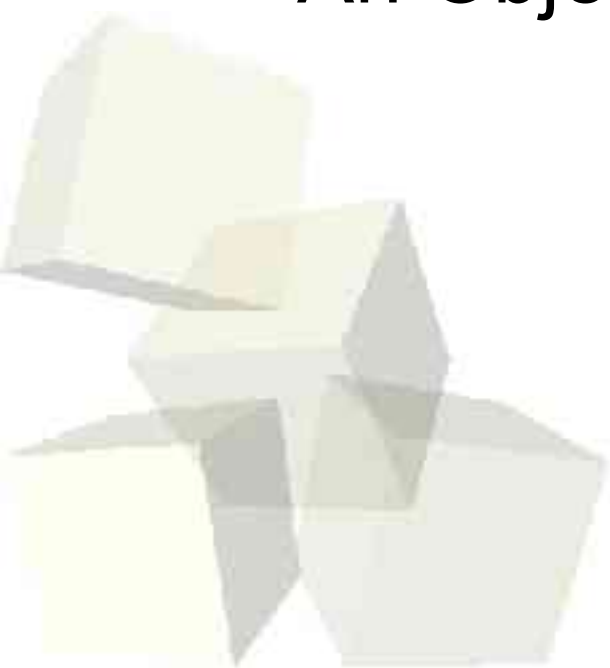




Quick Introduction to O'Caml

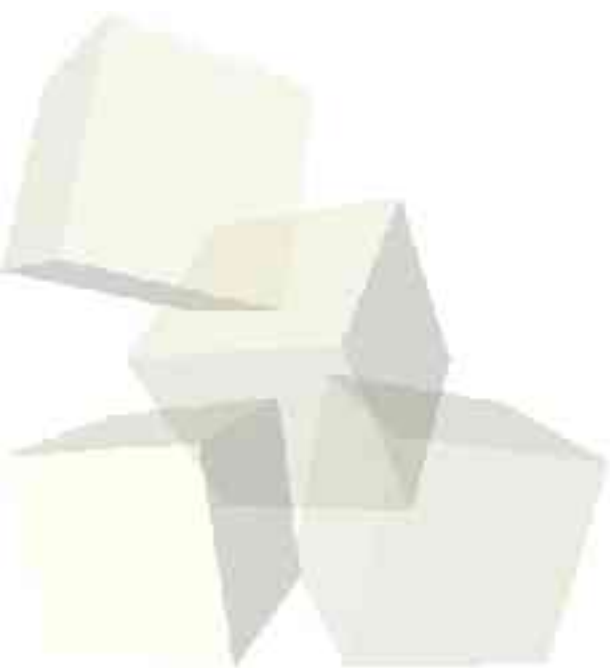
An Object-Oriented Functional Language
12-1-2004





Opening Discussion

- What did we talk about last class?
- Do you have any questions about the assignments?





Basics

- O'Caml, short of Objective Caml, is a derivative of ML. In that regard, it has many of the features that you are used to from ML. However, it is not so similar that you could just sit down and start
- One significant difference between ML and O'Caml is that O'Caml has more imperative aspects to it, or makes it easier to do things in an imperative way.

Simple Differences

- Operators for reals include “.”.
- No fun or val. Use let to define things. “in” is optional for local declarations.
- Recursive functions need to be defined with “let rec”.
- O'Cam1 uses ';' a LOT more than ML. Every statement ends with “;” and both lists and records are separated by ';' instead of ','.
- A function type can be declared with the keyword function instead of the shorthand.
 - let func = function(x,y) ->x+y;;



Patterns

- O'Cam1 also makes extensive use of patterns.
 - ◆ `match <exprt> with <p1> -> <e1> | ...`
- Patterns can include multiple constant/wildcard options or ranges of characters.
- You can't match a pattern on multiple arguments. To match on a curried function, you make the match separate using “`match <expr> with <p1> -> <e1> | ...`”
- Matches can have guards (`when`) or use character intervals.



Types

- You have to declare record types, and tuples aren't records. To get something out of a record type you use the “dot” notation or pattern matching. Some fields can simply be eliminated in a pattern. The labels are like constructors that can be hidden.
- You can create a revised copy of a record by giving $\{\langle \text{name} \rangle \text{ with } \langle l1 \rangle = \langle e1 \rangle; \dots\}$ where name is an existing record.
- type in O'Cam1 works like both type and datatype in ML.



Imperative Programming

- O'Caml has mutable types including vectors denoted with `[|...|]`. The elements are accessed with `“.<index>”` and you can mutate them using `<-`.
- Character strings are also mutable. We access their elements with `“.<index>”`.
- The keyword “mutable” can be put in front of a record name to make that field mutable.
- O'Caml also has reference types.
- We can do I/O similar to ML and commands can be sequenced with a single `;` between them.



Iteration and Thoughts

- O'Caml provides both for a while loops to do iteration.
- My only problem with the mutable stuff in O'Caml is that strings are mutable. That will make it much harder to build good parallelizing compilers.
- I will say though that the serial compilers for O'Caml do a very good job. The Great Computer Language Shootout shows this.



Classes and Objects in O'Caml

- As the name implies, O'Caml has objects in it and classes as well.
- Classes are defined as follows
 - `class <name> = object <methods and vals> end;;`
- Methods are created with the `method` keyword and members with `val`. A member can also be declared mutable.
- Methods are called with “#” instead of “.”.
- Immediate objects leave out the class declaration. “`object ... end`” These can occur in expressions.



More on Classes

- Classes in O'Cam1 allow inheritance as well as public and private methods.
- O'Cam1 also provides a way to easily do functional object-oriented programming, even with inheritance. Functions that would normally mutate an object need to return a new object of that type. The problem is that the exact type of a base class isn't the same as for derived classes. A `{< val=newVal; ... >}` syntax gets around this.
- Calling methods can have strong typing with open types.



Last Slide on Classes

- Classes get to name and refer to this (called self in O'Caml literature).
- Classes can be parameterized.
- They allow multiple inheritance.
- O'Caml provides subtyping and inclusion polymorphism in ways that are distinct from inheritance.
- Basically, class stuff in O'Caml is complex, but very powerful.

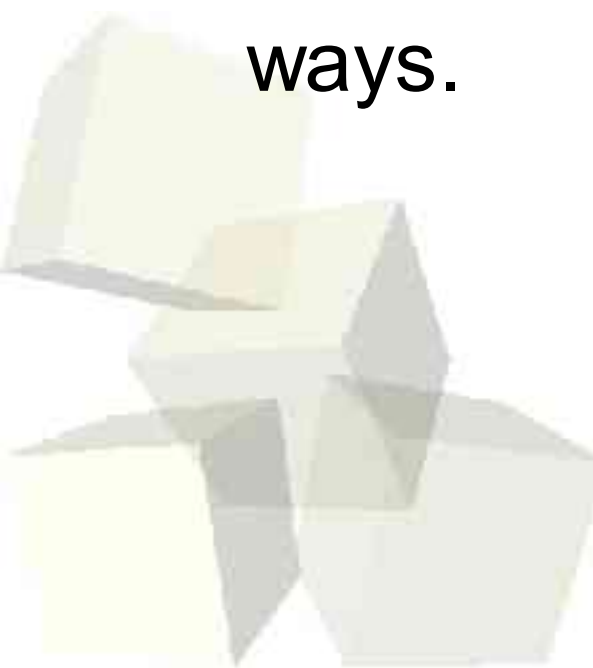


Libraries

- O'Caml has very extensive built-in libraries. These include not only things like data structures, but also graphics and concurrency.
- It also includes tools for doing lexical analysis and parsing.
- Basically, O'Caml comes with a depth of libraries and tools that is more Java in some regards. Unfortunately it doesn't seem quite as well documented to me.



Learning More

- You can learn more about O'Caml at ocaml.org. This site has links to downloads, manuals, and even a book in PDF format. For my learning style I actually liked the manual in PDF better than the book in many ways.
- 



Minute Essay

- Next time I teach functional, should I consider using O'CamI instead of ML? Why or why not?

