

Algorithmique - Programmation 1

Cours 10

Université Henri Poincaré

CESS Epinal

Automne 2008

MCours.com



Plan

Cam1 fonctionnel vs impératif

Le type unit

Données mutables

Les structures de contrôle (boucles)

Les tableaux

Les entrées/sorties

Interprétation vs compilation

Cam1 fonctionnel vs impératif

► Cam1 fonctionnel :

- Liens entre noms et valeurs

```
# let a = 5;;
```

- Fonctions produisant de nouvelles valeurs à partir de valeurs d'entrée (paramètres)

```
# let b = (min a 4);;
```

Cam1 fonctionnel vs impératif

▶ Cam1 fonctionnel :

- Liens entre noms et valeurs

```
# let a = 5;;
```

- Fonctions produisant de nouvelles valeurs à partir de valeurs d'entrée (paramètres)

```
# let b = (min a 4);;
```

▶ Remarques :

- On ne peut que définir de nouveaux liens via `let`, on ne peut pas **modifier** un lien existant

- Toute expression a une valeur **typée**

```
# let c = min a (if (b<5) then 0 else 1);;
```

Cam1 fonctionnel vs impératif

► Cam1 impératif :

- Comment interagir avec l'environnement ?
(valeur de retour d'un appel système ?)
- Comment modifier des variables ?
(exemple : modification d'une liste passée en paramètre d'une fonction)

Plan

Cam1 fonctionnel vs impératif

Le type unit

Données mutables

Les structures de contrôle (boucles)

Les tableaux

Les entrées/sorties

Interprétation vs compilation

Références bibliographiques

Exercices

Le type unit

- ▶ Type de donnée ayant une seule valeur : ()

```
# ();;
```

```
- : unit = ()
```

- ▶ Symbolise le type *vide*

- ▶ Type utilisé comme type de retour des opérations à effet de bord (tels que les appels systèmes) :

```
# print_int 5;;
```

```
5- : unit = ()
```

```
# print_string "hello world!\n";;
```

```
hello world!
```

```
- : unit = ()
```

Le type unit (suite)

- ▶ Pour enchaîner les expressions de type unit, on utilise le ;

- ▶ Exemple :

```
# print_string "saisissez un nombre:" ;
  let i = read_int() in print_int i;
  print_newline() ;;
saisissez un nombre:12
12
- : unit = ()
```

- ▶ Il est possible de grouper une séquence d'expressions entre les mots-clés begin et end
- ▶ Remarque: une séquence d'expressions peut se terminer par une expression de type \neq unit, cette dernière expression sera le type de la séquence

Plan

Cam1 fonctionnel vs impératif

Le type unit

Données mutables

Les structures de contrôle (boucles)

Les tableaux

Les entrées/sorties

Interprétation vs compilation

Références bibliographiques

Exercices

Données mutables

- ▶ Lien nom-valeur **modifiable**
(inutile de recréer un lien avec `let`)
- ▶ Notion de **pointeur** sur un emplacement mémoire
- ▶ On parle également de **référence** (adresse mémoire)
- ▶ Utilisation des références en Caml :
 - création

```
# let x = ref 5;;  
val x : int ref = {contents = 5}
```
 - écriture

```
# x := 6;;  
- : unit = ()
```
 - lecture

```
# !x ;;  
- : int = 6
```

Données mutables (suite)

- ▶ Remarque: les références sont **typées**
- ▶ Référence sur un entier \neq référence sur une liste:

```
# let x = ref [2.3 ; 3.6];;  
val x : float list ref = {contents = [2.3; 3.6]}  
# List.hd (!x);;  
- : float = 2.3
```

Plan

Cam1 fonctionnel vs impératif

Le type unit

Données mutables

Les structures de contrôle (boucles)

Les tableaux

Les entrées/sorties

Interprétation vs compilation

Références bibliographiques

Exercices

Les structures de contrôle (boucles)

- ▶ Boucle *pour* (for):

```
for compteur entier = debut to fin do
    sequence d'actions (type unit)
done
```

```
for compteur entier = debut downto fin do
    sequence d'actions (type unit)
done
```

- ▶ Boucle *tant que* (while):

```
while test vrai do
    sequence d'actions (type unit)
done
```

Les structures de contrôle (suite)

- ▶ Exemples d'utilisation :

```
# for i = 1 to 5 do
    print_int i ; print_string " ";
done;;
1 2 3 4 5 - : unit = ()
```

MCours.com

Les structures de contrôle (suite)

► Exemples d'utilisation :

```
# for i = 1 to 5 do
    print_int i ; print_string " ";
done;;
```

```
1 2 3 4 5 - : unit = ()
```

```
# for i = 5 downto 1 do
    print_int i ; print_string " ";
done;;
```

```
5 4 3 2 1 - : unit = ()
```

Les structures de contrôle (suite)

► Exemples d'utilisation :

```
# for i = 1 to 5 do
  print_int i ; print_string " ";
done;;
1 2 3 4 5 - : unit = ()

# for i = 5 downto 1 do
  print_int i ; print_string " ";
done;;
5 4 3 2 1 - : unit = ()

# let r = ref 1 in
  while (!r < 6) do
    print_int !r ; print_string " ";
    r := !r + 1;
  done;;
1 2 3 4 5 - : unit = ()
```


Plan

Caml fonctionnel vs impératif

Le type unit

Données mutables

Les structures de contrôle (boucles)

Les tableaux

Les entrées/sorties

Interprétation vs compilation

Références bibliographiques

Exercices

Les tableaux

- ▶ Structure de données où les éléments sont indexés par un nombre entier (position dans le tableaux)

Les tableaux

- ▶ Structure de données où les éléments sont indexés par un nombre entier (position dans le tableaux)
- ▶ Création d'un tableau :

```
# Array.make 5 'a';;
```

```
val t : char array = [|'a'; 'a'; 'a'; 'a'; 'a'|]
```

```
# let u = [|'t'; 'o'; 't'; 'o'|];;
```

```
val u : char array = [|'t'; 'o'; 't'; 'o'|]
```

Les tableaux

- ▶ Structure de données où les éléments sont indexés par un nombre entier (position dans le tableaux)

- ▶ Création d'un tableau :

```
# Array.make 5 'a';;  
val t : char array = [|'a'; 'a'; 'a'; 'a'; 'a'|]  
  
# let u = [|'t'; 'o'; 't'; 'o'|];;  
val u : char array = [|'t'; 'o'; 't'; 'o'|]
```

- ▶ Accès en lecture à un élément d'un tableau :

```
# u.(1);;  
- : char = 'o'
```

Les tableaux

- ▶ Structure de données où les éléments sont indexés par un nombre entier (position dans le tableaux)
- ▶ Création d'un tableau :

```
# Array.make 5 'a';;
val t : char array = [|'a'; 'a'; 'a'; 'a'; 'a'|]
# let u = [|'t'; 'o'; 't'; 'o'|];;
val u : char array = [|'t'; 'o'; 't'; 'o'|]
```

- ▶ Accès en lecture à un élément d'un tableau :

```
# u.(1);;
- : char = 'o'
```

- ▶ Accès en écriture à un élément d'un tableau :

```
# u.(1) <- 'a';;      - : unit = ()
# u;                  - : char array = [|'t'; 'a'; 't'; 'o'|]
```

Les tableaux (suite)

- ▶ Taille d'un tableau: `Array.length`
- ▶ Concaténation de tableaux: `Array.append`
- ▶ Copie d'un tableau: `Array.copy`
- ▶ Dans un tableau sont stockées des valeurs (types de base) ou des références (types complexes), exemple :

Les tableaux (suite)

- ▶ Taille d'un tableau: `Array.length`
- ▶ Concaténation de tableaux: `Array.append`
- ▶ Copie d'un tableau: `Array.copy`
- ▶ Dans un tableau sont stockées des valeurs (types de base) ou des références (types complexes), exemple :

```
# let v = [| 0; 0|];;  
val v : int array = [|0; 0|]
```

Les tableaux (suite)

- ▶ Taille d'un tableau: `Array.length`
- ▶ Concaténation de tableaux: `Array.append`
- ▶ Copie d'un tableau: `Array.copy`
- ▶ Dans un tableau sont stockées des valeurs (types de base) ou des références (types complexes), exemple:

```
# let v = [| 0; 0 |];;  
val v : int array = [|0; 0|]  
  
# let w = [| v ; v |];;  
val w : int array array = [| [|0; 0|]; [|0; 0|] |]
```


Les tableaux (suite)

- ▶ Taille d'un tableau: `Array.length`
- ▶ Concaténation de tableaux: `Array.append`
- ▶ Copie d'un tableau: `Array.copy`
- ▶ Dans un tableau sont stockées des valeurs (types de base) ou des références (types complexes), exemple:

```
# let v = [| 0; 0 |];;  
val v : int array = [|0; 0|]  
  
# let w = [| v ; v |];;  
val w : int array array = [| [|0; 0|]; [|0; 0|] |]  
  
# v.(0) <- 1;  
- : unit = ()
```

Les tableaux (suite)

- ▶ Taille d'un tableau: `Array.length`
- ▶ Concaténation de tableaux: `Array.append`
- ▶ Copie d'un tableau: `Array.copy`
- ▶ Dans un tableau sont stockées des valeurs (types de base) ou des références (types complexes), exemple:

```
# let v = [| 0; 0 |];;
val v : int array = [|0; 0|]

# let w = [| v ; v |];;
val w : int array array = [| [|0; 0|]; [|0; 0|] |]

# v.(0) <- 1;;
- : unit = ()

# w;;
- : int array array = [| [|1; 0|]; [|1; 0|] |]
```

Plan

Cam1 fonctionnel vs impératif

Le type unit

Données mutables

Les structures de contrôle (boucles)

Les tableaux

Les entrées/sorties

Interprétation vs compilation

Références bibliographiques

Exercices

Les entrées/sorties

- ▶ **Interactions** entre un programme et le système d'exploitation de l'ordinateur :

affichages, saisies au clavier, écriture / lecture de fichiers,
etc

- ▶ Procédures (fonctions de type unit) principales :

- `# read_int;;`
- : `unit -> int = <fun>`
- `# print_int ;;`
- : `int -> unit = <fun>`
- `print_newline ;;`
- : `unit -> unit = <fun>`

Les entrées/sorties (suite)

► Ouverture / fermeture d'un **canal** :

- # open_in;;
- : string -> in_channel = <fun>
- # open_out ;;
- : string -> out_channel = <fun>
- # close_in ;;
- : in_channel -> unit = <fun>

► Lecture / écriture dans un canal :

- # input ;;
- : in_channel -> string -> int -> int -> int
- # output ;;
- : out_channel -> string -> int -> int -> unit
- # input_line ;;
- : in_channel -> string = <fun>

Les entrées/sorties : exemple

```
# let monFichier = open_out "toto";;
val monFichier : out_channel = <abstr>
# output monFichier "let a = [| 1 |] ;;";;
- : int -> int -> unit = <fun>
# output monFichier "let a = [| 1 |] ;;" 0 17;;
- : unit = ()
# close_out monFichier ;;
- : unit = ()
# exit 1;;
```

```
$ less toto.ml
let a = [| 1 |] ;
```

Plan

Cam1 fonctionnel vs impératif

Le type unit

Données mutables

Les structures de contrôle (boucles)

Les tableaux

Les entrées/sorties

Interprétation vs compilation

Références bibliographiques

Exercices

Interprétation vs compilation

- ▶ **Interprétation** d'un programme Caml : **exécution** du code Caml **par un programme spécifique** (top-level)
- ▶ **Compilation** d'un programme Caml : **traduction** du code Caml **en code pour une machine virtuelle** (bytecode) ou pour le **processeur** (assembleur)
- ▶ La **compilation** d'un programme Caml écrit dans un fichier texte, **produit un fichier exécutable** par la machine

- ▶ **Utilisation** du compilateur Caml :

```
$ ocamlc -o out.exe code.ml
```

```
$ ./out.exe
```

- ▶ NB : cette compilation crée des fichiers `.cmo` et `.cmi`

Interprétation vs compilation (suite)

- ▶ Structure d'un programme Caml :

```
(* Definition de constantes globales *)
```

```
let x = ... ;;
```

```
(* Definition de fonctions *)
```

```
let f x = ... ;;
```

```
(* Definition de la fonction principale *)
```

```
let main() = ... ;;
```

```
(* Point de depart du programme *)
```

```
let _ = ... ; main() ; ... ; exit 0;;
```

Référence bibliographique

- ▶ Développement d'applications avec Objective Caml

Emmanuel CHAILLOUX - Pascal MANOURY - Bruno PAGANO

Disponible en ligne :

<http://www.pps.jussieu.fr/Livres/ora/DA-OCAML/>

Exercices

1. Ecrire un programme Caml choisissant un nombre aléatoire entre 1 et 12, et demandant à l'utilisateur un nombre. Si ce nombre est plus grand que le nombre choisi, le programme affiche trop, sinon pas assez jusqu'à ce que l'utilisateur trouve.
2. Ecrire un programme Caml qui recherche l'élément minimum d'un tableau.

MCours.com