loaded in the interpreter and will not go anywhere near the compiler, so *do not* provide anything having to deal with the Haskell compiler such as a `main` function or I/O operations.

Provide in your script comments that demonstrate and test your functions. Such comments should include copy-and-paste content from the terminal where you ran and tested your program. Note that any part of your work that has not been thus demonstrated will not be marked.

Finally, include at the top of your script a comment with the names and emails of all the collaborators.

Once the script is assembled as explained above submit it by email.

Recall that assignments can be solved in groups[1] (of maximum three students), and a single solution per group should be submitted. Also recall that a penalty of 10% per day will be applied to late submissions.

[1] It is often easier to learn new concepts by working together in a group—you may learn more if you work with smart people. This being said, the purpose of group assignments is that all people in a group should be involved in creating the solution. You can of course choose to sign an assignment you haven't done, but then you risk not learning the material, and this will come back to haunt you during the mid-term examination.