1. 10 points: A grammar is a set of production rules for strings in a formal language. Consider the following grammar.

<stmt> ::= if ‘(‘ <expr> ‘)’ <stmt>

 | if ‘(‘ <expr> ‘)’ <stmt> else <stmt>

1. What is the definition of an ambiguous grammar?
2. Is this grammar ambiguous?
3. Prove that this grammar is or is not ambiguous using parse trees.
4. How does Haskell address the if-else ambiguity?
5. 10 points: Give the types of the following:
6. True
7. ‘a’
8. 4 == 5
9. [1,6,3,9]
10. “This test is a piece of cake!”
11. (1, ‘a’)
12. factorial n = product [1..n]
13. circumference r = 2 \* pi \* r
14. fst (x, \_) = x
15. max a b = if a > b then a else b
16. 4 points: Using the :: operator, specify 20 as:
17. A bounded Integer
18. An unbounded Integer
19. A floating point number
20. A double precision floating point number.
21. 3 points: Define a function in Haskell called divTwo that takes a number and divides it by 2.

divTwo :: (Integral a) => a -> a

1. 3 points: Define a function in Haskell called tripleOne that takes a number, multiplies it by 3 and adds 1 to the result.

tripleOne :: (Integral a) => a -> a

1. 5 points: Using an if statement, define a function in Haskell that takes a number and creates a Collatz chain. In other words, if the number is even then divide the number by two. If the number is odd then multiply it by 3 and add 1.

Example: chain 10

[10,5,16,8,4,2,1]

chain :: (Integral a) => a -> [a]

1. 5 points: Rewrite chain as above using guards instead of an if statement.
2. 5 points: Write a command to find the Collatz chains for all numbers between 1 and 100. Use your chain function.
3. 5 points: Use list comprehensions to convert all uppercase letters to lowercase letters. You may use toLower from Data.Char **IF YOU IMPORT IT FIRST**.
4. 10 points: Using recursion, define the replicate function.

replicate’ :: (Num i, Ord i) => i -> a -> [a]

1. 5 points: What is the definition of a curried function?
2. 5 points: Give an example of a section (partially applied function). Use + or \*.
3. 5 points: Rewrite divTwo using a partially applied function.
4. 5 points: Write a function to find the factors of a number using list comprehension

factors :: (Integral a) => a -> [a]

1. 5 points: Write a function to check and see if a number is prime. You may use the factors function defined above.

isPrime :: (Integral a) => a -> Bool

1. 5 points: Write a command to find the largest prime number less than 1,000,000.

1. 5 points: Define a function called factorial using folds.

factorial :: (Num a, Enum a) => a -> a

1. 5 points: What is the difference between type and data in Haskell
2. 10 points: You have an array of 5 different data models that take in a number n and return a prediction. Using function application ($) and map, write a function to get predictions from your five models.

let models = [model1, model 2, model3, model4, model5]

getPredictions :: a -> [a -> b] -> [b]

1. 20 points: Consider the following definition of a Binary Search Tree in Haskell.

data Tree a = EmptyTree | Node a (Tree a) (Tree a) deriving (Show, Read, Eq)

*--Function to insert 1 node into an empty tree*

singleton :: a -> Tree a

singleton x = Node x EmptyTree EmptyTree

*--Function to insert a new node into a tree*

treeInsert :: (Ord a) => a -> Tree a -> Tree a

*--Base case is inserting a node into an empty tree*

*-- which is the same as creating a singleton tree*

treeInsert x EmptyTree = singleton x

treeInsert x (Node a left right)

 | x == a = Node x left right

 | x < a = Node a (treeInsert x left) right

 | x > a = Node a left (treeInsert x right)

*--Function to build a tree given a list.*

buildTree :: (Ord a) => [a] -> Tree a

buildTree xs = foldr treeInsert EmptyTree xs

Define a function called treeElem that will tell us if an item is found in our search tree.

treeElem :: (Ord a) => a -> Tree a -> Bool

1. 10 points: Define a new data type for the days of the week. You should be able to compare equality, compare order (i.e. Monday < Tuesday), convert them to and from strings (for printing etc.) and create lists of days.

data Day =