Programmazione Imperativa

Array

Contengono tutti elementi dello stesso tipo, hanno dimensione fissata al momento della creazione (come in C, Java, . . .) e possono essere acceduti tramite indici numerici (non sono associativi)

Un array può essere definito in due modi:

- **Letterale**

  Nella definizione di un nuovo array, gli elementi vengono posti tra `[| . . . |]`

  ```
  let arr = [|3;5;2|] ;;
  ```

  ```
  val arr : int array = [|3; 5; 2|]
  ```

- Tramite `Array.make`
  Con parametri la dimensione e il valore iniziale

  ```
  Array.make 10 0;;
  ```

  ```
  
  ```

La lunghezza di un array può essere ottenuta tramite:

```
let n = Array.length a;;
```

```
val n : int = 3
```

Si può accedere agli elementi tramite:

- Indici (0, . . . , n-1) con la sintassi `.(i)`

```
let e = a.(1) ;;
```

```
val e : int = 5
```

- Scrittura, con la sintassi `<-`

```
a.(1) <- 6 ;;
a;;
```

```
- : unit = ()
- : int array = [|3; 6; 2|]
```

Questo costrutto è un **comando di assegnamento**

Record Mutabili

E' possibile rendere mutabili uno o più campi tramite il modificatore `mutable`

```
type persona =
{
    nome: string;
    cognome: string;
    mutable eta: int;
}
```

```
type persona = { nome : string; cognome : string; mutable eta : int; }
```

```
let mario = {nome="mario"; cognome="rossi"; eta=30 } ;;
```

```
val mario : persona = {nome = "mario"; cognome = "rossi"; eta = 30}
```

Anche in questo caso, l'assegnamento si può effettuare tramite costrutto `<-`

```
mario.eta <- 31 ;
```

Riferimenti

Oltre ad array e record mutabili, è possibile definire anche singole variabili mutabili usando il tipo `ref`, ad esempio:

```
let x = ref 12 ;;
```

```
val x : int ref = {contents = 12}
```

💡 E' possibile accedere alla variabile ref come record

```
x.contents ;;
```

```
- : int = 12
```

Ma in realtà abbiamo una sintassi specifica:

```
!x ;; (* corrisponde a x.contents *)
x := 14 ;; (* corrisponde a x.contents <- 14 *)
```

```
- : int = 13
- : unit = ()
```

L'operazione tramite `!` è detta ***dereferenziazione***

```
let x = ref 0;;
let y = x;;

y:=100;;
!x;;
```

```
val x : int ref = {contents = 0}
val y : int ref = {contents = 0}

- : unit = ()
- : int = 100
```

Sequenze di comandi e cicli

Una sequenza di comandi/espressioni può essere definita tramite il separatore `;`

```
let x = ref 0;;
let y = x;;

x := !x+1 ; y := !y+1 ; (!x,!y) ;; (* sequenza di espressioni *)
```

```
val x : int ref = {contents = 0}
val y : int ref = {contents = 0}

- : int * int = (2, 2)
```

Le espressioni vengono valutate una dopo l'altra e l'ultima fornisce il risultato finale

Cicli `for`

La sintassi del comando `for` è la seguente:

```
for <variable> = <start> to <end> do
    ...
done
```

oppure

```
for <variable> = <start> downto <end> do
    ...
done
```

dove `<start>` e `<end>` sono valori interi entro cui far variare la `<variabile>`

```
for i = 1 to 10 do
    print_endline (string_of_int i)
done ;;
```

```
1
2
3
4
5
6
7
8
9
10

- : unit = ()
```

Cicli `while`

La sintassi del comando while è la seguente:

```
while <condizione> do
    ...
done
```

dove `<condizione>` è una espressione di tipo `boolean`

```
let x=ref 10 in
while !x>0 do
    print_endline (string_of_int !x) ;
    x := !x/2
done
```

```
10
5
2
1

- : unit = ()
```

In OCaml non è consigliato usare i cicli poiché

- la sintassi non è molto friendly
- non è possibile interromperli brutalmente (non esiste `break` o `return`)
- il for non è flessibile sull'uso degli indici (no incremento di 2 per volta)

Eccezioni

Le eccezioni sono un modo usato in molti linguaggi di programmazione per gestire le situazioni anomale e di errore nei programmi

Esistono eccezioni predefinire di sistema ma possiamo anche definirne di nuove noi, ad esempio:

```
exception Lista_vuota ;;
exception Stringa_errata of string ;;
```

```
exception Lista_vuota
exception Stringa_errata of string
```

Possono essere sollevate tramite
`raise`

```
raise Lista_vuota;;
```

```
Exception: Lista_vuota.
Called from file "toplevel/toploop.ml", line 208, characters 17-27
```

```
raise (Stringa_errata "test") ;;
```

```
Exception: Stringa_errata "test".
Called from file "toplevel/toploop.ml", line 208, characters 17-27
```

e possono essere usate, ad esempio, per interrompere una funzione in caso di errore

```
let giorno_num g =
    match g with
     | "lun" -> 1 | "mar" -> 2 | "mer" -> 3 | "gio" -> 4
     | "ven" -> 5 | "sab" -> 6 | "dom" -> 7
     | _ -> raise (Stringa_errata g) ;;
```

Inoltre, è possibile intercettare e gestire eccezioni con il costrutto `try ... with`

```
try <espressione> with
| <pat1> -> <esp1>
| <pat2> -> <esp2>
...
| <patN> -> <espN>
```