

## Mise à Niveau et Ouverture (STL)

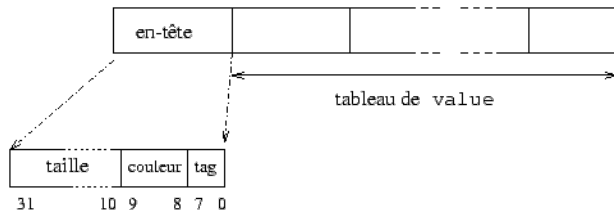


Emmanuel Chailloux

# Représentation des données en OCaml

**Polymorphisme paramétrique:** représentation uniforme

- ▶ 1 mot mémoire
  - ▶ valeurs immédiates (*int*, *char*, constructeurs constants)
  - ▶ autres valeurs : pointeur vers le tas (zone d'allocation dynamique)
- ▶ == teste l'égalité sur le mot (valeurs immédiate ou pointeur)



# GC en Objective Caml

à **générations**: 2 générations (ancienne et nouvelle)

- ▶ Stop&Copy sur la nouvelle (GC mineur)
- ▶ Mark&Sweep incémental sur l'ancienne génération (GC majeur)

**caractéristiques:**

- ▶ Un objet jeune qui survit à 1 GC change de zone.
- ▶ Conservation des pointeurs de zone ancienne vers zone jeune
- ▶ Si le Mark&Sweep échoue, alors un GC compactant est déclenché pour la génération ancienne.

Le module Gc permet de contrôler les paramètres du GC.

# Module Gc (1)

- ▶ statistiques (type *stat* :
  - ▶ `Gc.stat : unit -> Gc.stat`
- ▶ contrôler les paramètres du tas (type *control*)
  - ▶ `Gc.get : unit -> Gc.control+`  
`Gc.set : Gc.control -> unit`
- ▶ forcer le GC :
  - ▶ `Gc.minor() : unit -> unit`
  - ▶ `Gc.major() : unit -> unit`
  - ▶ `Gc.compact() : unit -> unit`

## Module Gc (2)

### Statistiques:

```
1 # Gc.stat();;
2 - : Gc.stat =
3 {Gc.minor_words = 112065.; promoted_words = 0.;
4  major_words = 60074.; minor_collections = 0;
5  major_collections = 0; heap_words = 126976;
6  heap_chunks = 1; live_words = 60074;
7  live_blocks = 11226; free_words = 66902;
8  free_blocks = 1; largest_free = 66902;
9  fragments = 0; compactions = 0;
10 top_heap_words = 126976; stack_size = 53}
```

et fonction d'affichage :

```
print_stat : out_channel -> unit
```

## Module Gc (3)

```
1 # Gc.stat();;
2 - : Gc.stat =
3 {Gc.minor_words = 680793.; promoted_words = 7360.;
4  major_words = 117587.; minor_collections = 2;
5  major_collections = 4; heap_words = 126976;
6  heap_chunks = 1; live_words = 106362;
7  live_blocks = 23795; free_words = 20614;
8  free_blocks = 1; largest_free = 20614;
9  fragments = 0; compactions = 2;
10 top_heap_words = 126976; stack_size = 53}
11 # Gc.major();;
12 - : unit = ()
13 # Gc.stat() ;;
14 - : Gc.stat =
15 {Gc.minor_words = 702675.; promoted_words = 7360.;
16  major_words = 118290.; minor_collections = 2;
17  major_collections = 5; heap_words = 126976;
18  heap_chunks = 1; live_words = 106671;
19  live_blocks = 23880; free_words = 20305;
20  free_blocks = 18; largest_free = 19911;
21  fragments = 0; compactions = 2;
22  top_heap_words = 126976; stack_size = 53}
```

# Module Gc (4)

## Contrôle:

```
1 # let c = Gc.get () ;;
2 val c : Gc.control =
3   {Gc.minor_heap_size = 262144;
4     major_heap_increment = 126976;
5     space_overhead = 80;
6     verbose = 0;
7     max_overhead = 500;
8     stack_limit = 1048576;
9     allocation_policy = 0}
```

## Module Gc (5)

Par exemple, le champ `verbose` peut prendre des valeurs de 0 à 127 activant 7 indicateurs différents.

```
1 # c.Gc.verbose <- 127 ;;
2 - : unit = ()
3 # Gc.set c ;;
4 - : unit = ()
5 # Gc.compact () ;;
```

**affichage:**

```
1 Heap compaction requested
2 <>Sweeping 9223372036854775807 words
3 Starting new major GC cycle
4 Marking 9223372036854775807 words
5 Subphase = 10
6 Sweeping 9223372036854775807 words
7 Compacting heap...
8 done.
```

Les différentes phases du GC sont indiquées ainsi que le nombre d'objets traités.



# Mesure de taille de valeurs OCaml (1)

- ▶ calcul sur les valeurs vivantes entre 2 Gc encadrant un calcul (si calcul pur)
- ▶ exploration d'une valeur en tenant compte du partage

fonction de test : intervalle

```
1 let rec i a b =  
2   if a > b then []  
3   else a :: (i (a+1) b);;
```

## Mesure de taille de valeurs OCaml (2)

calcul sur les objets vivants :

```
1 let mesure f x =  
2   Gc.compact ();  
3   let s1 = Gc.stat() in  
4     let u = f x in  
5     Gc.compact();  
6     let s2= Gc.stat() in  
7     (u,s1,s2) ;;  
8  
9 let _,b,c = mesure (i 1) 1000;;
```

## Mesure de taille de valeurs OCaml (3)

résultats :

```
1 # let _,b,c = mesure (i 1) 1000;;
2 val b : Gc.stat =
3   {Gc.minor_words = 1048056.; promoted_words = 169833.;
4     major_words = 330159.; minor_collections = 19; major_collections = 18;
5     heap_words = 491520; heap_chunks = 1; live_words = 269812;
6     live_blocks = 66474; free_words = 221708; free_blocks = 1;
7     largest_free = 221708; fragments = 0; compactions = 9;
8     top_heap_words = 491520; stack_size = 73}
9 val c : Gc.stat =
10  {Gc.minor_words = 1051079.; promoted_words = 172856.;
11    major_words = 333182.; minor_collections = 21; major_collections = 20;
12    heap_words = 491520; heap_chunks = 1; live_words = 272835;
13    live_blocks = 67478; free_words = 218685; free_blocks = 1;
14    largest_free = 218685; fragments = 0; compactions = 10;
15    top_heap_words = 491520; stack_size = 75}
16 # c.promoted_words -. b.promoted_words;;
17 - : float = 3023.
18 # c.live_words - b.live_words;;
19 - : int = 3023
```

## Mesure de taille de valeurs OCaml (4)

créer ou utiliser une fonction d'exploration : bibliothèque Obj

```
1 # Obj.reachable_words;;  
2 - : Obj.t -> int = <fun>  
3 # Obj.repr;;  
4 - : 'a -> Obj.t = <fun>
```

```
1 # let mesure2 f x =  
2     let u = f x in  
3     u,Obj.reachable_words (Obj.repr u) ;;  
4 val mesure2 : ('a -> 'b) -> 'a -> 'b * int = <fun>  
5  
6 # let _,r=mesure2 (i 1) 1000;;  
7 val r : int = 3000
```