CMSC 330: Organization of Programming Languages

Functional Programming with OCaml

Reminders / Announcements

- Project 3 was posted

More Basics…

```
# let l1 = [1;2;3];;
val l1 : int list = [1; 2; 3]
# let l2 = [1;2;3];;
val l2 : int list = [1; 2; 3]
# l1 == l2;;
- : bool = false          (shallow equality)
# l1 = l2;;
- : bool = true           (deep equality)
```

- <> is negation of =
- != is negation of ==

More Examples of Recursion

```
• sum l (* sum of elts in l *)
  let rec sum l = match l with
      [] -> 0
    | (x::xs) -> x + (sum xs)
```

```
• negate l (* negate elements in list *)
   let rec negate l = match l with
       [] -> []
     | (x::xs) -> (-x) :: (negate xs)
```

```
• last l  (* last element of l *)
  let rec last l = match l with
      [x] -> x
    | (x::xs) -> last xs
```

More Examples (cont'd)

```
(* return a list containing all the elements in the
   list l followed by all the elements in list m *)
```

- `append (l, m)`

```
let rec append (l, m) = match l with
    [] -> m
  | (x::xs) -> x::(append (xs, m))
```

- `rev l  (* reverse list; hint: use append *)`

```
let rec rev l = match l with
    [] -> []
  | (x::xs) -> append ((rev xs), [x])
```

- **`rev`** takes $O(n^2)$ time. Can you do better?

A Clever Version of Reverse

```
let rec rev_helper (l, a) = match l with
    [] -> a
  | (x::xs) -> rev_helper (xs, (x::a))
let rev l = rev_helper (l, [])
```

- Let's give it a try

```
rev [1; 2; 3] →
rev_helper ([1;2;3], []) →
rev_helper ([2;3], [1]) →
rev_helper ([3], [2;1]) →
rev_helper ([], [3;2;1]) →
[3;2;1]
```