Class 8: Git Yer Pointers Here

Action Items

PS2 is due Monday, September 30. You should be at least up to Problem 4 by now.

Before Thursday's class, read either the paper or slides for *MapReduce: Simplified Data Processing on Large Clusters* (Jeffrey Dean and Sanjay Ghemawat, Symposium on Operating System Design and Implementation, December, 2004).

Lists in Rust

For this exercise, we'll implement a linked list data structure using Rust. (Note that the Rust library provides a vector type, so it is unlikely you would want to use this linked list code, but it is a worthwhile exercise to explore some issues in programming in Rust.)

Version 1: Automatically Managed

```
struct Node { head : int, tail : Option<@Node> }
type List = Option<@Node> ;

impl ToStr for List {
    fn to_str(&self) -> ~str {
        fn elements_to_str(n: @Node) -> ~str {
            match (n.tail) {
                None => fmt!("%?", n.head),
                Some(tail) => fmt!("%?, %s", n.head, elements_to_str(tail))
            }
        }

        match(*self) {
            None => ~"Null",
            Some(n) => fmt!("[%s]", elements_to_str(n))
} } } // don't do this unless you are printing 60 copies of your code

fn main() {
    let lst : List =
        Some(@Node{head: 1, tail:
          Some(@Node{head : 2, tail:
            Some(@Node{head: 3, tail: None})})});
    println(fmt!("%s", lst.to_str()));
}
```

Version 2: Mutable List with Mapping

```
struct Node {
    head : int,
    tail : Option<@mut Node>
}

type List = Option<@mut Node> ;

trait Map {
    fn mapr(&self, &fn(int) -> int);
}

impl Map for List {
    // Thanks to dbaupp for figuring out why this breaks when called "map"!
    fn mapr(&self, f: &fn(int) -> int) {
        let mut current = *self;
        loop {
            match(current) {
                None => break,
                Some(node) => { node.head = f(node.head); current = node.tail },
            }
        }
    }
}

// ToStr removed to save space

fn main() {
    let lst : List =
       Some(@mut Node{head: 1, tail:
          Some(@mut Node{head : 2, tail:
             Some(@mut Node{head: 3, tail: None})})});
    lst.mapr(|x: int| { x + 1 });
    lst.mapr(|x: int| { x * x });
    lst.mapr(|x: int| { if x % 2 == 0 { x } else { -x }});
    println(lst.to_str());
}
```

Exercise 1. Add an `append(&self, List)` method that appends two lists. (You should think carefully about what semantics `append` should have before implementing it.)

Exercise 2. Add an `filtr(&self, &fn(int) -> bool)` method that removes elements for which a predicate is not true from a list.