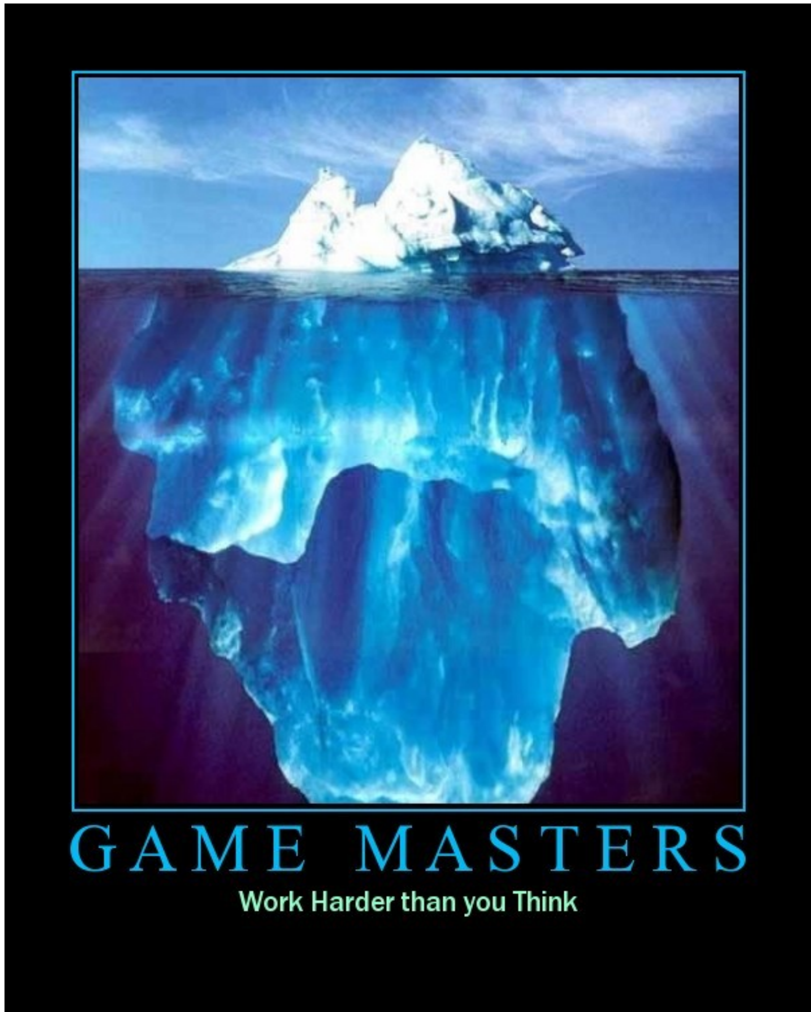
Functional Programming

Introduction To Cool

Cunning Plan

- ML Functional Programming
  - Fold
  - Sorting
- Cool Overview
  - Syntax
  - Objects
  - Methods
  - Types



CS 4501 – Compilers Practicum

- **Mondays 5:00 to 6:30, Olsson 011**
  - Typically until 6:00
- To be enrolled in CS 4501 (Compilers Practicum) you *must* be able to attend its listed lecture time.
- First Meeting: Next Week!
  - Monday, January 30th

PS1c Correct Answer Statistics

- Language choice, as of noon today ...

– Python	28 (+5 partially correct)
– Ruby	22 (+2 partially correct)
– C	8
– OCaml	3
– Cool	0

• Students Submitting > 0 Times:	40
• Students Taking Class For Credit:	46+

- This assignment was originally named “PS0”.

Undergraduate Research

- Reminder: you can help Alex Landau, get +5 points of extra credit on PS1, be entered in a drawing for a $50 Amazon gift card, and advance human knowledge by completing
  - **http://church.cs.virginia.edu/~zpf5a/code-quality/**
  - by Midnight, Sunday January 29th
- An undergraduate will be first author on this paper.

This is my final day

- ... as your ... *companion* ... through Ocaml and Cool. After this we start the interpreter project.

One-Slide Summary

- Functions and type inference are **polymorphic** and operate on more than one type (e.g., List.length works on int lists and string lists).
- **fold** is a powerful higher-order function (like a swiss-army knife or duct tape).
- **Cool** is a Java-like language with classes, methods, private fields, and inheritance.

Pattern Matching (Error?)

- Simplifies Code (eliminates ifs, accessors)

```
type btree =   (* binary tree of strings *)
  | Node of btree * string * btree
  | Leaf of string
let rec height tree = match tree with
  | Leaf _ -> 1
  | Node(x,_,y) -> 1 + max (height x) (height y)
let rec mem tree elt = match tree with
  | Leaf str | Node(_,str,_) -> str = elt
  | Node(x,_,y) -> mem x elt || mem y elt
```

Pattern Matching (Error?)

- Simplifies Code (eliminates ifs, accessors)

```
type btree =   (* binary tree of strings *)
  | Node of btree * string * btree
  | Leaf of string
let rec height tree = match tree with
  | Leaf _ -> 1
  | Node(x,_,y) -> 1 + max (height x) (height y)
let rec mem tree elt = match tree with
  | Leaf str | Node(_,str,_) -> str = elt
  | Node(x,_,y) -> mem x elt || mem y elt
```

bug?

Pattern Matching (Error!)

- Simplifies Code (eliminates ifs, accessors)

```
type btree =    (* binary tree of strings *)
  | Node of btree * string * btree
  | Leaf of string
let rec bad tree elt = match tree with
  | Leaf str | Node(_,str,_) -> str = elt
  | Node(x,_,y) -> bad x elt || bad y elt
let rec mem tree elt = match tree with
 | Leaf str | Node(_,str,_) when str = elt -> true
 | Node(x,_,y) -> mem x elt || mem y elt
```

Recall: Polymorphism

- Functions and type inference are polymorphic
  - Operate on more than one type
  - let rec length x = match x with
  - | [] -> 0
  - | hd :: tl -> 1 + length tl
  - val length : α list -> int
  - length [1;2;3] = 3
  - length [“algol”; ”smalltalk”; ”ml”] = 3
  - length [1 ; “algol”] = type error!

α means “any one type”

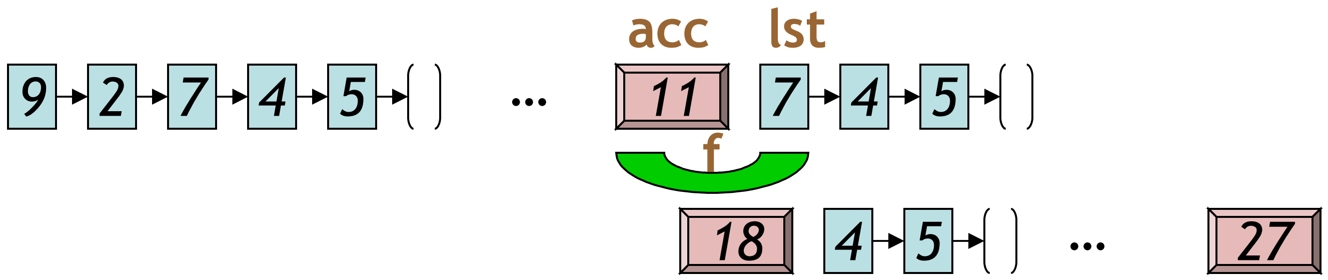
Recall: Higher-Order Functions

- Function are first-class values
  - Can be used whenever a value is expected
  - Notably, can be passed around
  - Closure captures the environment
  - let rec map f lst = match lst with
  - | [] -> []
  - | hd :: tl -> f hd :: map f tl
  - val map : (α -> β) -> α list -> β list
  - let offset = 10 in
  - let myfun x = x + offset in
  - val myfun : int -> int
  - map myfun [1;8;22] = [11;18;32]
- Extremely powerful programming technique
  - General iterators
  - Implement abstraction

f is itself a function!

Recall: Fold

- The **fold** operator comes from Recursion Theory (Kleene, 1952)
  - let rec **fold** f acc lst = match lst with
  - | [] -> acc
  - | hd :: tl -> fold f (f acc hd) tl
  - **val fold : (α -> β -> α) -> α -> β list -> α**
- Imagine we're summing a list (f = addition):



Fold Is Powerful!

- Let's build things out of Fold!
  - **length** lst = fold (fun acc elt -> acc + 1) 0 lst
  - **sum** lst = fold (fun acc elt -> acc + elt) 0 lst
  - **product** lst=fold (fun acc elt -> acc * elt) 1 lst
  - **and** lst = fold (fun acc elt -> acc & elt) true lst

Map From Fold

- let **map** myfun lst =
- fold (fun acc elt -> (myfun elt) :: acc) [] lst
  - Types: **(myfun : α -> β)**
  - Types: **(lst : α list)**
  - Types: **(acc : β list)**
  - Types: **(elt : α)**
- How do we do **sort**?
  - **(sort : (α -> α -> bool) -> α list -> α list)**

Do nothing which is of no use.
- **Miyamoto Musashi**, 1584-1645

Insertion Sort in OCaml

```
let rec insert_sort cmp lst =
  match lst with
  | [] -> []
  | hd :: tl -> insert cmp hd (insert_sort cmp tl)
and insert cmp elt lst =
  match lst with
  | [] -> [elt]
  | hd :: tl when cmp hd elt ->
         hd :: (insert cmp elt tl)
  | _ -> elt :: lst
```

What's the worst case running time?

Sorting Examples

- **langs = [“fortran”; “algol”; “c”]**
- **courses = [216; 333; 415]**
- <u>sort</u> (fun a b -> a < b) langs
  - [“algol”; “c”; “fortran”]
- <u>sort</u> (fun a b -> a > b) langs
  - [“fortran”; “c”; “algol”]
- <u>sort</u> (fun a b -> strlen a < strlen b) langs
  - [“c”; “algol”; “fortran”]
- <u>sort</u> (fun a b -> match is_odd a, is_odd b with

```
    | true, false -> true (* odd numbers first *)
    | false, true -> false (* even numbers last *)
    | _, _ -> a < b (* otherwise ascending *)) courses
```

  - [333 ; 415 ; 216]

Java uses Inner Classes for this.

Partial Application and Currying

- let myadd x y = x + y
- **val myadd : int -> int -> int**
- myadd 3 5 = 8
- let addtwo = myadd 2
  - How do we know what this means? We use referential transparency! Basically, just substitute it in.
- **val addtwo : int -> int**
- addtwo 77 = 79
- Currying: "if you fix some arguments, you get a function of the remaining arguments"

- ML, Python and Ruby all support functional programming
  - closures, anonymous functions, etc.
- ML has strong static typing and type inference (as in this lecture)
- Ruby and Python have “strong” dynamic typing (or duck typing)
- All three combine OO and Functional
  - … although it is rare to use both.

Modern Languages

- This is the most widely-spoken first language in the European Union. It is the third-most taught foreign language in the English-speaking world, after French and Spanish. Its word order is a bit more relaxed than English (since nouns are inflected to indicate their cases, as in Latin) – famously, verbs often appear at the very end of a subordinate clause. The language's famous "Storm and Stress" movement produced classics such as *Faust*.

Natural Languages

- This linguist and cognitive scientist is famous for, among other things, the sentence "**Colorless green ideas sleep furiously**". Introduced in his 1957 work *Syntactic Structures*, the sentence is correct but has not understandable meaning, thus demonstrating the distinction between syntax and semantics. Compare **"Time flies like an arrow; fruit flies like a banana."** which illustrates garden path syntactic ambiguity.

Cool Overview

- Classroom Object-Oriented Language
- Design to
  - Be implementable in one semester
  - Give a taste of implementing modern features
    - Abstraction
    - Static Typing
    - Inheritance
    - Dynamic Dispatch
    - And more ...
  - But many “grungy” things are left out

A Simple Example

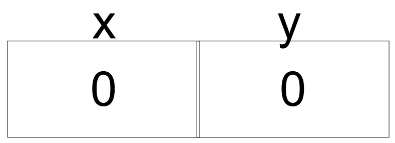
```
class Point {
        x : Int <- 0;
        y : Int <- 0;
};
```

- Cool programs are sets of class definitions
  - A special **Main** class with a special method **main**
  - Like Java
- **class** = a collection of fields and methods
- Instances of a class are **objects**

Cool Objects

```
class Point {
    x : Int <- 0;
    y : Int; (* use default value *)
};
```

- The expression "new Point" creates a new object of class Point
- An object can be thought of as a record with a slot for each attribute (= field)



Methods

```
class Point {
     x : Int <- 0;
     y : Int <- 0;
     movePoint(newx : Int, newy : Int) : Point {
          { x <- newx;
            y <- newy;
            self;
          } -- close block expression
     }; -- close method
}; -- close class
```

- A class can also define methods for manipulating its attributes
- Methods refer to the current object using **self**

Aside: Semicolons

```
class Point {
     x : Int <- 0;
     y : Int <- 0;
     movePoint(newx : Int, newy : Int) : Point {
          { x <- newx;
            y <- newy;
            self;
          } -- close block expression
     }; -- close method
}; -- close class
```

Yes, it's
somewhat arbitrary.
Still, don't get it wrong.

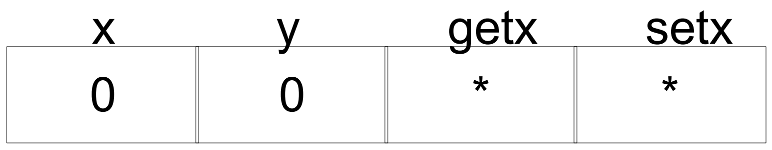
Information Hiding

- Methods are **global**
- Attributes are **local** to a class
  - They can *only* be accessed by *that class's methods*

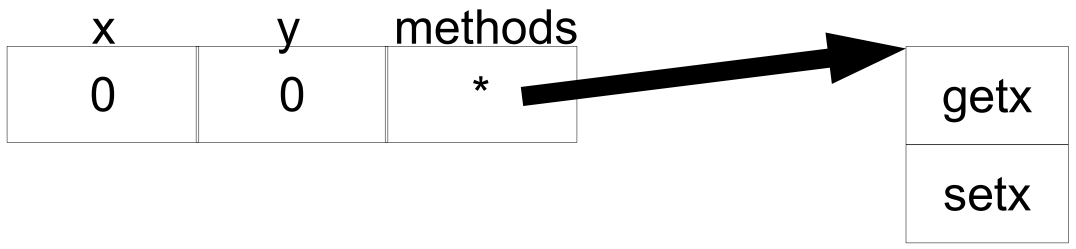
```
class Point {
      x : Int <- 0;
      y : Int <- 0;
      getx () : Int { x } ;
      setx (newx : Int) : Int { x <- newx };
};
```

Methods and Object Layout

- Each object knows how to access the code of its methods
- As if the object contains a slot pointing to the code



- In reality, implementations save space by sharing these pointers among instances of the same class



Inheritance

- We can extend points to color points using **subclassing** => **class hierarchy**

```
class ColorPoint extends Point {
      color : Int <- 0;
      movePoint(newx:Int, newy:Int) : Point {
            {  color <- 0;
               x <- newx; y <- newy;
               self;
            }
      };
};
```

Note references to fields x y –
They're defined in Point!

x	y	color	movePoint
0	0	0	*

Kool Types

Kool-Aid Man
OH, YEAAHH!
MATTEL ELECTRONICS INTELLIVISION
Use with any INTELLIVISION Master Component.

- Every class is a <u>**type**</u>
- Base (built-in, predefined) classes:

– **Int**	for integers
– **Bool**	for booleans: true, false
– **String**	for strings
– **Object**	root of class hierarchy

- All variables must be declared
  – compiler infers types for expressions (like Java)

Cool Type Checking

- x : Point;
- x <- new ColorPoint;

- ... is well-typed if Point is an ancestor of ColorPoint in the class hierarchy
  - Anywhere a Point is expected, a ColorPoint can be used (Liskov, ...)
- Rephrase: ... is well-typed if ColorPoint is a <u>subtype</u> of Point
- <u>Type safety</u>: a well-typed program *cannot* result in run-time type errors

Method Invocation and Inheritance

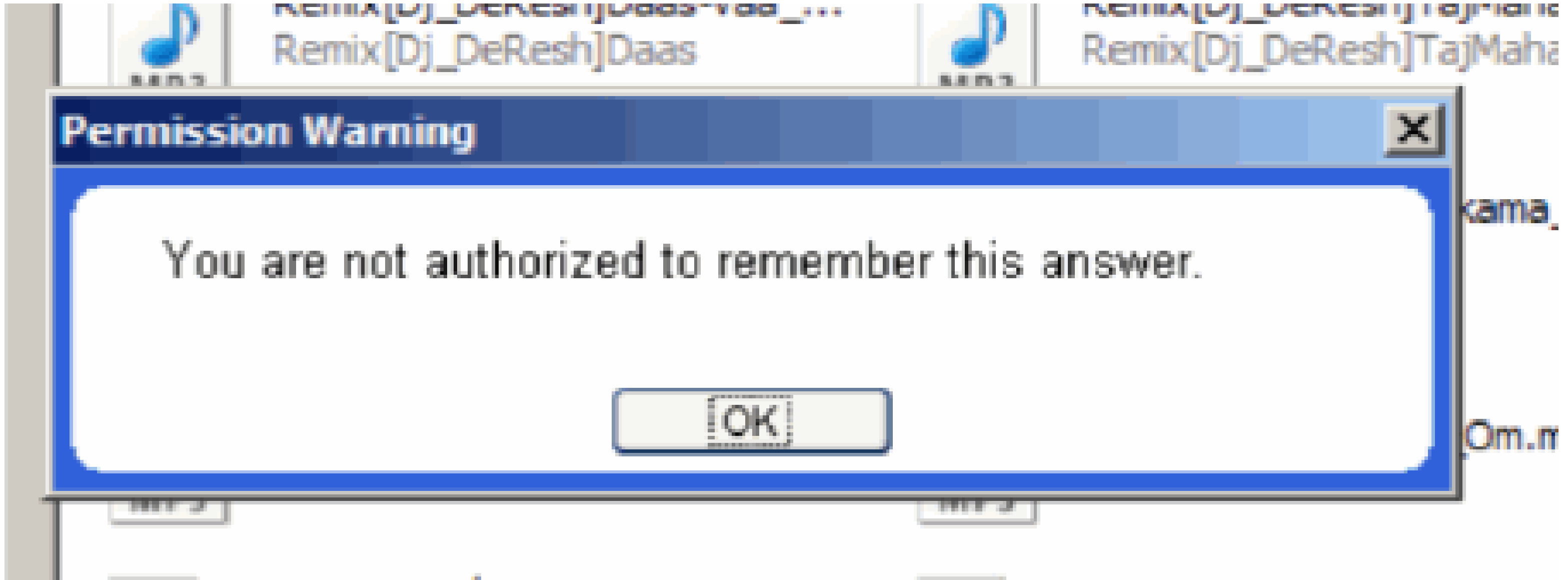
- Methods are invoked by (dynamic) **dispatch**
- Understanding dispatch in the presence of inheritance is a subtle aspect of OO
  - **p : Point;**
  - **p <- new ColorPoint;**
  - **p.movePoint(1,2);**
- p has static type Point
- p has dynamic type ColorPoint
- p.movePoint must invoke ColorPoint version

Other Expressions

- Cool is an expression language (like Ocaml)
  - Every expression has a type and a value
  - Conditionals   if E then E else E fi
  - Loops   while E loop E pool
  - Case/Switch   case E of x : Type => E ; ... esac
  - Assignment   x <- E
  - Primitive I/O   out_string(E), in_string(), ...
  - Arithmetic, Logic Operations, ...
- Missing: arrays, floats, interfaces, exceptions
  - Plus: you tell me!

Cool Memory Management

- Memory is allocated every time "**new E**" executes
- Memory is deallocated automatically when an object is not reachable anymore
  - Done by a **garbage collector** (GC)



You are not authorized to remember this answer.

Course Project

- A complete **interpreter**
  - Cool Source ==> Executed Program
  - No optimizations
  - Also no GC
- Split in 4 programming assignments (PAs)
- There is adequate time to complete assignments
  - But start early and follow directions
- PA2-5 ==> individual or teams (of max **2**)
- (Compilers: Also alone or teams of two.)

Real-Time OCaml Demo

- I will code up these, with explanations, until time runs out.
  - Read in a list of integers and print the sum of all of the odd inputs.
  - Read in a list of integers and determine if any sublist of that input sums to zero.
  - Read in a directed graph and determine if node END is reachable from node START.
- You pick the order.
- Bonus: Asymptotic running times?

Homework

- PA1c Due
- Reading: Chapters 2.1 – 2.2, Dijkstra, Landin

- Bonus for getting this far: questions about **<u>fold</u>** are very popular on tests! If I say “write me a function that does foozle to a list”, you should be able to code it up with fold.