Programming HW 9

CS 169 Spring 19, Linnell

Scala Pattern Matching and Inheritance

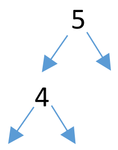
Due Fri 6/7 in class

Submission instructions at the end

1. (5 points) Read the blog post here: https://www.smashingmagazine.com/2014/07/dont-be-scared-of-functional-programming/
   (you only need to read up to "Let's Get Real")
   Write a 100-150 word summary of this article; you don't have to rehash the whole running example, but please provide at least three ideas that you found interesting in the article. If you want to read something a little deeper, here's another blog post you may find interesting: http://www.ibm.com/developerworks/library/j-ft20/

2. (10 points) Add the following functions to the `BSTree` class we discussed Friday 5/31

   `def depth:Int` This function should return the number of values traversed between the root and the deepest leaf (inclusive). That is, the tree below has depth 2. You may NOT use pattern matching (the match/case syntax).

   `def exactsubtree(that:BSTree):Boolean` This function should return true if `this` appears in the exact same structure as a subtree of `that` (down to the Leaf references). You MUST use pattern matching.

2. (5 points) Write a function `def findlast(xs:List[Int], x:Int):Int` that returns the index of the last time the element x appears in the list xs. Return -1 if the element does not appear in the list. For this function, you MUST use pattern matching, and you may NOT use any built-in list functions. I will give 2 points extra credit if you do it without a helper function.

4. (5 points). You may wish to wait until after class Monday to do this problem.
   a. Implement the class and functions below as described in the comments. You will build on your Set class from last week.

```
class singletonSet(elem:Int) extends Set(/*fill this in*/){
//This class creates a set with only one element, elem.
//Now that we have a set with only one element, we can do better with
//forall.  Implement this functions to work for ALL integers,
//not just those in the range -1000 to 1000
override def forall(p:Int=>Boolean):Boolean =
}
```

b. Define a function `subs[T<:Set](s:T, ss:List[T]):List[T]` that returns a list containing all of the sets in `ss` that are subsets of `s`. You must use pattern matching. +1 point extra credit if your function calls `s's forall function`.

5. 5 points extra credit. Create a new class `groupoid[T<:Set](s:T, op:(Int, Int)=>Int)`. A groupoid is a mathematical structure consisting of a set of elements (we'll restrict this to subsets of the Integers for our purposes) and an operation on those elements. However, in order to be a groupoid, the set must be closed under the operation; that is, if the operation is applied to any two elements of the set, the result of that operation must also be in the set. Your class should have a single member, a `val closed` which is set equal to a call to `s's forall` function.

   For instance, the set of all integers is closed under both addition and multiplication, and the set of integers mod 5 ({0, 1, 2, 3, 4}) is closed under addition mod 5 (f(x) = (x+y)%5). You must also include a main function which creates groupoids corresponding to these three cases and prints the value of their `closed` member.

   Note: This problem is quite difficult.

Submission instructions: You will print out your code for each problem, stapling together multiple sheets (there will be deductions for unstapled homework!). Turn in the hardcopy at the beginning of class. You will ALSO submit your code as .txt files as an attachment, to cs169@math.scu.edu The subject line of the email should be "CS169 HW9 YourLastName YourIDNumber "