Programming Languages and Compilers (CS 421)

Elsa L Gunter

2112 SC, UIUC

https://courses.engr.illinois.edu/cs421/sp2023

Based in part on slides by Mattox Beckman, as updated by Vikram Adve and Gul Agha

Features of OCAML

- Higher order applicative language
- Call-by-value parameter passing
- Modern syntax
- Parametric polymorphism
  - Aka structural polymorphism
- Automatic garbage collection
- User-defined algebraic data types

Why learn OCAML?

- Many features not clearly in languages you have already learned
- Assumed basis for much research in programming language research
- OCAML is particularly efficient for programming tasks involving languages (eg parsing, compilers, user interfaces)
- Industrially Relevant:
  - Jane Street trades billions of dollars per day using OCaml programs
  - Major language supported at Bloomberg
- Similar languages: Microsoft F#, SML, Haskell, Scala

Session in OCAML

```
% ocaml
Objective Caml version 4.07.1
# (* Read-eval-print loop; expressions and
   declarations *)
  2 + 3;;     (* Expression *)
- : int = 5
# 3 < 2;;
- : bool = false
```

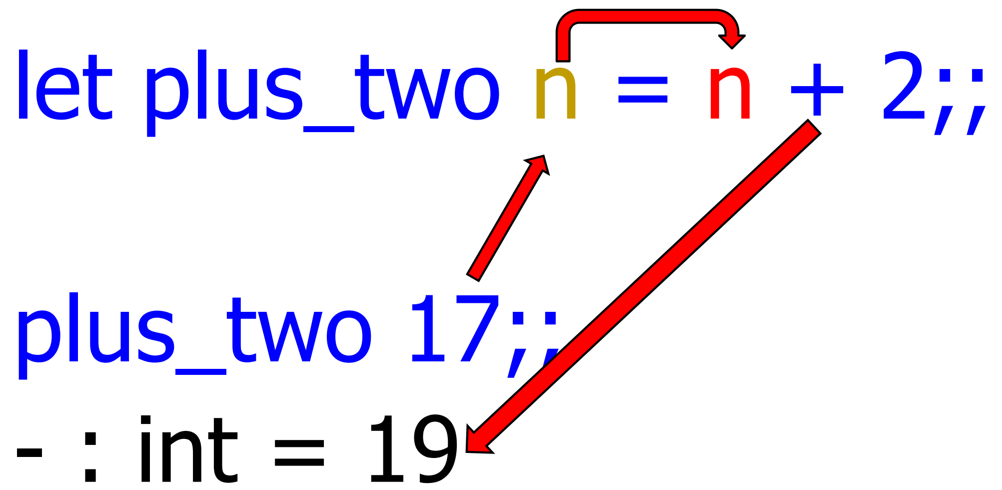
Declarations; Sequencing of Declarations

```
# let x = 2 + 3;;   (* declaration *)
val x : int = 5
# let test = 3 < 2;;
val test : bool = false
# let a = 1 let b = a + 4;; (* Sequence of dec
  *)
val a : int = 1
val b : int = 5
```

Functions

```
# let plus_two n = n + 2;;
val plus_two : int -> int = <fun>
# plus_two 17;;
- : int = 19
```

Functions



Environments

- *Environments* record what value is associated with a given identifier
- Central to the semantics and implementation of a language
- Notation

$$\rho = \{name_1 \rightarrow value_1, name_2 \rightarrow value_2, \ldots\}$$

Using set notation, but describes a partial function

- Often stored as list, or stack
  - To find value start from left and take first match

Environments

X → 3

name → "Steve"

. . .

y → 17

region → (5.4, 3.7)

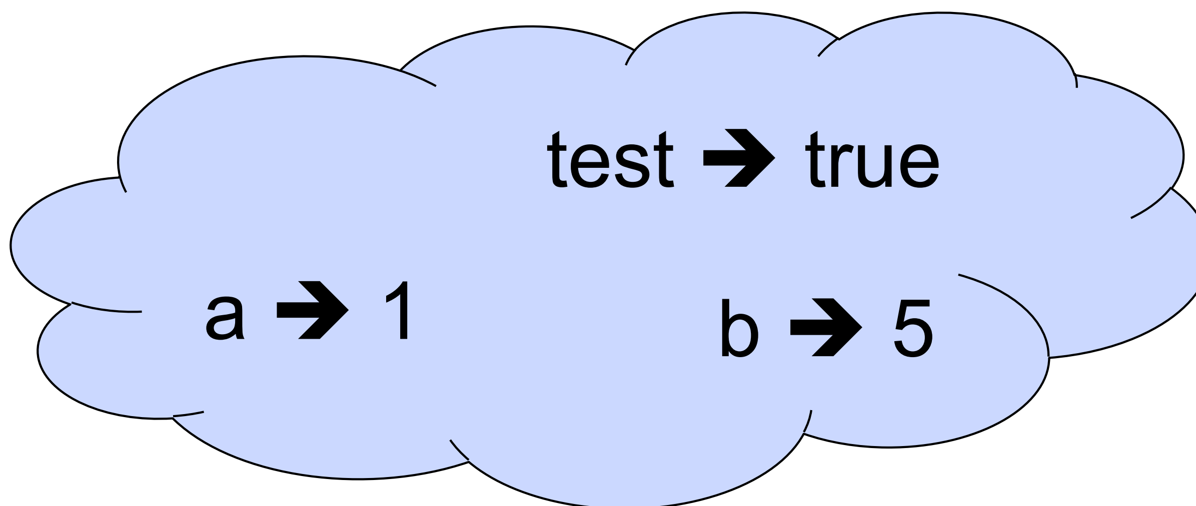
id → {Name = "Paul",
Age = 23,
SSN = 999888777}

b → true

Global Variable Creation

```
# 2 + 3;;     (* Expression *)
// doesn't affect the environment
# let test = 3 < 2;;     (* Declaration *)
val test : bool = false
//  ρ1 = {test → false}
# let a = 1 let b = a + 4;; (* Seq of dec *)
//  ρ2 = {b → 5, a → 1, test → false}
```

Environments



$// \ \rho_2 = \{b \rightarrow 5, a \rightarrow 1, test \rightarrow false\}$

```
let test = 3.7;;
```

- What is the environment after this declaration?

New Bindings Hide Old

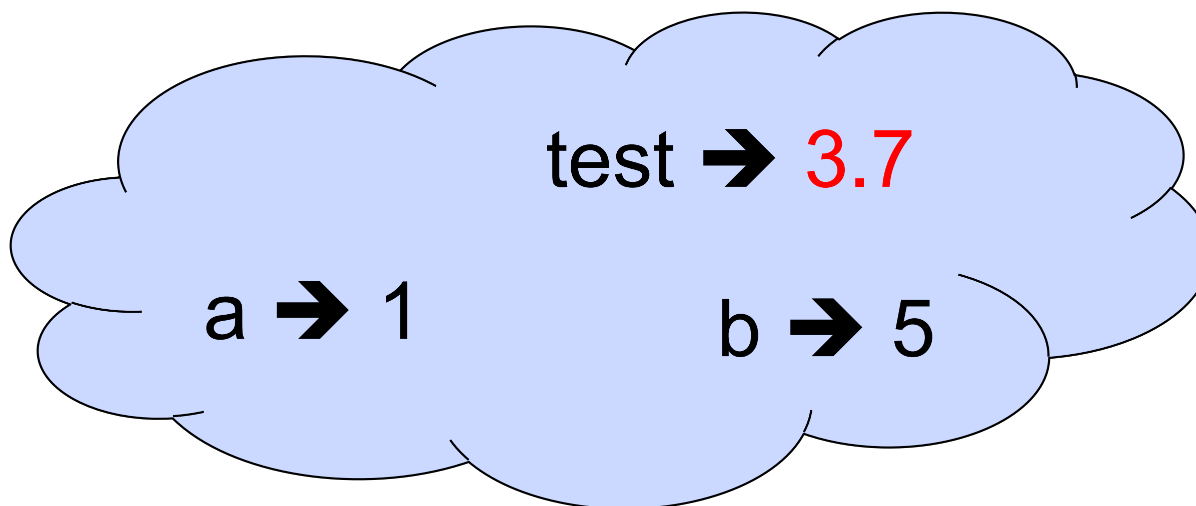
// $\rho_2 = \{b \to 5, a \to 1, test \to false\}$

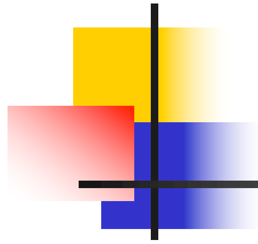
```
let test = 3.7;;
```

- What is the environment after this declaration?

// $\rho_3 = \{test \to 3.7, a \to 1, b \to 5\}$

Environments





Now it’s your turn

You should be able to start ACT1

Local Variable Creation

// ρ_3 = {test → 3.7, a → 1, b → 5}

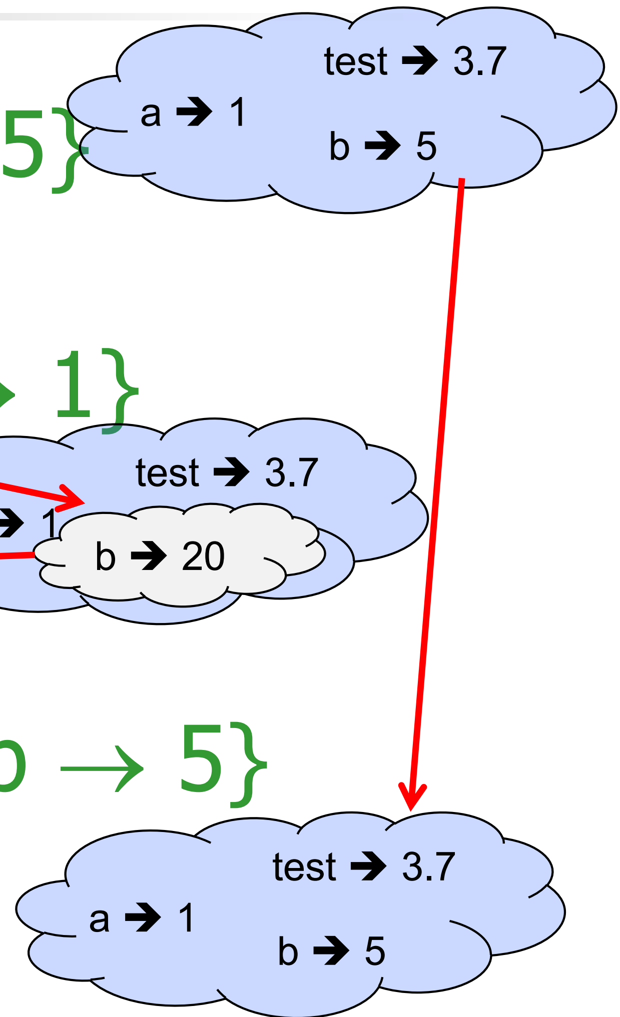
```
# let b = 5 * 4
```

// ρ_4 = {b → 20, test → 3.7, a → 1}

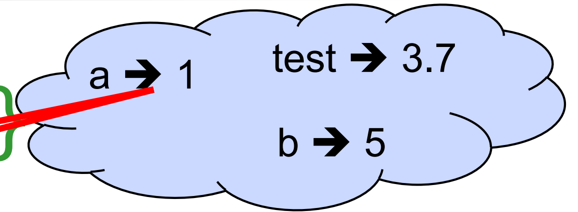
```
    in 2 * b;;
- : int = 40
```

// ρ_5 = ρ_3= {test → 3.7, a → 1, b → 5}

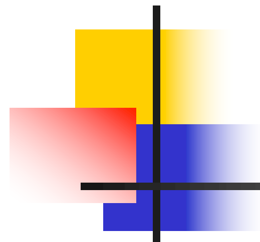
```
# b;;
- : int = 5
```



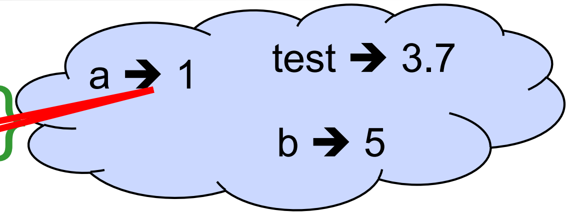
Local let binding




```
// ρ5 =ρ3 ={test → 3.7, a → 1, b → 5}
# let c =
    let b = a + a
//   ρ6 = {b → 2} + ρ3
//       ={b → 2, test → 3.7, a → 1}
    in b * b;;
val c : int = 4
//   ρ7 = {c → 4, test → 3.7, a → 1, b → 5}
# b;;
- : int = 5
```



Local let binding

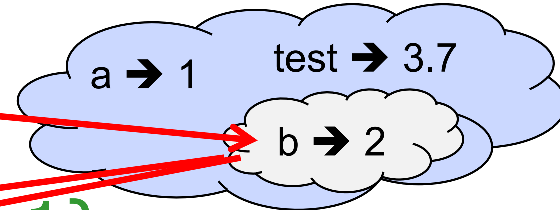


// $\rho_5 = \rho_3 = \{test \rightarrow 3.7, a \rightarrow 1, b \rightarrow 5\}$

```
# let c =
    let b = a + a
```

// $\rho_6 = \{b \rightarrow 2\} + \rho_3$

// $\quad = \{b \rightarrow 2, test \rightarrow 3.7, a \rightarrow 1\}$




```
    in b * b;;
val c : int = 4
```

// $\rho_7 = \{c \rightarrow 4, test \rightarrow 3.7, a \rightarrow 1, b \rightarrow 5\}$

```
# b;;
- : int = 5
```

Local let binding

// $\rho_5 = \rho_3 = \{test \rightarrow 3.7, a \rightarrow 1, b \rightarrow 5\}$

```
# let c =
    let b = a + a
```

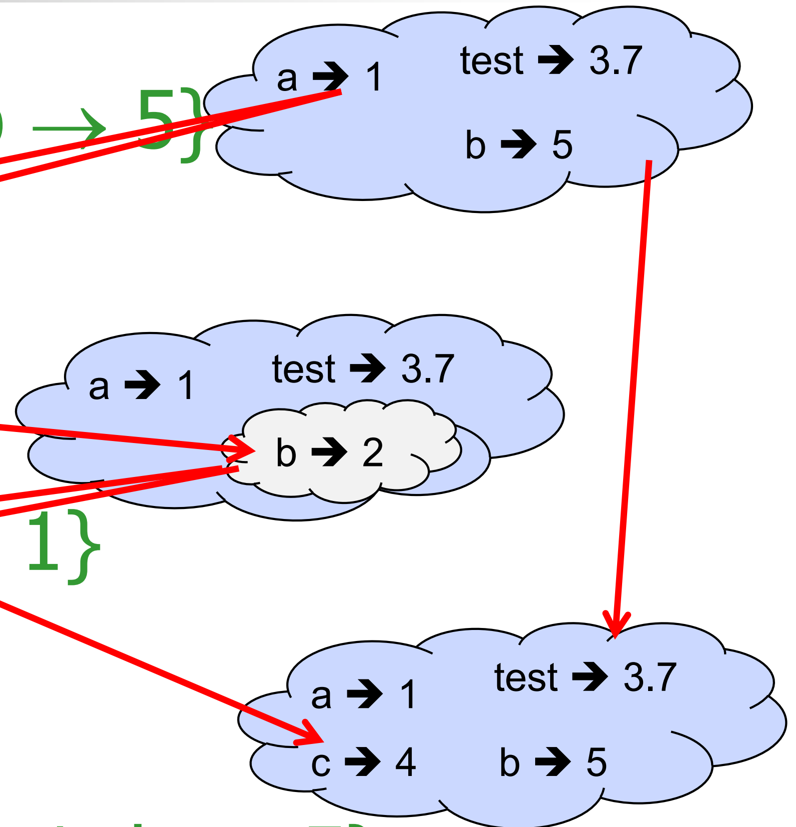
// $\rho_6 = \{b \rightarrow 2\} + \rho_3$

// $= \{b \rightarrow 2, test \rightarrow 3.7, a \rightarrow 1\}$

```
    in b * b;;
val c : int = 4
```

// $\rho_7 = \{c \rightarrow 4, test \rightarrow 3.7, a \rightarrow 1, b \rightarrow 5\}$

```
# b;;
- : int = 5
```



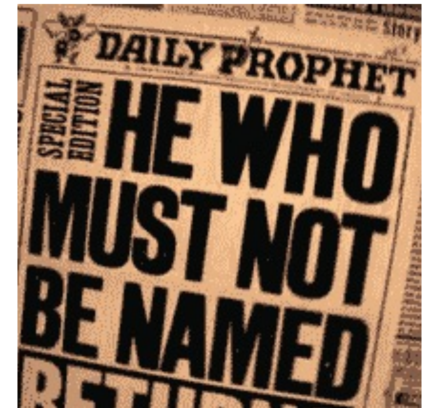
Functions

```
# let plus_two n = n + 2;;
val plus_two : int -> int = <fun>
# plus_two 17;;
- : int = 19
```

Functions

```
let plus_two n = n + 2;;

plus_two 17;;
- : int = 19
```

Nameless Functions (aka Lambda Terms)




```
fun n -> n + 2;;

(fun n -> n + 2) 17;;
- : int = 19
```

Functions

```
# let plus_two n = n + 2;;
val plus_two : int -> int = <fun>
# plus_two 17;;
- : int = 19
# let plus_two = fun n -> n + 2;;
val plus_two : int -> int = <fun>
# plus_two 14;;
- : int = 16
```

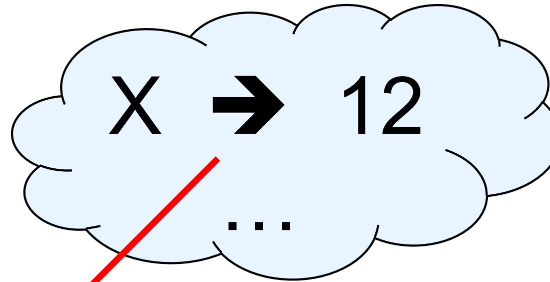
First definition syntactic sugar for second

Using a nameless function

```
# (fun x -> x * 3) 5;;   (* An application *)
- : int = 15
# ((fun y -> y +. 2.0), (fun z -> z * 3));;
  (* As data *)
- : (float -> float) * (int -> int) = (<fun>,
  <fun>)
```

Note: in fun v -> exp(v), scope of variable is only the body exp(v)

Values fixed at declaration time




```
# let x = 12;;
val x : int = 12
# let plus_x y = y + x;;
val plus_x : int -> int = <fun>
# plus_x 3;;
```

What is the result?

Values fixed at declaration time

```
# let x = 12;;
val x : int = 12
# let plus_x y = y + x;;
val plus_x : int -> int = <fun>
# plus_x 3;;
- : int = 15
```

Values fixed at declaration time

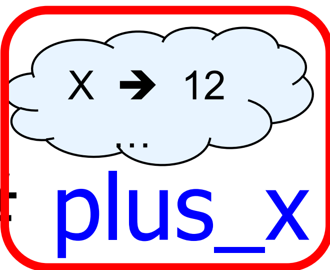
```
# let x = 7;;   (* New declaration, not an update *)
val x : int = 7

# plus_x 3;;
```

What is the result this time?

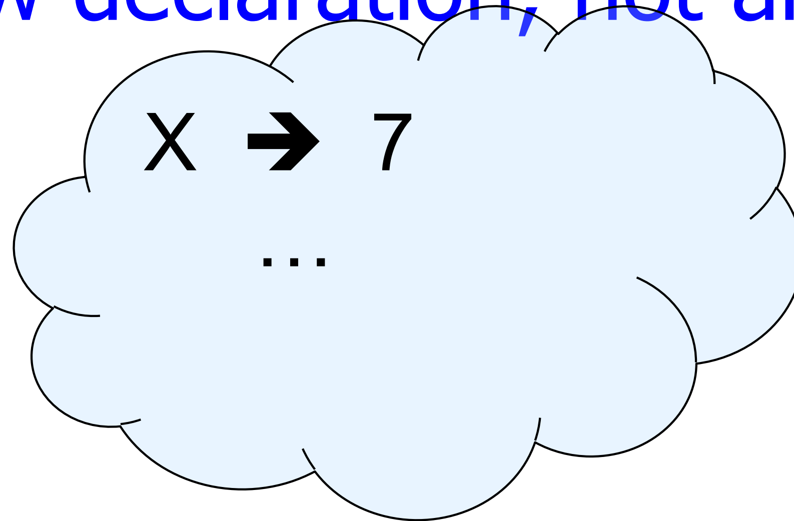
Values fixed at declaration time

let x = 7;;   (* New declaration, not an update *)

val x : int = 7



plus_x 3;;



What is the result this time?

Values fixed at declaration time

```
# let x = 7;;   (* New declaration, not an
  update *)
val x : int = 7

# plus_x 3;;
- : int = 15
```

Question

- Observation: Functions are first-class values in this language
- Question: What value does the environment record for a function variable?
- Answer: a closure

Save the Environment!

- A *closure* is a pair of an environment and an association of a formal parameter (the input variables)* with an expression (the function body), written:

$$f \rightarrow < (v1,\ldots,vn) \rightarrow exp, \rho_f >$$

- Where ρ_f is the environment in effect when f is defined (if f is a simple function)

* Will come back to the “formal parameter”

Closure for plus_x

- When plus_x was defined, had environment:

$$\rho_{plus_x} = \{\ldots, x \rightarrow 12, \ldots\}$$

- Recall: let plus_x y = y + x

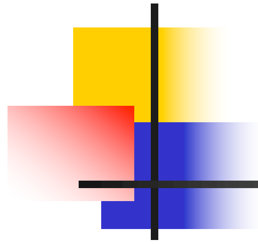
  is really let plus_x = fun y -> y + x

- Closure for fun y -> y + x:

$$< y \rightarrow y + x, \rho_{plus_x} >$$

- Environment just after plus_x defined:

$$\{plus_x \rightarrow < y \rightarrow y + x, \rho_{plus_x} >\} + \rho_{plus_x}$$



Now it’s your turn

You should be able to complete ACT1

125 minutes