# CS 254: Introduction to Functional Programming • Sample Midterm <br> Fritz Ruehr • Willamette University 

Name:

## Instructions

This is a sample exam which shows what typical questions look like. A real exam would have about 100 points; for example, perhaps 15 T/F questions = 15 pts; 10 short answer = 50 points; 3 longer answer = 35 points. An hour and a half exam might run 6-7 pages total, but note that a lot of the space is for answers.

## A. Basic knowledge-True or False (2 points each)

1. In Haskell, strings of text are presented as lists of individual characters.
2. In Haskell, numbers of type Int are presented as lists of digit characters.
3. Variables defined by local declarations (using where) are available throughout the whole module. $\qquad$

Typically 10-15 questions worth 1-2 points each in the "True or False" section.

## B. Multiple choice and short answer questions (5 points each)

1. One danger of recursion (i.e., when a variable or function is used in its own definition) is that it might cause evaluation to go on forever. How does Haskell handle this possibility?
(a) Haskell uses its type system to reject such definitions as "nonsense"
(b) definitions that may evaluate forever are not allowed: they give an error when you load the file
(c) Haskell will keep trying to evaluate as long as the evaluation goes
(d) when WinHugs sees that evaluation is taking forever, it stops and asks if you want to continue
[choose the single best answer]
2. Say that $f$ is a function which takes a list of characters and an (arbitrarily large) integer, as two separate arguments, and returns a list of Booleans; how would you write this fact (that $f$ has this type) in Haskell?

Typically 8-12 questions worth 4-6 points each in the "Multiple choice/short answer" section.

## C. Longer answer questions (10 points each)

1. Reduce the following expression according to the definitions given (as on the written homework). Show all the intermediate steps and do your best to get an answer. But if the evaluation will go on forever, stop evaluating (and say so).
```
\(g x=x+2\)
\(f a b=a+g b\)
\(h \mathrm{x}=\) if \(\mathrm{x}>1\) then f x 3 else \(h(x-3)\)
```

h ( 9 3)
... [lots of space for answer here] ...

Typically 3-5 questions worth 8-12 points each in the "Longer answer" section.

I also supply an appendix with types and descriptions of functions from the Haskell standard Prelude.

## Appendix-some useful definitions from the Haskell Prelude

(we will assume that the "training wheels are on", so that numeric types default to Integer)

```
(+) :: Integer -> Integer -> Integer -- adds numbers
(>) :: Integer -> Integer -> Bool -- "is greater than"
(<) :: Integer -> Integer -> Bool -- "is less than"
(==) :: Integer -> Integer -> Bool -- "is equal to"
[] :: [a]
(:) :: a -> [a] -> [a]
(&&) :: Bool -> Bool -> Bool -- logical and (both must be True)
(|| :: Bool -> Bool -> Bool -- or (one must be True)
length :: [a] -> Integer
-- length of a list
reverse :: [a] -> [a]
map :: (a->b) -> [a]-> [b]
-- nil (the empty list)
-- cons (builds lists)
-- returns reversed list
-- apply function to every
-- element of a list
```

