15-150: Principles of Functional Programming

Quiz 5 Practice

Summer 2022

Name: _____

Andrew ID: _____

Circle the lab you are in:

A	Julia and Yosef	GHC 5207
B	Juhi, Nicole, and Susan	GHC 5208
C	Nancy and Shengchao	GHC 5210
D	Thea and Cindy	GHC 4303
E	Jacky, Jonathan, and Sonya	WEH 7500

Question	Points	Score
Modules Practice	0	
Sequences Practice	0	
Total:	0	

Answer the questions in the spaces provided. You will have the full 80 minutes of class time to complete this quiz. Good luck!! Y'all got this.

Question 1: Modules Practice

(a) (0 points) Define a structure `Foo` that can ascribe to `sig datatype t = Bar val baz : t -> t end`, or state that none exists.

Solution:

```
structure Foo =
struct
    datatype t = Bar
    fun baz x = x
end
```

(b) (0 points) Define a structure `Awooga` that *cannot* ascribe to `sig end`, or state that none exists.

Solution: none exists

(c) (0 points) Rational numbers can be constructed as the rational closure over integers by considering the ratio of pairs of integers. To add with such pairs-as-rationals, we use the fact that $\frac{a}{b} + \frac{c}{d} = \frac{ad+bc}{bd}$ to deduce $(a, b) + (c, d) = (ad + bc, bd)$, as our pairs simply stand for these ratios.

```
signature NUM =
sig
     type t
     val plus : t -> t -> t
     val times : t -> t -> t
end
```

Given the above signature `NUM` for *numbers*, define a functor `Rationalize` that would transform an integer number module into a rational number module[1]

Solution:

```
functor Rationalize (N:NUM) : NUM =
struct

     type t = N.t * N.t

     fun plus (a,b) (c,d) =
         (N.plus (N.times a d) (N.times b c),
         N.times b d)

     fun times (a,b) (c,d) =
         (N.times a c, N.times b d)

end
```

(d) (0 points) If you ascribe the above functor `:>NUM`, what problem might a client using that functor run into?

Solution: There is no way to them to create something of the type `t` in the first place because it would be left abstract.

[1] Really it would create the rational closure of any `NUM` module, integer or not.

Question 2: Sequences Practice

Consider the following code[2]

```
fun listTab f n =
    if n = 0
    then []
    else (f n) :: (listTab f (n-1))

fun id x = x
```

(a) (0 points) Give a tight asymptotic upper band on the *work* of applying the function `Seq.tabulate (listTab id)` to the integer `n`.

Solution: $O(n^2)$

(b) (0 points) Give a tight asymptotic upper band on the *span* of applying the function `Seq.tabulate (listTab id)` to the integer `n`.

Solution: $O(n)$

(c) (0 points) Give a tight asymptotic upper band on the *work* of applying the function `listTab (Seq.tabulate id)` to the integer `n`.

Solution: $O(n^2)$

(d) (0 points) Give a tight asymptotic upper band on the *span* of applying the function `listTab (Seq.tabulate id)` to the integer `n`.

Solution: $O(n)$

(e) (0 points) **True** or **False** : Applying `Seq.reduce op@ []` to `s` always has $O(log(|s|))$ span.

Solution: false (note `op@` does not have constant span)

[2]The `List` module actually already defines a function called `tabulate`, but we don't expect you to be familiar with it.

(f) (0 points) Define a function `thin : 'a seq -> 'a seq` that copies its input list but skips elements at even indices, keeping only the odd ones. For example, `thin <15,1,5,0>` should map to `<1,0>`. Your implementation should have constant span.

Solution:

```
fun thin s =
    Seq.tabulate
        (fn i => Seq.nth s (i * 2 + 1))
        ((Seq.length s) div 2)
```