Announcements

1. No PS this week
2. Online evaluations

1

1

Lecture 25
Call By Name
Call By Need

T. METIN SEZGIN

2

Learning outcomes of this lecture

- A student attending this lecture should be able to:
  1. Understand the philosophy of lazy evaluation
  2. Understand call by need and call by name and how they work
  3. Understand the uses of lazy evaluation
  4. Trace and CBNeed CBName evaluation using the env & store
  5. Implement CBNeed CBName

3

IREF -- Call-by-reference

```
let p = proc (x) set x = 4
in let a = 3
   in begin (p a); a end
```

Evaluates to 4

4

Nugget

In Call by Value, a copy of the argument is passed
In Call by Reference, address of variable is passed

5

Uses of call-by-reference

- Multiple return values

```
let swap = proc (x) proc (y)
            let temp = x
            in begin
                set x = y;
                set y = temp
               end
in let a = 33
   in let b = 44
      in begin
          ((swap a) b);
          -(a,b)
         end
```

6

Implementing CBR

- Expressed and denoted values remain the same
- Location allocation policy changes
  - If the formal parameter is a variable, pass on the reference
  - Otherwise, put the value of the formal parameter into the memory, pass a reference to it

$$
\begin{aligned}
ExpVal &= Int + Bool + Proc \\
DenVal &= Ref(ExpVal)
\end{aligned}
$$

```
(call-exp (rator rand)
  (let ((proc (expval->proc (value-of rator env)))
        (arg (value-of-operand rand env)))
    (apply-procedure proc arg)))
```

apply-procedure : $Proc \times Ref \to ExpVal$

```
(define apply-procedure
  (lambda (proc1 val)
    (cases proc proc1
      (procedure (var body saved-env)
        (value-of body
          (extend-env var val saved-env))))))
```

value-of-operand : $Exp \times Env \to Ref$

```
(define value-of-operand
  (lambda (exp env)
    (cases expression exp
      (var-exp (var) (apply-env env var))
      (else
        (newref (value-of exp env))))))
```

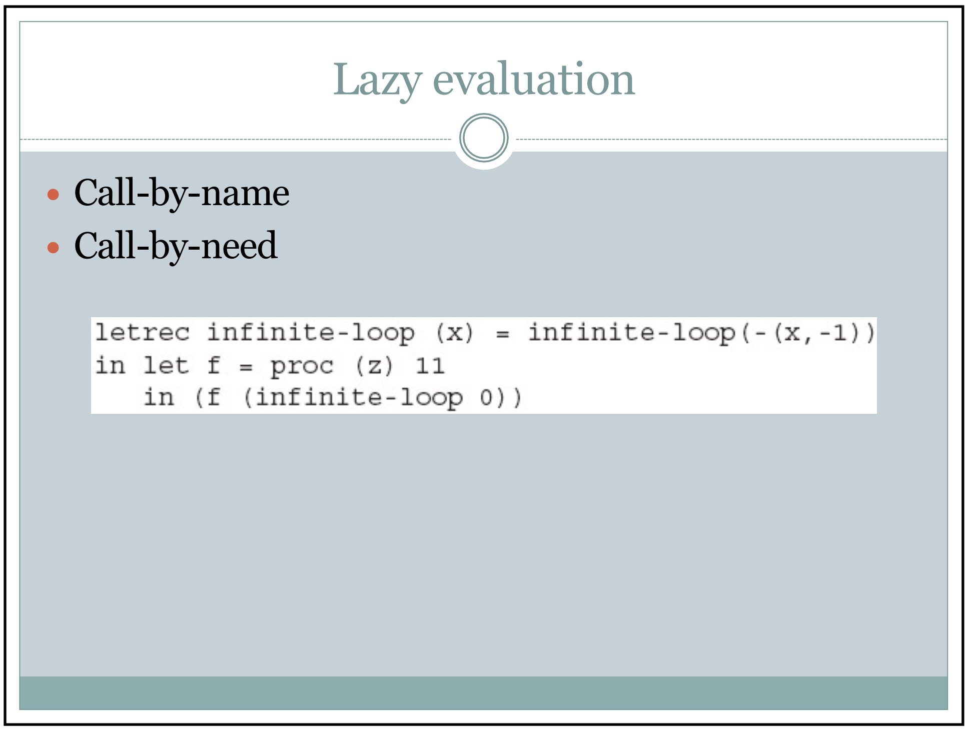
7

Another example

```
let b = 3
in let p = proc (x) proc(y)
                begin
                 set x = 4;
                 y
                end
   in ((p b) b)
```

- Here there is variable aliasing
- This evaluates to 4

8

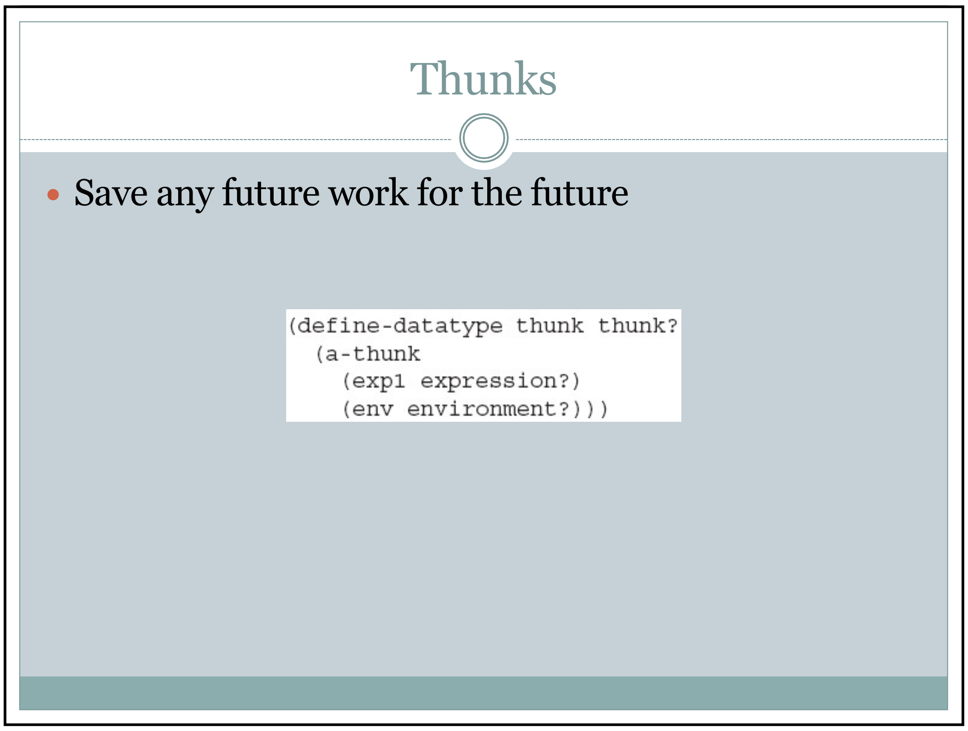


Lazy evaluation

- Call-by-name
- Call-by-need

```
letrec infinite-loop (x) = infinite-loop(-(x,-1))
in let f = proc (z) 11
   in (f (infinite-loop 0))
```

9

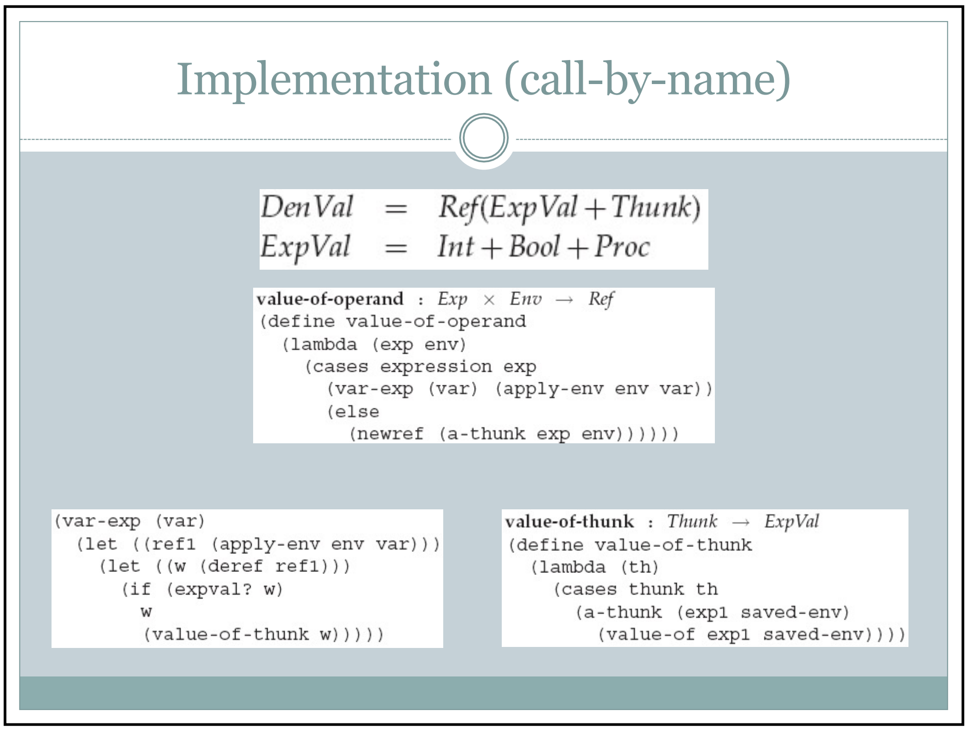


Thunks

- Save any future work for the future

```
(define-datatype thunk thunk?
  (a-thunk
    (exp1 expression?)
    (env environment?)))
```

10



Implementation (call-by-name)

$$
\begin{aligned}
DenVal &= Ref(ExpVal + Thunk) \\
ExpVal &= Int + Bool + Proc
\end{aligned}
$$

value-of-operand : $Exp \times Env \to Ref$

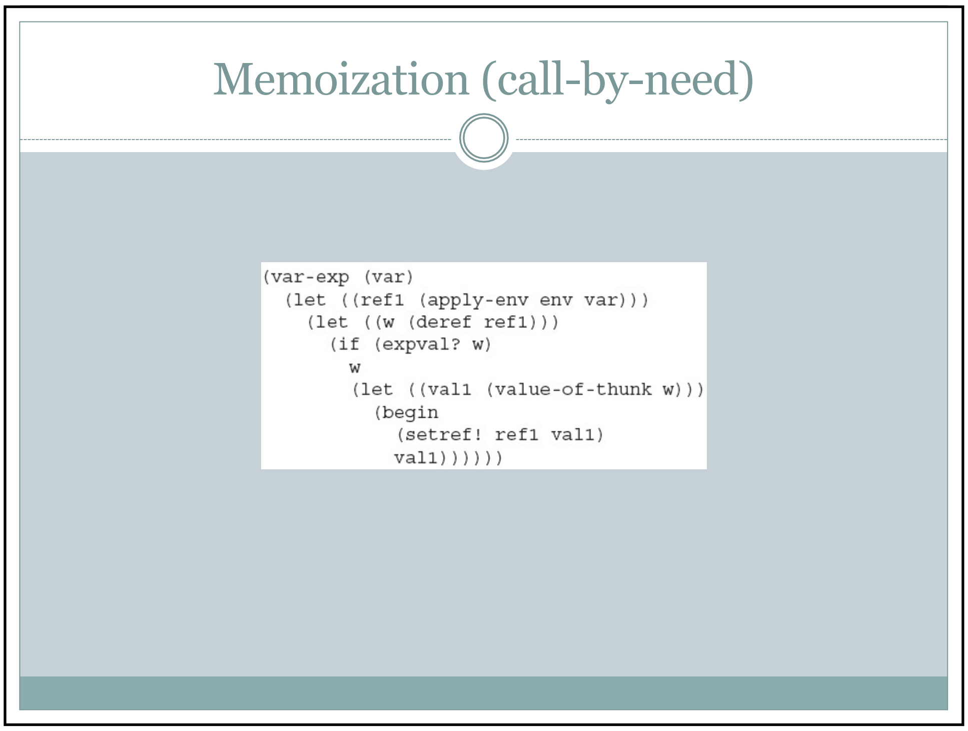
```
(define value-of-operand
  (lambda (exp env)
    (cases expression exp
      (var-exp (var) (apply-env env var))
      (else
        (newref (a-thunk exp env))))))
```

```
(var-exp (var)
  (let ((ref1 (apply-env env var)))
    (let ((w (deref ref1)))
      (if (expval? w)
        w
        (value-of-thunk w)))))
```

value-of-thunk : $Thunk \to ExpVal$

```
(define value-of-thunk
  (lambda (th)
    (cases thunk th
      (a-thunk (exp1 saved-env)
        (value-of exp1 saved-env)))))
```

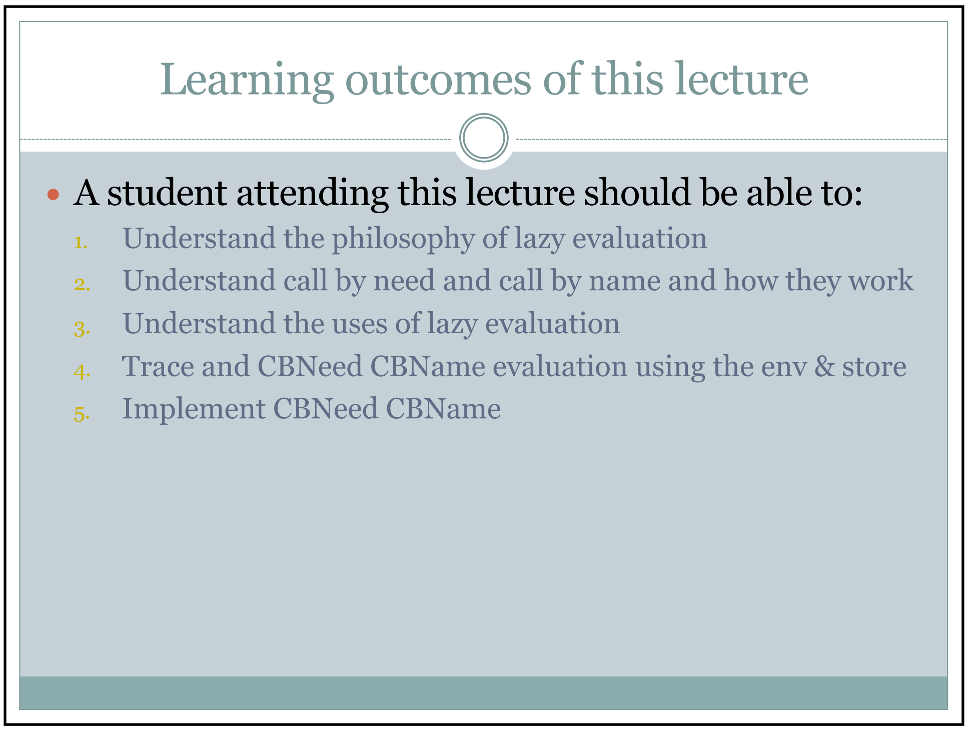
11



Memoization (call-by-need)

```
(var-exp (var)
  (let ((ref1 (apply-env env var)))
    (let ((w (deref ref1)))
      (if (expval? w)
        w
        (let ((val1 (value-of-thunk w)))
          (begin
            (setref! ref1 val1)
            val1))))))
```

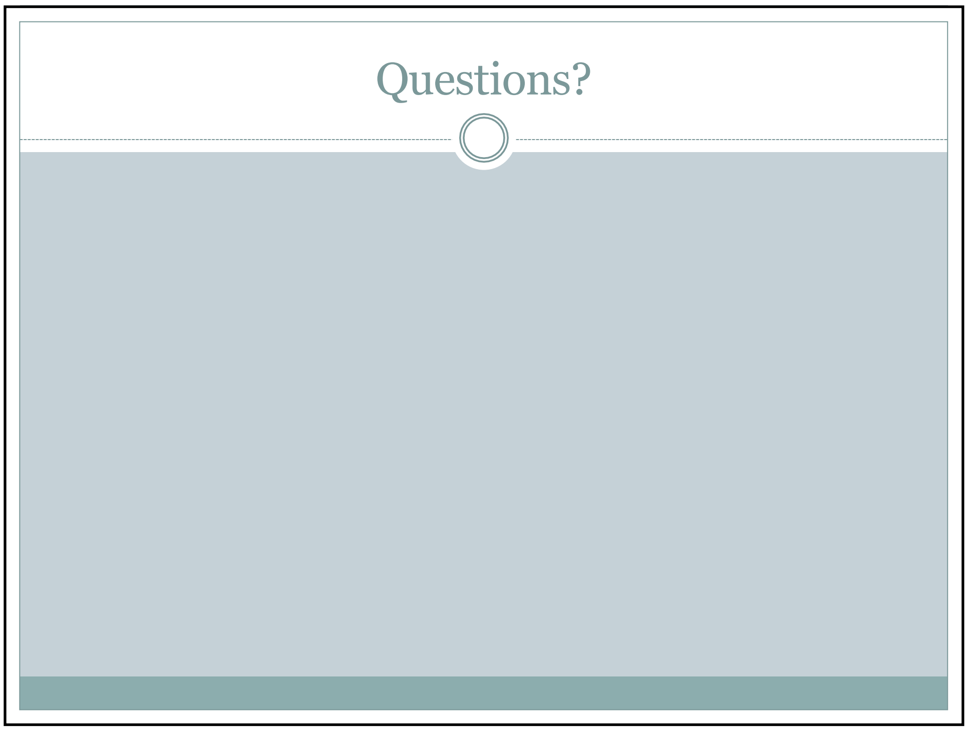
12



Learning outcomes of this lecture

- A student attending this lecture should be able to:
  1. Understand the philosophy of lazy evaluation
  2. Understand call by need and call by name and how they work
  3. Understand the uses of lazy evaluation
  4. Trace and CBNeed CBName evaluation using the env & store
  5. Implement CBNeed CBName

13



Questions?

14