

Administrivia. This homework is based on Chapters 17 and 18 of Thompson. You may work singly or in pairs on this assignment. This homework is due in the bin in CST 3-212 by 5pm on **Friday, April 25**.

General Hints: Be very careful with indenting your code in `do`'s. In particular, if you use `if-then-else`'s then the `then` should be to the right of the `if` and the `else` should be lined up with the `then`. (For example, see Example 3 on page 388 of Thompson.)

The file <http://www.cis.syr.edu/courses/cis252/code/hw11.hs> gives you a starting point for this assignment. The function `inits` in the `List` module may be handy.¹

Warnings: These problems may drive you crazy, but they are easy in retrospect. (Cold comfort, I know.) As explained in class, to test the IO functions you probably want to run Hugs in regular terminal window (*i.e.*, *not* in Emacs). Also, Hugs and GHCi handle terminal I/O a bit differently, with Hugs being the easier to work with for this assignment.

Exercises

Exercise 1. Thompson's Problem 17.22 (page 369). That is, define

```
factorial, fibonacci :: [Integer]
```

where `factorial` is the infinite list of all factorials (e.g., `[1,1,2,6,24,120,...]`) and `fibonacci`

is the infinite list of all Fibonacci numbers (e.g., `[0,1,1,2,3,5,8,13,21,...]`). Here is an example of how to go about defining such monsters. First, let us define (as Thompson does on page 364)

```
ones :: [Integer]
ones = 1:ones
```

which is an infinite list of 1s. Then

```
ints :: [Integer]
ints = 0:zipWith (+) ints ones
```

is a (recursive) definition of the infinite list of non-negative integers, *i.e.*,

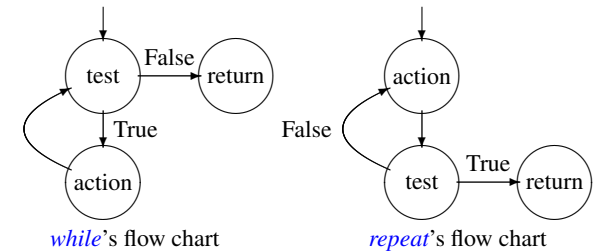
```
0:zipWith (+) ints ones
      0 : 1 : 2 : 3 : ...
= 0 : +  +  +  +
      1 : 1 : 1 : 1 : ...
= 0 : 1 : 2 : 3 : 4 : ...
= [0,1,2,3,4...]
```

For the `factorial` problem you can pull a similar trick with `(*)` in place of `(+)` and a different list in place of `ones`. For `fibonacci`, note that

```
0 1 1 2 3 5 8 13 ...
+ 1 1 2 3 5 8 13 21 ...
-----
1 2 3 5 8 13 21 34 ...
```

Exercise 2. Thompson's Problem 17.24 (page 370). For example: `(runningsums ones)` yields `[1,2,3,4,5,6 ..]`.

Exercise 3. Thompson's Problem 18.7 (page 395). In terms of flow charts, here is how `repeat` differs from `while`.



Exercise 4. Thompson's Problems 18.8 and 18.9 (page 395). For 18.9, just return the sum of the *n* numbers read (so we don't have to worry about division). Also, if *n*, the initial number read, is negative or zero, then return 0 as the sum.

What to hand in. Hand in your Haskell code and a transcript of the hugs session in which you run each of the tests. *Don't forget:* For each of the problems, provide: (i) a contract, (ii) a purpose statement, (iii) examples, (iv) the definition, and (v) tests. **N.B.** The grade for each problem will be about 40% correctness of your definition, 40% completeness of your tests, and 20% for following the design recipe, the quality of your answer, etc.

¹For info on `inits`, see <http://haskell.org/ghc/docs/latest/html/libraries/base/Data-List.html#v%3Ainits>; for more on factorials, see <http://en.wikipedia.org/wiki/Factorial>; and more on Fibonacci numbers, see http://en.wikipedia.org/wiki/Fibonacci_numbers.