## Introduction

 Pour le moment, nous nous sommes limités aux aspects fonctionnels d'OCaml et donc aux variables et aux structures de données non mutables.

## Champ mutable d'un enregistrement

#### Introduction

- Pour le moment, nous nous sommes limités aux aspects fonctionnels d'OCaml et donc aux variables et aux structures de données non mutables.
- Cependant, OCaml est un langage de programmation multi-paradigme et la programmation impérative (et donc les variables mutables) peuvent être manipulées en OCaml.

## Champ mutable d'un enregistrement

# C9 OCaml: aspects fonctionnels

1. Variables mutables

#### Introduction

- Pour le moment, nous nous sommes limités aux aspects fonctionnels d'OCaml et donc aux variables et aux structures de données non mutables.
- Cependant, OCaml est un langage de programmation multi-paradigme et la programmation impérative (et donc les variables mutables) peuvent être manipulées en OCaml.

## Champ mutable d'un enregistrement

On peut déclarer en OCaml un enregistrement ayant des champs mutables grâce au mot-clé mutable.

#### Introduction

- Pour le moment, nous nous sommes limités aux aspects fonctionnels d'OCaml et donc aux variables et aux structures de données non mutables.
- Cependant, OCaml est un langage de programmation multi-paradigme et la programmation impérative (et donc les variables mutables) peuvent être manipulées en OCaml.

## Champ mutable d'un enregistrement

On peut déclarer en OCaml un enregistrement ayant des champs mutables grâce au mot-clé mutable. Par exemple,

```
type variable = {mutable valeur : int};;
```

# C9 OCaml : aspects fonctionnels

1. Variables mutables

#### Introduction

- Pour le moment, nous nous sommes limités aux aspects fonctionnels d'OCaml et donc aux variables et aux structures de données non mutables.
- Cependant, OCaml est un langage de programmation multi-paradigme et la programmation impérative (et donc les variables mutables) peuvent être manipulées en OCaml.

## Champ mutable d'un enregistrement

On peut déclarer en OCaml un enregistrement ayant des champs mutables grâce au mot-clé mutable. Par exemple,

```
type variable = {mutable valeur : int};;
```

▲ Pour modifier la valeur du champ on utilise <-. Le symbole = est réservé à la comparaison.

## Exemple

```
(* Type "variable" ayant un champ valeur mutable*)
   type variable = {mutable valeur : int};;
   (* on crée une variable ayant son champ valeur à 42 *)
   let u = {valeur = 42}::
   (* comme ce champ est mutable, on change la valeur avec <-*)
   u.valeur <- 20;
   (* cette expression renvoie unit et modifie la valeur de u*)
10
   (* Affichera 20 *)
11
   print_int u.valeur
12
```



#### Les références

• Le type ref est prédéfini dans OCaml et correspond exactement à ce que nous venons de faire (sauf que le champ mutable s'appelle contents)



#### Les références

 Le type ref est prédéfini dans OCaml et correspond exactement à ce que nous venons de faire (sauf que le champ mutable s'appelle contents)
 Par exemple let a = {contents = 5}; crée une variable ayant son champ mutable contents qui vaut 5.

# C9 OCaml : aspects fonctionnels

1. Variables mutables

- Le type ref est prédéfini dans OCaml et correspond exactement à ce que nous venons de faire (sauf que le champ mutable s'appelle contents)
   Par exemple let a = {contents = 5}; crée une variable ayant son champ mutable contents qui vaut 5.
- Afin d'alléger la syntaxe, on peut écrire directement let a = ref 5;;.

# C9 OCaml : aspects fonctionnels

1. Variables mutables

- Le type ref est prédéfini dans OCaml et correspond exactement à ce que nous venons de faire (sauf que le champ mutable s'appelle contents)
   Par exemple let a = {contents = 5}; crée une variable ayant son champ mutable contents qui vaut 5.
- Afin d'alléger la syntaxe, on peut écrire directement let a = ref 5;;.
   On retiendra que l'effet reste le même : on a crée une variable a ayant un champ mutable entier qui contient 5.

- Le type ref est prédéfini dans OCaml et correspond exactement à ce que nous venons de faire (sauf que le champ mutable s'appelle contents)
   Par exemple let a = {contents = 5}; crée une variable ayant son champ mutable contents qui vaut 5.
- Afin d'alléger la syntaxe, on peut écrire directement let a = ref 5;;.
   On retiendra que l'effet reste le même : on a crée une variable a ayant un champ mutable entier qui contient 5.
- Si on veut accéder à la valeur du champ mutable, une syntaxe "adoucie" est aussi disponible avec !. On écrira par exemple print\_int !a pour afficher le contenu du champ mutable de a.

- Le type ref est prédéfini dans OCaml et correspond exactement à ce que nous venons de faire (sauf que le champ mutable s'appelle contents)
   Par exemple let a = {contents = 5}; crée une variable ayant son champ mutable contents qui vaut 5.
- Afin d'alléger la syntaxe, on peut écrire directement let a = ref 5;;.
   On retiendra que l'effet reste le même : on a crée une variable a ayant un champ mutable entier qui contient 5.
- Si on veut accéder à la valeur du champ mutable, une syntaxe "adoucie" est aussi disponible avec !. On écrira par exemple print\_int !a pour afficher le contenu du champ mutable de a.
  - On retiendra que c'est identique à print\_int a.contents

- Le type ref est prédéfini dans OCaml et correspond exactement à ce que nous venons de faire (sauf que le champ mutable s'appelle contents)
   Par exemple let a = {contents = 5}; crée une variable ayant son champ mutable contents qui vaut 5.
- Afin d'alléger la syntaxe, on peut écrire directement let a = ref 5;;.
   On retiendra que l'effet reste le même : on a crée une variable a ayant un champ mutable entier qui contient 5.
- Si on veut accéder à la valeur du champ mutable, une syntaxe "adoucie" est aussi disponible avec !. On écrira par exemple print\_int !a pour afficher le contenu du champ mutable de a.
  - On retiendra que c'est identique à print\_int a.contents
- Pour modifier la valeur de a, une syntaxe plus simple est aussi disponible avec :=. On écrira par exemple a := 5

- Le type ref est prédéfini dans OCaml et correspond exactement à ce que nous venons de faire (sauf que le champ mutable s'appelle contents)
   Par exemple let a = {contents = 5}; crée une variable ayant son champ mutable contents qui vaut 5.
- Afin d'alléger la syntaxe, on peut écrire directement let a = ref 5;;.
   On retiendra que l'effet reste le même : on a crée une variable a ayant un champ mutable entier qui contient 5.
- Si on veut accéder à la valeur du champ mutable, une syntaxe "adoucie" est aussi disponible avec !. On écrira par exemple print\_int !a pour afficher le contenu du champ mutable de a.
  - On retiendra que c'est identique à print\_int a.contents
- Pour modifier la valeur de a, une syntaxe plus simple est aussi disponible avec :=. On écrira par exemple a := 5
   Par exemple, a:= !a +1 permet d'incrémenter de 1 la valeur (du champ mutable) de a.



## Exemple

Créer une reférence a vers 42 et une référence b vers 2023. Echanger le contenu de ces deux variables en utilisant une troisième référence temp

## Exemple

Créer une reférence a vers 42 et une référence b vers 2023. Echanger le contenu de ces deux variables en utilisant une troisième référence temp

```
let a = ref 42;;
let b = ref 2023;;
let temp = ref !a;;
a := !b;
b := !temp;
```



## C9 OCaml: aspects fonctionnels

2. Boucles

## Boucle for

Il y a deux versions :

• Indice croissant :

```
for indice = debut to fin do
    expression
done
```

## Boucle for

#### Il y a deux versions:

• Indice croissant :

```
for indice = debut to fin do
expression
done
```

Indice décroissant :

```
for indice = debut downto fin do
expression
done
```

## C9 OCaml: aspects fonctionnels

2. Boucles

## Boucle for

Il y a deux versions :

```
• Indice croissant :
```

```
for indice = debut to fin do
    expression
done
```

#### • Indice décroissant :

```
for indice = debut downto fin do
    expression
done
```

#### Attention :

## Boucle for

Il y a deux versions :

Indice croissant :

```
for indice = debut to fin do expression
done
```

• Indice décroissant :

```
for indice = debut downto fin do
expression
done
```

#### Attention :

• expression est de type unit (ex : affectation, affichage, ...)

## Boucle for

Il y a deux versions :

```
• Indice croissant :
```

```
for indice = debut to fin do expression done
```

• Indice décroissant :

```
for indice = debut downto fin do
expression
done
```

#### Attention :

- expression est de type unit (ex : affectation, affichage, ...)
- Les deux bornes de la boucle sont incluses

### Boucle for

Il y a deux versions :

```
Indice croissant :
```

```
for indice = debut to fin do
expression
done
```

Indice décroissant :

```
for indice = debut downto fin do
expression
done
```

#### Attention :

- expression est de type unit (ex : affectation, affichage, ...)
- Les deux bornes de la boucle sont incluses
- Pas de break ou de continue et indice ne peut pas être modifié dans la boucle



## Exemple de boucle for

• Ecrire une fonction harmonique\_asc qui prend en argument un entier n et calcule la somme des inverses des entiers non nuls jusqu'à n de façon ascendante.



## Exemple de boucle for

- Ecrire une fonction harmonique\_asc qui prend en argument un entier n et calcule la somme des inverses des entiers non nuls jusqu'à n de façon ascendante.
- Même question de façon descendante avec la fonction harmonique\_desc



## Exemple de boucle for

- Ecrire une fonction harmonique\_asc qui prend en argument un entier n et calcule la somme des inverses des entiers non nuls jusqu'à n de façon ascendante.
- Même question de façon descendante avec la fonction harmonique\_desc
- Que pensez de l'écart observé entre les deux valeurs pour n assez grand?



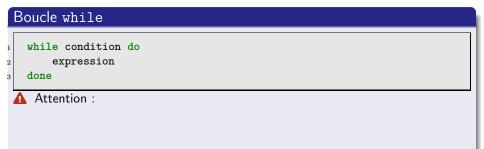
## Boucle while

while condition do expression done



## Boucle while

while condition do expression done



## Boucle while

while condition do expression done

done

#### Attention :

• expression est de type unit (ex : affectation, affichage, ...)

## Boucle while

while condition do expression

done

#### Attention :

- expression est de type unit (ex : affectation, affichage, ...)
- condition est de type bool

## Boucle while

while condition do expression

done

#### Attention :

- expression est de type unit (ex : affectation, affichage, ...)
- condition est de type bool
- Pas de break



## Exemple

• Ecrire une fonction permettant de calculer la somme des n premiers entiers à l'aide d'une boucle while.



## Exemple

- Ecrire une fonction permettant de calculer la somme des n premiers entiers à l'aide d'une boucle while.
- $\bullet$  Ecrire une fonction log2 qui prend en argument un entier n et renvoie l'entier k tel que  $2^{k-1} < n \le 2^k$



Les tableaux en OCaml, sont identiques à ceux vus en C. C'est à dire qu'il s'agit d'une structure mutable dont les éléments sont rangés de façon contigue en mémoire (et donc avec un accès en O(1)).

• Les tableaux sont notés entre [| et |]



```
• Les tableaux sont notés entre [| et |]
let ex = [|2; 7; 9; 14 |]
```

- Les tableaux sont notés entre [| et |] let ex = [|2; 7; 9; 14 |]
- L'accès à un élément se fait avec la notation tab. (i)



- Les tableaux sont notés entre [| et |] let ex = [|2; 7; 9; 14 |]
- L'accès à un élément se fait avec la notation tab.(i) print int ex.(1) affiche 7

- Les tableaux sont notés entre [| et |] let ex = [|2; 7; 9; 14 |]
- L'accès à un élément se fait avec la notation tab.(i) print int ex.(1) affiche 7
- La structure est mutable, attention l'affectation se fait avec <- (sans let)

- Les tableaux sont notés entre [| et |] let ex = [|2; 7; 9; 14 |]
- L'accès à un élément se fait avec la notation tab.(i) print\_int ex.(1) affiche 7
- La structure est mutable, attention l'affectation se fait avec <- (sans let) ex.(1)<-13 le continu du tableau est maintenant [|2; 13; 9; 14 |]



• Array.make qui prend en argument un entier et une valeur et crée le tableau contenant n fois cette valeur

- Array.make qui prend en argument un entier et une valeur et crée le tableau contenant n fois cette valeur
- Array.length donne le nombre d'élément du tableau (en O(1))

- Array.make qui prend en argument un entier et une valeur et crée le tableau contenant n fois cette valeur
- Array.length donne le nombre d'élément du tableau (en O(1))
- Array.of\_list Array.to\_list permet de convertir depuis ou vers une liste.

- Array.make qui prend en argument un entier et une valeur et crée le tableau contenant n fois cette valeur
- Array.length donne le nombre d'élément du tableau (en O(1))
- Array.of\_list Array.to\_list permet de convertir depuis ou vers une liste.
- Array.map, Array.iter, Array.fold\_left (ou Array.fold\_right) sont les équivalents sur les tableaux des fonctions de même non sur les listes.



## Exemples

• Crée le tableau contenant les doubles des 100 premiers entiers



## Exemples

- Crée le tableau contenant les doubles des 100 premiers entiers
- Ecrire la fonction affiche permettant d'afficher les éléments d'un tableau d'entier



## **Exemples**

- Crée le tableau contenant les doubles des 100 premiers entiers
- Ecrire la fonction affiche permettant d'afficher les éléments d'un tableau d'entier
- Rappeler l'algorithme du crible d'Erastothène et l'écrire en OCaml.