

## 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

1/20/23

1

## 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

1/20/23

2

## 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

1/20/23

3

## 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
```

1/20/23

4

## 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 declarations *)
```

```
val a : int = 1
```

```
val b : int = 5
```

1/20/23

5

## Functions

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

```
val plus_two : int -> int = <fun>
```

```
# plus_two 17;;
```

```
- : int = 19
```

1/20/23

6

## Functions

```
let plus_two n = n + 2;;  
plus_two 17;;  
- : int = 19
```

1/20/23

7

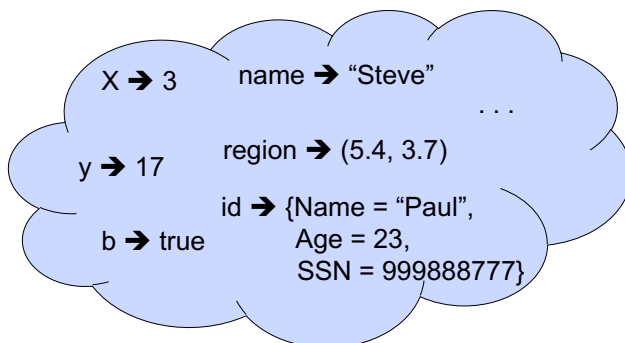
## Environments

- *Environments* record what value is associated with a given identifier
- Central to the semantics and implementation of a language
- Notation  
 $\rho = \{name_1 \rightarrow value_1, name_2 \rightarrow value_2, \dots\}$   
Using set notation, but describes a partial function
- Often stored as list, or stack
  - To find value start from left and take first match

1/20/23

20

## Environments



1/20/23

21

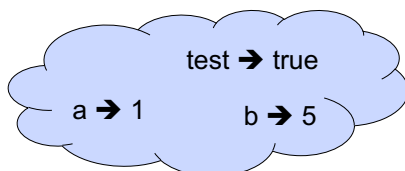
## Global Variable Creation

```
# 2 + 3;;    (* Expression *)  
// doesn't affect the environment  
# let test = 3 < 2;;    (* Declaration *)  
val test : bool = false  
//  $\rho_1 = \{\text{test} \rightarrow \text{false}\}$   
# let a = 1 let b = a + 4;; (* Seq of dec *)  
//  $\rho_2 = \{b \rightarrow 5, a \rightarrow 1, \text{test} \rightarrow \text{false}\}$ 
```

1/20/23

22

## Environments



1/20/23

23

## New Bindings Hide Old

```
//  $\rho_2 = \{b \rightarrow 5, a \rightarrow 1, \text{test} \rightarrow \text{false}\}$   
let test = 3.7;;
```

- What is the environment after this declaration?

1/20/23

24

## New Bindings Hide Old

```
// ρ2 = {b → 5, a → 1, test → false}
let test = 3.7;;
```

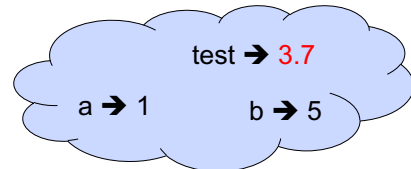
- What is the environment after this declaration?

```
// ρ3 = {test → 3.7, a → 1, b → 5}
```

1/20/23

25

## Environments



1/20/23

26

Now it's your turn

You should be able to start ACT1

1/20/23

27

## Local Variable Creation

```
// ρ3 = {test → 3.7, a → 1, b → 5}
```

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

```
// ρ4 = {b → 20, test → 3.7, a → 1}
```

```
in 2 * b;;
```

```
- : int = 40
```

```
// ρ5 = ρ3 = {test → 3.7, a → 1, b → 5}
```

```
# b;;
```

```
- : int = 5
```

1/20/23

29

## 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
```

1/20/23

30

## 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
```

1/20/23

31

## 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
```

1/20/23

32

## Functions

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

1/20/23

33

## Functions

```
let plus_two n = n + 2;;
plus_two 17;;
- : int = 19
```

1/20/23

34

## Nameless Functions (aka Lambda Terms)

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



1/20/23

35

## 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

1/20/23

36

## 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>)
```

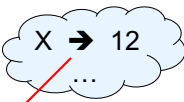
Note: in fun v -> exp(v), scope of variable is only the body exp(v)

1/20/23

37

## 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?

1/20/23

39

## 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
```

1/20/23

40

## Values fixed at declaration time

```
# let x = 7;; (* New declaration, not an  
update *)  
val x : int = 7  
  
# plus_x 3;;
```

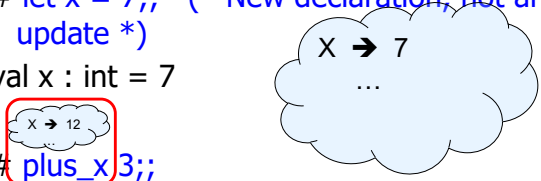
What is the result this time?

1/20/23

41

## 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?

1/20/23

42

## Values fixed at declaration time

```
# let x = 7;; (* New declaration, not an  
update *)  
val x : int = 7  
  
# plus_x 3;;  
- : int = 15
```

1/20/23

43

## Question

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

1/20/23

44

## 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:

$$f \rightarrow \langle (v_1, \dots, v_n) \rightarrow \text{exp}, \rho_f \rangle$$

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

\* Will come back to the "formal parameter"

1/20/23

45

## Closure for plus\_x

- When plus\_x was defined, had environment:

$$\rho_{\text{plus\_x}} = \{\dots, x \rightarrow 12, \dots\}$$

- Recall: `let plus_x y = y + x`

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

- Closure for `fun y -> y + x`:

$$\langle y \rightarrow y + x, \rho_{\text{plus\_x}} \rangle$$

- Environment just after plus\_x defined:

$$\{\text{plus\_x} \rightarrow \langle y \rightarrow y + x, \rho_{\text{plus\_x}} \rangle\} + \rho_{\text{plus\_x}}$$

1/20/23

46

Now it's your turn

You should be able to  
complete ACT1

1/20/23

47

125 minutes

1/20/23

48