CS 2510 Exam 2 – Fall 2014

Instructions

This exam takes data definitions that we have seen in class and asks you to extend them in various ways. *Read the problems carefully* before diving into the code, so you understand how the problems relate to each other.

Almost every method on this exam can be written in a few short lines of code. If you find yourself wanting to write a lot of code, stop and reconsider your approach.

- You will submit your completed exam via WebCAT. You will not be graded on minor details of WebCAT style, but you will also not receive any feedback from WebCAT. You will be allowed ***exactly one submission*** through WebCAT, and we will not be providing tests or hints for you to check your work; that is your task.
- This is a take-home exam and you may use your notes or homeworks for help, but it is still a solo exam. Any evidence of collaboration or other cheating will result in a grade of 0.
- We have provided skeleton code for you to complete. You will not need to modify any of the method implementations we have given you, but do be sure to fill in all `???` comments.
- You may use any of the Java techniques we have learned so far to solve the problems; not all of them will be needed.
- You may use Eclipse to help you design and code your solutions. Accordingly, you should write proper Java code: use `t.checkExpect(c, e);` in an `Examples` class for your tests.
- Be sure to thoroughly test your code. The algorithms on this exam may have subtle edge cases; you have plenty of time to find them and figure them out, so do so!

Good luck!

Problem 1 We have seen the interface for `IList<T>` many times; here is an excerpt:

8 POINTS

```
interface IList<T> {
  int length();
  <U> IList<U> map(IFunc<T, U> func);
}
```

Until now, we have only implemented lists using `Cons<T>` and `Empty<T>`. If we wanted to manipulate lists, by appending two lists together or adding an element to the end, we've had to define recursive methods that achieve that for us. This problem (and the next) tries another approach.

Define two new classes that implement this interface:

- `Snoc<T>` contains a list and an item and represents adding the item to the *end* of the list ("snoc" is "cons" spelled backwards, and is "like a cons but backwards")
- `Append<T>` contains two lists and represents appending the second to the end of the first.

12 POINTS

Problem 2 Add two new methods to the `IList` interface and implement them on the four classes that implement the interface:

- `IList<T> reverse()` constructs the reverse of this list
- `IList<T> normalize()` "simplifies" a list that is constructed from any of `Cons<T>`, `Empty<T>`, `Snoc<T>` or `Append<T>`, and produces a new list that uses only `Cons<T>` and `Empty<T>`.

Problem 3 For this problem, we are going to use `ILoString`, `ConsLoString`, and `MtLoString`. The methods `length` and `contains` have been defined for you. Suppose I construct the following examples:

6 POINTS

```
class ExamplesLists {
   ILoString myList;
   void initData() {
      ConsLoString c1 = new ConsLoString("Groundhogs' Day", new MtLoString());
      ConsLoString c2 = new ConsLoString("New Year's Day", c1);
      ConsLoString c3 = new ConsLoString("Thanksgiving", c2);
      ConsLoString c4 = new ConsLoString("Halloween", c3);
      ConsLoString c5 = c1;
      ConsLoString c6 = new ConsLoString("Columbus Day", c4);
      c5.rest = c1;
      this.myList = c6;
   }
}
```

Define a test method `void brokenTest(Tester t) { ... }` containing *just one test* about `myList` that neither passes nor fails, but rather crashes with a `StackOverflow` error.

8 POINTS

Problem 4 To detect the problem in the previous question, we need an algorithm that can detect whether a given `ILoString` ends in an `MtLoString` or not. We have added to the `ConsLoString` class a boolean field called `seenAlready`, and initialized it to `false`. Design a method `boolean wellFormedList()` on `ILoString` that returns `true` if the list ends in a `MtLoString`. You will need to use (and modify) the `seenAlready` field, and also return a value from this method. You should not need any helper methods.