# CMSC 330: Organization of Programming Languages

**Functional Programming with OCaml** 

## Background

- 1973 ML developed at Univ. of Edinburgh
  - Part of a theorem proving system LCF
    - The Logic of Computable Functions
- SML/NJ ("Standard ML of New Jersey")
  - http://www.smlnj.org
  - Developed at Bell Labs and Princeton; now Yale,
     AT&T Research, Univ. of Chicago (among others)
- OCaml
  - http://www.ocaml.org
  - Developed at INRIA (The French National Institute for Research in Computer Science)

CMSC 330 2

#### Dialects of ML

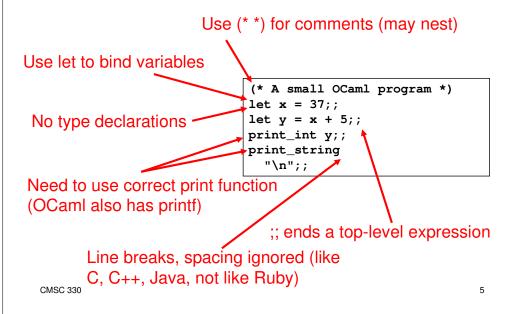
- Other dialects include MoscowML, ML Kit, Concurrent ML, etc.
  - But SML/NJ and OCaml are most popular
  - O = "Objective," but probably won't cover objects
- · Languages all have the same core ideas
  - But small and annoying syntactic differences
  - So you shouldn't buy a book just because it has ML in the title, because it may not cover OCaml

#### Features of ML

- · Higher-order functions
  - Functions can be parameters and return values
- "Mostly functional"
- Data types and pattern matching
  - Convenient for certain kinds of data structures
- Type inference
  - No need to write types in the source language, but the language is statically typed
  - Supports parametric polymorphism (like generics in Java, and templates in C++)
- Exceptions
- Garbage collection

CMSC 330 3 CMSC 330

#### A Small OCaml Program- Things to Notice



## Run, OCaml, Run

- OCaml programs can be compiled using ocamle
  - produces .cmo ("compiled object") and .cmi ("compiled interface") files
    - · we'll talk about interface files later
  - by default, also links to produce executable a.out
    - · use -o to specify output file name
    - use -c to compile only to .cmo/.cmi and not to link

CMSC 330 6

## Run, OCaml, Run (cont'd)

Compiling and running the previous small program:

```
Ocaml1.ml:
  (* A small OCaml program *)
let x = 37;;
let y = x + 5;;
print_int y;;
print_string "\n";;
```

```
% ocamlc ocaml1.ml
% ./a.out
42
%
```

CMSC 330

## Run, OCaml, Run (cont'd)

OCaml also has a special top-level, similar to Ruby

```
ocaml1.ml:
 % ocaml
                                             (* A small OCaml program *)
        Objective Caml version 3.12.1
                                             let x = 37;;
                                             let v = x + 5;;
                                             print_int y;;
 # #use "ocaml1.ml";;
                                             print_string "\n";;
 val x : int = 37
 val y : int = 42
                            #use loads in a file one line at a time
 42- : unit = ()
                           prints type and value of each expr
 -: unit =()
# x;;
  : int = 37
                      unit = "no interesting value" (like void)
                    "-" = "the expression you just typed"
CMSC 330
```

## Run, OCaml, Run (cont'd)

Expressions can also be typed and evaluated at the top-level:

```
# 3 + 4;;
-: int = 7
# let x = 37;;
val x : int = 37
# x;;
-: int = 37
# let y = 5;;
val y : int = 5
\# let z = 5 + x;
val z : int = 42
# print_int z;;
42- : unit = ()
# print_string "Larry Herman is amazing!";;
Larry Herman is amazing! - : unit = ()
# print_int "Larry Herman is amazing!";;
This expression has type string but is here used with type int
```

## **Basic Types in OCaml**

Read e: t has "expression e has type t"

```
42 : int true : bool
"hello" : string 'c' : char
3.14 : float () : unit (* don't care value *)
```

OCaml has static types to help you avoid errors

```
- Note: Sometimes the messages are a bit confusing
# 1 + true;;
This expression has type bool but is here used with
type int
```

- Watch for the underline as a hint to what went wrong
- But not always reliable

CMSC 330 10

## More on the Let Construct

- let is more often used for local variables
  - let x = e1 in e2 means
    - evaluate e1
    - then evaluate e2, with x bound to result of evaluating e1
    - x is not visible outside of e2

```
let pi = 3.14 in pi *. 3.0 *. 3.0;;
pi;;

bind pi in body of let floating point multiplication
```

error

# More on the Let Construct (cont'd)

· Compare to similar usage in Java/C

```
let pi = 3.14 in
  pi *. 3.0 *. 3.0;;
pi;;
```

```
float pi = 3.14;

pi * 3.0 * 3.0;
}
pi;
```

• In the top-level, omitting in means "from now on":

```
# let pi = 3.14;;
```

(\* pi is now bound in the rest of the top-level scope \*)

CMSC 330 11 CMSC 330 12

# **Nested Let**

• Uses of let can be nested

```
let pi = 3.14 in
let r = 3.0 in
   pi *. r *. r;;
(* pi, r no longer in scope *)
```

```
{
  float pi = 3.14;
  float r = 3.0;

  pi * r * r;
}
/* pi, r not in scope */
```

CMSC 330 13