CMSC 330: Organization of Programming Languages

Imperative OCaml
So Far, Only Functional Programming

- We haven’t given you any way so far to change something in memory
  - All you can do is create new values from old

- This actually makes programming easier in some ways
  - Don’t care whether data is shared in memory
    - Aliasing is irrelevant
  - Provides strong support for compositional reasoning and abstraction
    - Ex: Calling a function $f$ with argument $x$ always produces the same result
Imperative OCaml

- There are three basic operations on memory:
  - `ref : 'a -> 'a ref`
    - Allocate an updatable reference
  - `! : 'a ref -> 'a`
    - Read the value stored in reference
  - `:= : 'a ref -> 'a -> unit`
    - Write to a reference

```ocaml
let x = ref 3 (* x : int ref *)
let y = !x
x := 4
```
Comparison To L- and R-values

- Recall that in C/C++/Java, there’s a strong distinction between l- and r-values
  - An r-value refers to just a value, like an integer
  - An l-value refers to a location that can be written

- A variable's meaning depends on where it appears
  - On the right-hand side, it’s an r-value, and it refers to the contents of the variable
  - On the left-hand side of an assignment, it’s an l-value, and it refers to the location the variable is stored in
L-Values and R-Values In C

Store 3 in location x

Makes no sense

Read contents of x and store in location y

Notice that x, y, and 3 all have type `int`
Comparison To OCaml

In OCaml, an updatable location and the contents of the location have different types
  • The location has a ref type

```c
int x;  // C
Int y;
x = 3;
y = x;
3 = x;
```

```ocaml
let x = ref 0;;
let y = ref 0;;
x := 3;;  (* x : int ref *)
y := (!x);;
3 := x;;  (* 3 : int; error *)
```
Capturing A Ref In A Closure

- We can use refs to make things like counters that produce a fresh number “everywhere”

```ocaml
let next =
    let count = ref 0 in
    function () ->
        let temp = !count in
        count := (!count) + 1;
        temp;;

# next ();;
- : int = 0
# next ();;
- : int = 1
```