University of Virginia 9 September 2011
cs1120: Introduction of Computing
Explorations in Language, Logic, and Machines

Class 8: Recursive List Procedures

Assignments Due

- **By now you should have read the course book through Section 5.4**
- **Wednesday, 14 September: Problem Set 2** (Problem Set 2 is much longer and harder than Problem Set 1. Don't wait to get going on this!)
- **Friday, 16 September:** Read *The Information* Chapters 4 and 10.
- **Monday, 19 September:** Read course book through the end of Chapter 6.

Upcoming Help Schedule (all office hours are now in Rice Hall)

- Sunday: 1-6pm (Valerie/Joseph/Kristina, Rice 1st) – it would definitely be a good idea to take advantage of this for making progress on PS2!

Book Exercises

Yes, there are solutions to the exercises in the book available, at least through the end of Chapter 4. They are linked from http://www.computingbook.org. I will try to make solutions for later chapters available soon, but if there is a problem you want a solution for that isn't there yet, please ask me for it. It is even better if you can send me a candidate solution!

Lists

A *List* is either: (1) null, or (2) a Pair where the second part is a *List*.

Defining List Procedures:

1. **Be *very* optimistic!** Since lists themselves are recursive data structures, most problems involving lists can be solved with recursive procedures.
2. Think of the simplest version of the problem, something you can already solve. This is the base case. For lists, this is usually when the list is **null**.
3. Consider how you would solve the problem using the result for a slightly smaller version of the problem. This is the recursive case. For lists, the smaller version of the problem is usually the **cdr** of the list.

Define a procedure, **is-list?**, that takes as input any value and evaluates to true if the input value is a List, and false otherwise.

1. Define a procedure, **list-length**, that takes as input a list and outputs the number of elements in that list (without using the built-in **length** procedure).

2. Define a procedure, **find-closest**, that takes as input a list and a target number, and outputs the element in that list that is closest to the target number. (Note that this description is under-specified. It is up to you to define find-closest in a way that makes it clear what it should do for *all* possible inputs.)

3. (Note: this is a question on Problem Set 2) Define a procedure, **hamming-distance**, that takes as inputs two lists, and outputs the number of positions where the elements in the list are different. For example, **(hamming-distance (list 1 0 1 1) (list (1 1 1 1))** should evaluate to 1 and **(hamming-distance (list 1 1 2 0) (list 1 5 0))** should evaluate to 3.