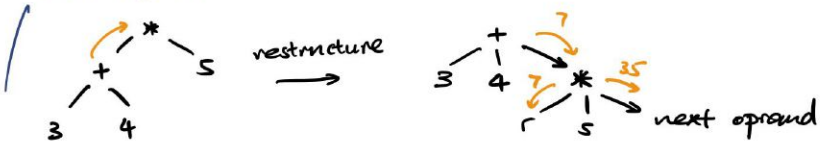
Lec 12

Continuation passing style (CPS) ⟷ (counterpart) direct style

Consider:

(3+4)*5




```
fun add x y k = k (x + y)
fun mult x y k = k (x * y)
```
] int → int → (int → 'a) → 'a

```
add 3 4 (fn r => mul r 5 Int.toString)
```
↑ continuation

* answer type: string (output type from continuation)
* initial continuation: whatever to do next

```
=> (fun r => mul r 5 Int.toString) (3+4)
=> (fun r => mul r 5 Int.toString) 7
=> [7/r] (fun r => mul r 5 Int.toString)
=> [7/r] (mul 7 5 Int.toString)
=> [7/r] (Int.toString (7*5))
=> Int.toString 35
=> "35"
```

* Ex in different styles

```
(* sum : int list -> int *)
fun sum [] = 0
  | sum x::xs = x + sum xs

fun sum L = foldl (op +) 0 L

val sum = fold (op +) 0

(* tsum : int list * int -> int
   ENS tsum(L, acc) ≅ sum L + acc
*)
fun tsum ([], acc) = acc
  | tsum (x::xs, acc) = tsum (xs, x + acc)
fun sum L = tsum(L, 0)
```

Familiar ways to write sum

CPS

```
(* csum : int list -> (int -> 'a) -> 'a
   REQ true
   ENS csum L k ≅ k (sum L)
*)
fun csum [] k = k 0
  | csum x::xs k = csum xs (fn r => k (x + r))
fun sum L = csum L (fn x => x)
```

Hmm everything's tail call, but not a way to improve performance. Compiler turns normal sum into CPS!

But notice we're building longer and longer k?

> Those inferior languages (...?)
> C doesn't pretend to be a programming language
> ... but python is not a real programming language
> but it pretends to be one ...

← random digression / joke

Exercise: trace csum [2,3] Int.toString

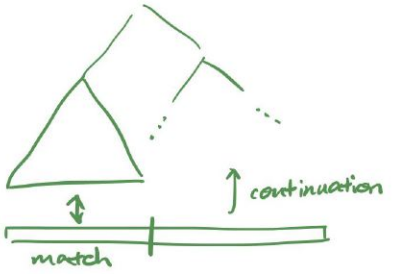
Back to trees

```
datatype 'a tree = Empty | Node of 'a tree * 'a * 'a tree

(* 'a tree → 'a list → 'a list *)
fun inorder Empty acc = acc
  | inorder (Node (L, x, R)) acc = inord L (x :: inorder R acc)

(* same int list → int list → bool *)
fun same [] [] = true
  | same x::xs y::ys = x = y andalso same xs ys
  | same _ _ = false
```

↑ able to stop early

```
(* treematch : int tree → int list → bool *)
fun treematch T L = same (inorder T []) L
```

⚠ bad performance if inorder before any comparison



CPS version — capable of early stopping

```
(* prefix : int tree → int list → (int list → bool) → bool
   REQ k is total
   ENS prefix T L k ⇒ { true  if L ≅ L1@L2 s.t. L1 is inord of T
                      |        and k L2 ↪ true
                      | false otherwise
*)
```

```
fun treematch T L = prefix T L (fn [] ⇒ true | _ ⇒ false)
```

built in as List.null — initial continuation

```
fun prefix (Empty : int tree) L k = k L
  | prefix (Node (l, x, r)) L k = prefix l L (fn [] ⇒ false
                                               | y::ys ⇒ x = y andalso prefix r ys k)
```

↑ tail recursive!

this is not call, just passed in as value

Ex. Polymorphise (...?)

Two continuations !

```
(* search  ('a -> bool) -> 'a tree -> ('a -> 'b) -> (unit 'b) -> 'b
   REQ  p total
    :
*)
fun search  p Empty  sc  fc  =  fc ()
  | search  p (Node(L, x, R))  sc  fc =
      if  p x  then  sc x
      else
          search p L sc (fn () => search p R sc fc)

fun findeven = search (fn n => n mod 2 = 0) T
                      (fn x => SOME x)
                      (fn () => NONE)
```

predicate

cont. success-callback

cont. failure-callback