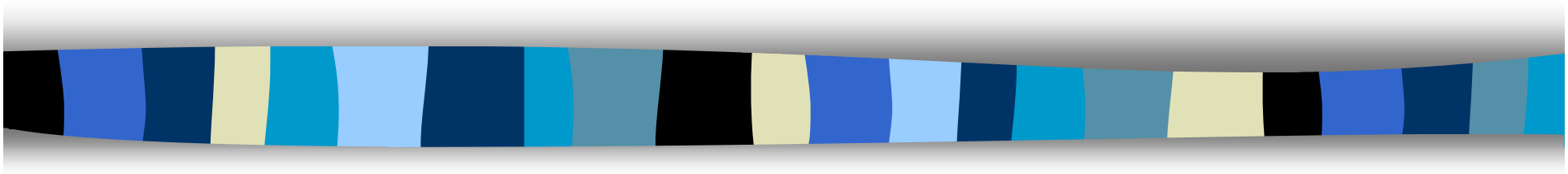
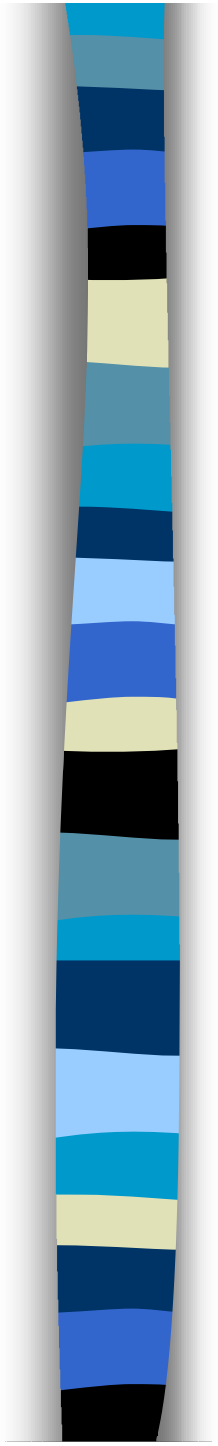


Design Patterns in CS2?

Robert "Corky" Cartwright

Rice University



Rice Perspective on CS1/CS2

- Good program design is *data-directed*.
- All program components process some form of data; *the structure of data should determine the structure of the program that processes it*.
- Fundamental principles of program design are language independent, but they are most easily explained and learned in context of (mostly) functional programming.

Design Principles in CS1

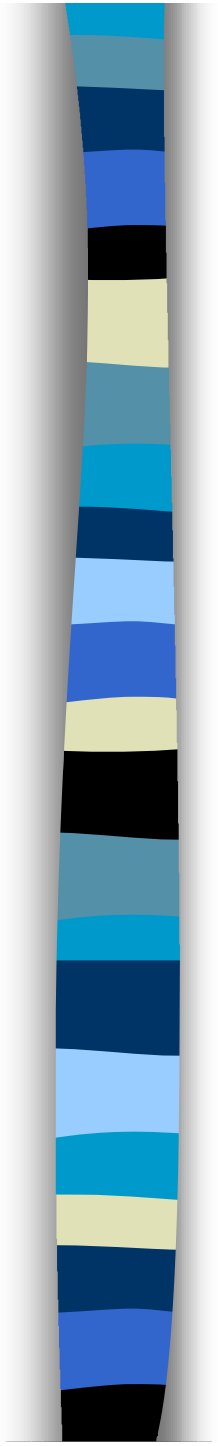
- Define the data first.
- Most data definitions are inductive, *e.g.*

  A PhoneDirectory is either:

  - Empty, or
  - Cons(e, pd)

  where e is an Entry and pd is a PhoneDirectory.

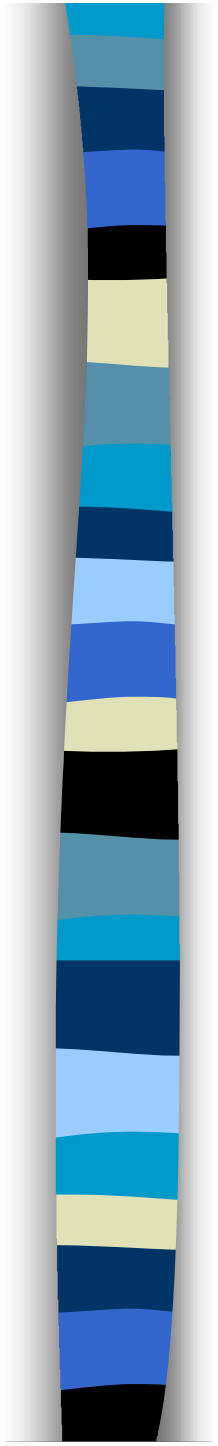
  An Entry is a pair Entry(name,number) where name and number are strings.



More CS1 Design Principles

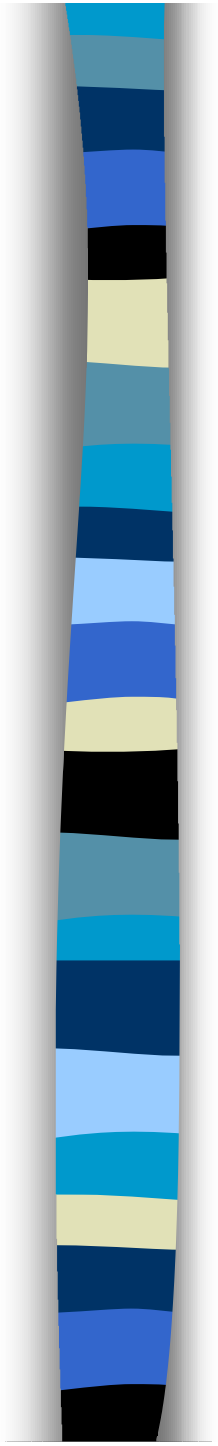
- Programs are functions that map input data to output data.
- Given a data definition, there is a program pattern that processes it, *e.g.*

```
function lookup(PhoneDirectory pd, …)
  case pd of
     Empty:  …
     Cons(e,pdt): …  lookup(pdt,...) …
```



Mission of CS2 at Rice

- Translate good functional design to good OO design via design patterns.
- Basic data structures and algorithms.
- Intuitive understanding of complexity of common algorithms.
- Introduction to concurrency?



OO Design Patterns

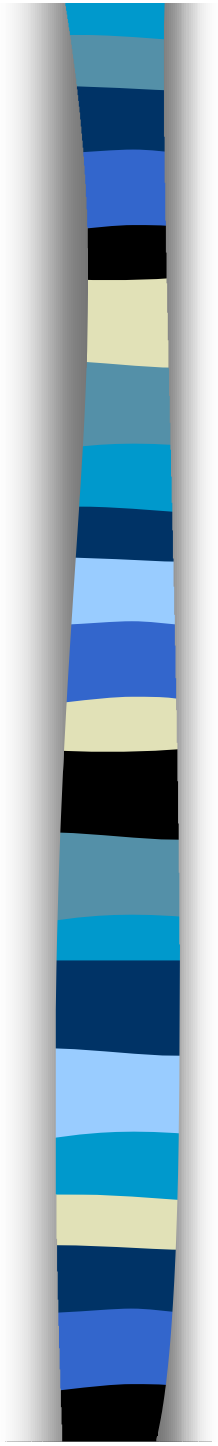
- Inductive data definition →

  Composite pattern
- Natural recursion scheme →

  Interpreter pattern
- Abstracting common patterns →

  Command and *Singleton* patterns
- Extensibility → *Visitor* pattern →
- Encapsulation of mutable state

  Iterator pattern



Example

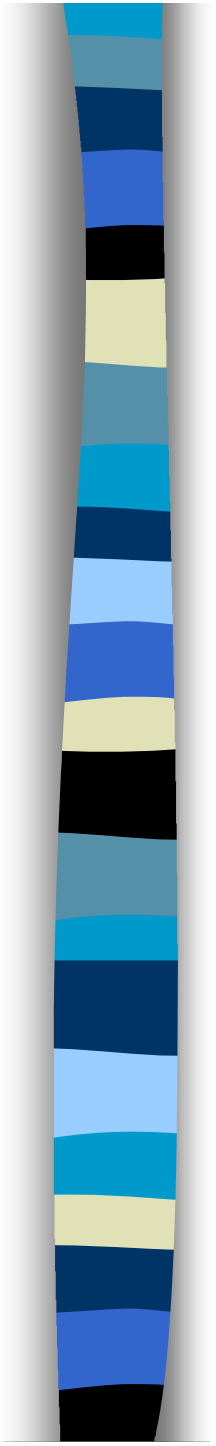
```
datatype Entry(String name, String phone)
datatype PhoneDirectory =
    Empty | Cons(Entry,PhoneDirectory)

String function lookup(PhoneDirectory pd, String name)
  case pd of
      Empty: return null
      Cons(first,pdRest):
         if (first.name = name) return first.phone
         else return lookup(pdRest, name)
```

becomes . . .

Corresponding OO Code

```
class Entry {
  String name, phone;
  Entry(String n, String p) { name = n; phone = p; }
}
abstract class PhoneDirectory {
  abstract String lookup(String name);
}
class Empty extends PhoneDirectory {
  String lookup(String name) { return null; }
}
class Cons extends PhoneDirectory {
  Entry first;
  PhoneDirectory rest;
  Cons(Entry f, PhoneDirectory r) { first = f; rest = r; }
  String lookup(String name) {
     if (name.equals(first.name)) return first.phone;
     else return rest.lookup(name);
  }
}
```



What Rice CS2 is not

- Anatomy of Java libraries
- Training in GUI building
- Programming in the large
- Algorithms encyclopedia