# CSE 130 [Winter 2014] Programming Languages

## Introduction to OCaml

(Continued)

Ravi Chugh

UCSDCSE
Computer Science and Engineering

Jan 09

# Announcements

- Accounts / Piazza still not ready ☹

  http://try.ocamlpro.com/

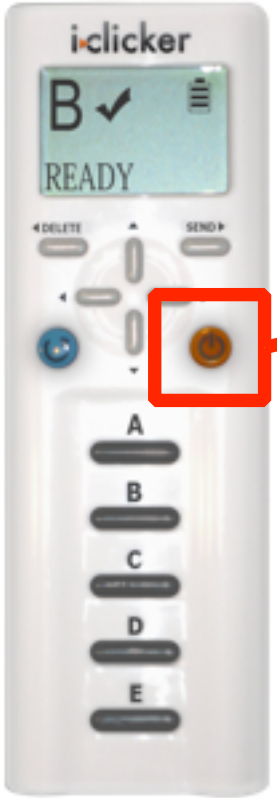
- HW #0 posted

- HW #1 posted after class, due Fri Jan 17

- Group Assignments and Seating Posted

  Let me know of special requests

# Clicker Frequency for This Room




1. Press and hold until blinking…
2. Enter BD

# Clicker Test

What's your favorite letter?

(a) a
(b) b
(c) c
(d) d
(e) None of the above

# Clicker Vote

**Discussion Sections**

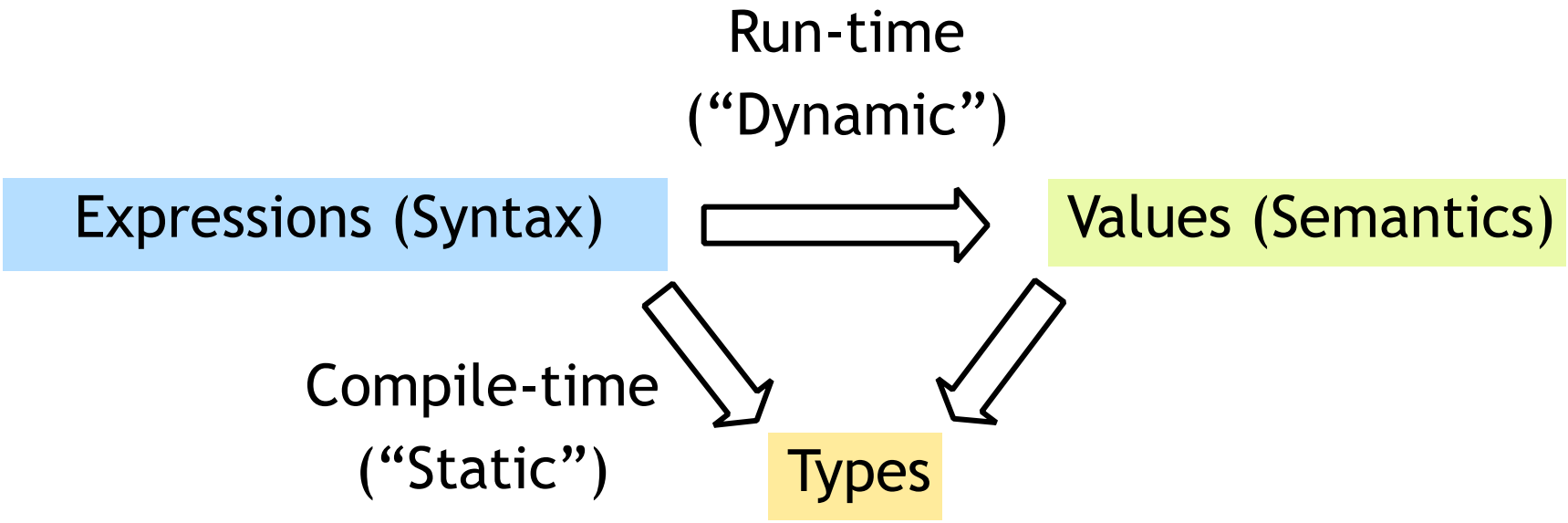
Wednesdays + Fridays

9:00am to 9:50 am

CSB 001

| | | |
|---|---|---|
| a. | Wed “Review” | Fri “Review” |
| b. | Wed “Review” | Fri “Extra” |
| c. | Wed “Extra” | Fri “Review” |
| d. | Wed “Extra” | Fri “Extra” |

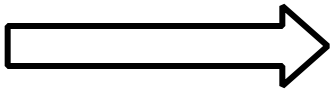
# ML's Holy Trinity




1. Enter an expression **e**
2. ML infers a type **t** or emits an **error**
3. ML evaluates expression **e** down to a value **v**
4. Value **v** is guaranteed to have type **t**

# Complex Type: Tuples (Products)

```
(9-3,"ab"^"cd",(2+2  ,7>8))
```



**(6, "abcd",(4,false))**

**(int * string * (int * bool))**

- Pairs, Triples, Quadruples, …
- Nesting:
  - Everything is an expression
  - Nest tuples in tuples

# Complex Type: Lists

| | | |
|---|---|---|
| `[];` | [] | 'a list |
| `[1;2;3];` | [1;2;3] | int list |
| `[1+1;2+2;3+3;4+4];` | [2;4;6;8] | int list |
| `["a";"b"; "c"^"d"];` | ["a";"b"; "cd"] | string list |
| `[(1,"a"^"b");(3+4,"c")];` | [(1,"ab");(7,"c")] | (int*string) list |
| `[[1];[2;3];[4;5;6]];` | [[1];[2;3];[4;5;6]]; | (int list) list |

- Unbounded size
- Can have lists of anything (e.g. lists of lists)
- But …

# Complex Type: Lists

```
[1; "pq"];
```

All elements must have same type

# Clicker Question

Which of these causes a type error?

```
(a) [1, 2, 3]
(b) ["1", 2, 3]
(c) "[1; 2; 3]"
(d) (1, 2, 3)
(e) ["1"; 2; 3]
```

# Lists: “Cons”truct

Nil operator

```
[]
```

```
[]=>[]
```

```
[] : 'a list
```

Cons operator

```
1::[2;3]
```

```
[1;2;3]
```

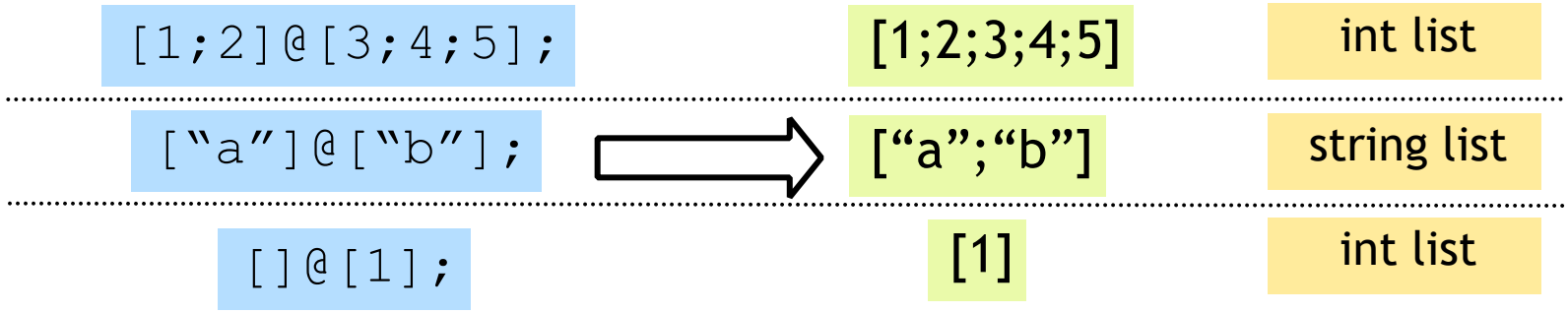
int list

$$\frac{e1 : T \quad e2 : T\ list}{e1::e2 : T\ list}$$

$$\frac{e1 \Rightarrow v1 \quad e2 \Rightarrow v2}{e1::e2 \Rightarrow v1::v2}$$

# Complex Type: Lists

List operator “Append” `@`

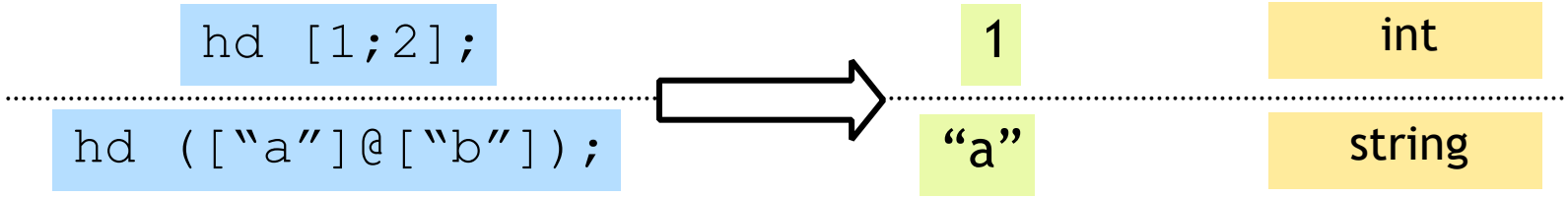



Can only append two lists… `1 @ [2;3];`

… of the same type `[1] @ [“a”;“b”];`

# Complex Type: Lists

List operator “Head” `List.hd`




ML types can’t catch some errors though…

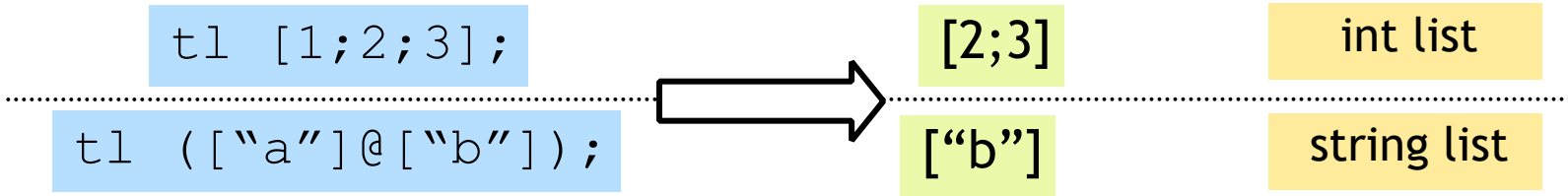
```
hd []
```

```
Exception: Failure "hd".
```

(ML does infer a type…)

# Complex Type: Lists

List operator “tail” `List.tl`




The tail of empty list is a run-time error…

```
tl []
Exception: Failure "tl".
```

Expressiveness of type systems is an **active area of research!**

# Clicker Question

What is the result of

```
(hd [[]; [1;2;3]]) = (hd [[]; ["a"]])
```

?

(a) Syntax Error

(b) `true : bool`

(c) `false : bool`

(d) Type Error (hd)

(e) Type Error (=)

# Lists: Deconstruct (or Destruct)

Head

$$\frac{e : T\ list}{hd\ e : T} \qquad \frac{e \Rightarrow v1::v2}{hd\ e \Rightarrow v1}$$

Tail

$$\frac{e : T\ list}{tl\ e : T\ list} \qquad \frac{e \Rightarrow v1::v2}{tl\ e \Rightarrow v2}$$

```
(hd [[];[1;2;3]]) = (hd [[];["a"]])
```

int list

string list

$$\frac{e_1 : T \quad e_2 : T}{e_1 = e_2 : bool}$$

# Recap: Tuples vs. Lists

What's the difference ?

- Tuples:
  - Different types, but fixed number:

    (3, "abcd") (int * string)

    - pair = 2 elts

    (3, "abcd",(3.5,4.2)) (int * string * (float * float))

    - triple = 3 elts
- Lists:
  - Same type, unbounded number:

    [3;4;5;6;7] int list
- Syntax:
  - Tuples = comma Lists = semicolon

# What About …

# Branches

# Clicker Question

What is the result of

```
if (1 < 2) then true else false
```

?

(a) Syntax Error

(b) `true`

(c) `false`

(d) Type Error

# Clicker Question

What is the result of

```
if (1 < 2) then [1;2] else 5
```

?

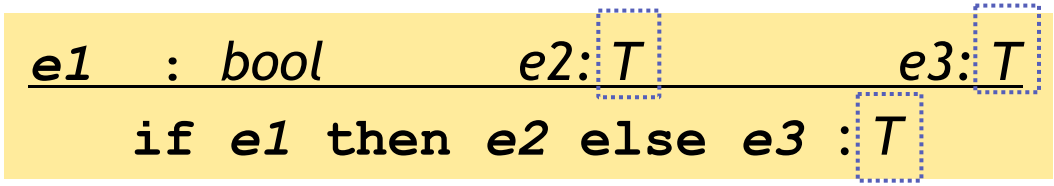
(a) Syntax Error

(b) `[1;2]`

(c) `5`

(d) Type Error

# If-then-else expressions



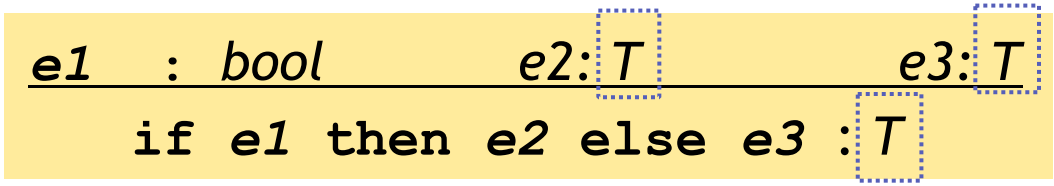

then-subexp and else-subexp must have the same type T!

and, if so, the overall if-expression has type T

```
if (1 < 2) then [1;2] else 5
```

```
if false then [1;2] else 5
```

# If-then-else expressions




then-subexp and else-subexp must have the same type T!

and, if so, the overall if-expression has type T

```
if 1>2 then [1,2] else []
```
[]

int list

```
if 1<2 then [] else [“a”]
```
[]

string list

```
(if 1>2 then [1,2] else []) = (if 1<2 then [] else [“a”])
```

# What About …

# Variables

# Clicker Question

I got this at the prompt:

```
# [x+x; x*x] ;;
- : int list = [20; 100]
```

What could I have typed before?

```
(a) x = 10 ;;
(b) int x = 10 ;;
(c) x == 10 ;;
(d) let x = 10 ;;
(e) x := 10 ;;
```

# Variables and Bindings

```
let x = e;;
```

"Bind the value of

expression e to the variable x"

```
# let x = 2 + 2;;
val x : int = 4
```

# Variables and Bindings

Expressions that appear later can use `x`

- Most recent "bound" value used for evaluation

```
# let x = 2 + 2;;
val x : int = 4
# let y = x * x * x;;
val y : int = 64
# let z = [x; y; x+y];;
val z : int list = [4;64;68]
#
```

# Variables and Bindings

Undeclared variables
(i.e. without a value binding)
are not accepted !

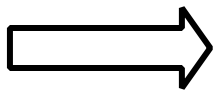
```
# let p = a + 1;;
Error: Unbound value a
```

Catches many bugs due to typos

# Local Bindings

… for expressions using “temporary” variables

```
let
   tempVar = x + 2 * y
in
  tempVar * tempVar
;;
```



17424 int

- `tempVar` is bound only inside expr body from `in` …
- Not visible (“not in scope”) outside

# Clicker Question

What is the result of

```
let x = 10 in
  (let z = 10 in x + z) + z
```

?

(a) Syntax Error

(b) 30

(c) Unbound Var Error (x)

(d) Unbound Var Error (z)

(e) Other Type Error

# Binding by Pattern-Matching

Simultaneously bind several variables

```
# let (x,y,z) = (2+3,"a"^"b", 1::[2]);;
val x : int = 5
val y : string = "ab"
val z : int list = [1;2]
```

# Binding by Pattern-Matching

But:

```
# let h::t = [1;2;3];;
Warning P: this pattern-matching not exhaustive.
val h : int = 1
val t : int list = [2;3]
```

Why is it whining ?

```
# let h::t = [];
Exception: Match_failure
# let xs = [1;2;3];;
val xs : int list = [1;2;3]
# let h::t = xs;;
Warning: Binding not exhaustive
val h : int = 1
val t : int list = [2;3]
```

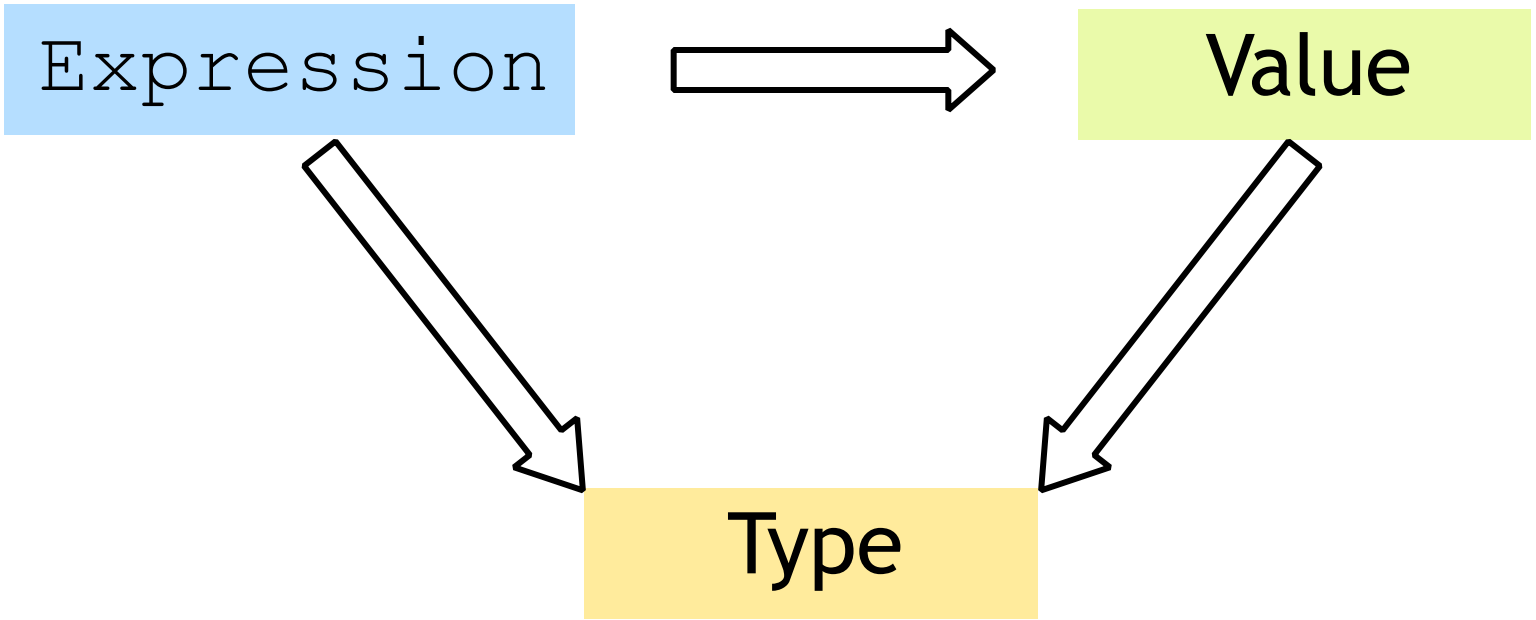
In general, `xs` may be empty (match failure!)

Another useful early warning

# What About …

# Functions

# Remember the Holy Trinity




Everything is an expression
Everything has a value
Everything has a type

Functions are values!

# Complex Type: Functions!

Parameter (formal)

Body Expr


<fun>

int -> int

```
# let inc = fun x -> x+1 ;;
val inc : int -> int = <fun>
# inc 0 ;;
- : int = 1
# inc 10 ;;
- : int = 11
```

How a call ("application") is evaluated:

1. Evaluate argument
2. Bind formal to arg value
3. Evaluate "Body expr"

# Complex Type: Functions!

Parameter (formal)

Body Expr




<fun>

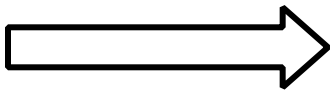
int -> int

Functions only have ONE parameter ?!?

# Solution #1: Simultaneous Binding

Parameter
(formal)

```
fun (x,y) -> x<y
```



<fun>

(int * int) -> bool

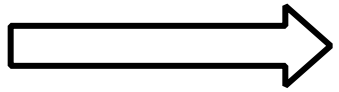
Functions only have
ONE parameter ?!?

# Solution #2: "Currying"

Parameter (formal)

Body Expr

```
fun x -> fun y-> x<y
```



<fun>

int -> (int -> bool)

Whoa! A function can return a function

```
# let lt = fun x -> fun y -> x < y ;;
val lt : int -> int -> bool = <fun>
# let is5Lt = lt 5 ;;
val is5lt : int -> bool = <fun>
# is5lt 10 ;;
- : bool = true
# is5lt 2 ;;
- : bool = false
```

# Clicker Question

What is the result of

```
(fun x -> not x) ?
```

(a) Syntax Error

(b) `<fun> : int -> int`

(c) `<fun> : int -> bool`

(d) `<fun> : bool -> int`

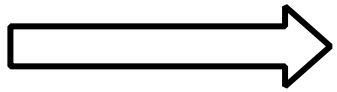
(e) `<fun> : bool -> bool`

# “Higher-Order” Function

Parameter (formal)

Body Expr

```
fun f -> fun x -> not(f x)
```



<fun>

(’a -> bool) -> (’a -> bool)

## A function can also take a function argument

```
# let neg = fun f -> fun x -> not (f x) ;;
val lt : int -> int -> bool = <fun>
# let is5gte = neg is5lt ;;
val is5gte : int -> bool = <fun>
# is5gte 10 ;;
- : bool = false
# is5gte 2 ;;
- : bool = true
(*… odd, even …*)
```

# Clicker Question

What is the result of

```
(fun f -> (fun x -> (f x) + x)) ?
```

(a) Syntax Error

(b) `<fun> : int -> int -> int`

(c) `<fun> : int -> int`

(d) `<fun> : (int -> int) -> int -> int`

# Shorthand for Function Binding

```
# let neg = fun f -> fun x -> not (f x);;
…

# let neg f x = not (f x) ;;
val neg : int -> int -> bool = <fun>

# let is5gte = neg is5lt ;;
val is5gte : int -> bool = <fun>
# is5gte 10 ;;
- : bool = false
# is5gte 2 ;;
- : bool = true
```

# A "Filter" Function

If `xs` "matches" ... then use

this pattern ... this Body Expr

```
# let rec filter f xs =
    match xs with
    | []       -> []
    | (x::xs') -> if f x
                  then x::(filter f xs')
                  else (filter f xs') ;;
val filter : ('a->bool)->'a list->'a list = <fun>
```

```
# let list1 = [1; 31; 12; 4; 7; 2; 10] ;;
# filter is5lt list1 ;;
- : int list = [31; 12; 7; 10]
# filter is5gte list1 ;;
- : int list = [1; 4; 2]
# filter even list1 ;;
- : int list = [12; 4; 2; 10]
```

# A “Partition” Function

```
# let partition f l = (filter f l, filter (neg f) l) ;;
val partition : ('a->bool)->'a list->'a list * 'a list


# let list1 = [1; 31; 12; 4; 7; 2; 10] ;;
- …

# partition is5lt list1 ;;
- : (int list * int list) = ([31;12;7;10],[1;4;2])

# partition even list1 ;;
- : (int list * int list) = ([12;4;2;10],[1;31;7])
```

# “Operators” are Functions

```
# 2 <= 3 ;;
- : bool = true
# “ba” <= “ab” ;;
- : bool = false

# let lt = (<) ;;
val lt : 'a -> 'a -> bool = <fun>

# lt 2 3 ;;
- : bool = true
# lt “ba” “ab” ;;
- : bool = false


# let is5Lt = lt 5 ;;
val is5lt : int -> bool = <fun>
# is5lt 10 ;;
- : bool = true
# is5lt 2 ;;
- : bool = false
```

# A “Quicksort” Function

```
let rec sort xs =
  match xs with
  | []     -> []
  | (h::t) -> let (l,r) = partition ((<) h) t in
              (sort l) @ (h::(sort r))
```

Now, let’s begin at the beginning …