Introduction to Functional Programming in *OCaml*

Roberto Di Cosmo, Yann Régis-Gianas, Ralf Treinen

Week 5 - Sequence 3: Mutable data structures: arrays

Revisiting the arrays

We have met the array data structure in the course on Week 2 - Sequence 3.

- fixed size
- direct access to the elements, via an index
- constant time for accessing any element
- particularly well adapted to loop constructs

Exercise

Use an array to find cubes which are squares

Find cubes that are squares I

```
let cubes n = Array.init n (fun i -> i*i*i);;
# val cubes : int -> int array = <fun>

let sqrti n =  truncate (sqrt (float n));;
# val sqrti : int -> int = <fun>

let issquare n = let s = sqrti n in s*s =n;;
# val issquare : int -> bool = <fun>
```

Find cubes that are squares II

```
let squarecubes n =
  let c = cubes n in
  for i = 0 to n-1 do
    if issquare c.(i) then
       (print_int c.(i);
        print_string "␣")
  done
;;
# val squarecubes : int -> unit = <fun>

squarecubes 100;;
# 0 1 64 729 4096 15625 46656 117649 262144 531441 - : unit = ()
```

Arrays are *mutable* data structures

OCaml's arrays are real arrays

- each cell of the array can be *modified in place*
- using the <- operator
- yes, the old value is lost!

Changing array contents I

```
let a = [|0;1;2;3;4|];;
# val a : int array = [|0; 1; 2; 3; 4|]

a.(0);;
# - : int = 0

a.(0) <- 100;;
# - : unit = ()

a.(0);;
# - : int = 100

a;;
# - : int array = [|100; 1; 2; 3; 4|]
```

Changing array contents II

```
let rotate a =
  let n = Array.length a in
  let v = a.(0) in
  for i = 0 to n-2 do
    a.(i) <- a.(i+1)
  done;
  a.(n-1)<-v;;
# val rotate : 'a array -> unit = <fun>

let x = Array.init 10 (fun i -> i);;
# val x : int array = [|0; 1; 2; 3; 4; 5; 6; 7; 8; 9|]
```

Changing array contents III

```
x;;
# - : int array = [|0; 1; 2; 3; 4; 5; 6; 7; 8; 9|]

rotate x;;
# - : unit = ()

x;;
# - : int array = [|1; 2; 3; 4; 5; 6; 7; 8; 9; 0|]

rotate x;;
# - : unit = ()

x;;
# - : int array = [|2; 3; 4; 5; 6; 7; 8; 9; 0; 1|]
```

The update operator <-

In place modification: `e1 <- e2`

- the expression `e1` denoting a mutable value is evaluated
- *the type checker ensures that* `e1` *is a mutable value*
- the mutable value is modified in place with the new value `e2`
- the type of the update operation is `unit`

Summary

- The `array` data type is actually *mutable*.
- The update operator `<-` modifies in place the cells of the arrays.