Licence STS, semestre 4 — 2009–2010
Langages et Génie Logiciel (Info 223) — 22 janvier 2010
`http://www.lri.fr/~paulin/LGL`

Cours 1 : Programmation impérative, organisation de la mémoire, références

1 Programmation impérative en Ocaml

Contrairement à une *expression*, une *instruction* dans un langage de programmation ne calcule pas de *valeur* mais modifie l'état de la machine (entrée, sortie, modification de l'état mémoire...), on dit qu'elle fait un *effet de bord*.

En CAML, une instruction est vue comme une expression ordinaire de type `unit`. Le type `unit` ne contient qu'un seul élément noté `()`.

Le type `unit` sera utilisé en *résultat* d'une fonction qui ne calcule pas de valeur mais fait un effet de bord. Il est aussi utilisé en *argument* de fonctions réalisant des effets de bord. On distingue :

```
let input1 = let n = read_int () in print_int n
let input2 () = let n = read_int () in print_int n
```

`input1` est une expression de type `unit` qui lit une entrée et l'imprime immédiatement *une seule fois* au moment de la définition ; la valeur de `input1` est égale à la constante `()`.
`input2` est une fonction de type `unit -> unit` qui à chaque application (`input2 ()`) lira une entrée et l'imprimera.

CAML propose des constructions impératives pour des conditionnelles, des boucles ou des séquences qui sont du *sucre syntaxique* pour des constructions fonctionnelles :

$$\frac{b : \texttt{bool} \quad e : \texttt{unit}}{\textbf{if } b \textbf{ then } e : \texttt{unit}} \qquad \frac{e_1\ e_2 : \texttt{unit}}{e_1; e_2 : \texttt{unit}} \qquad \frac{e_i : \texttt{unit} \quad i = 1 \ldots n}{\textbf{begin } e_1; \ldots; e_n \textbf{ end} : \texttt{unit}}$$

$$\frac{n_1\ n_2 : \texttt{int} \quad i : \texttt{int} \vdash e : \texttt{unit}}{\textbf{for } i = n_1 \textbf{ to } n_2 \textbf{ do } e \textbf{ done} : \texttt{unit}} \qquad \frac{n_1\ n_2 : \texttt{int} \quad i : \texttt{int} \vdash e : \texttt{unit}}{\textbf{for } i = n_1 \textbf{ downto } n_2 \textbf{ do } e \textbf{ done} : \texttt{unit}}$$

$$\frac{b : \texttt{bool} \quad e : \texttt{unit}}{\textbf{while } b \textbf{ do } e \textbf{ done} : \texttt{unit}}$$


Une boucle **for** est aussi efficace qu'un programme *récursif terminal*.

```
let funfor n1 n2 f =
    let rec fori i = if i <= n2 then begin f i; fori (i+1) end
in fori n1
```

En CAML, on utilise souvent des itérateurs comme `List.iter` qui applique un même traitement à tous les éléments d'une liste.

```
#List.iter
- : ('a -> unit) -> 'a list -> unit = <fun>
```

2 Organisation de la mémoire

Les données du programme sont organisées dans des *cellules* mémoire de taille limitée (en général 32 ou 64 bits) aussi appelés *mots*. On distingue les *données simples* (entiers machines, flottants) qui seront stockées sur une seule cellule et les *données composées* (tableaux, listes) qui nécessitent plusieurs cellules.

En CAML, une donnée composée est stockée en mémoire dans un ou plusieurs *blocs*. Un bloc est une suite de cellules permettant de stocker les composants de la donnée précédés d'une *entête* qui stocke des informations utiles sur le bloc comme sa taille.

En CAML, on ne manipule pas le bloc lui-même mais son *adresse*. Ainsi toute donnée est toujours identifiée par une information (valeur ou adresse) qui tient sur une seule cellule mémoire. Cette approche permet également de *partager* des cellules mémoire entre plusieurs objets.

 L'opérateur d'égalité = ne permet pas de savoir si deux cellules mémoires sont différentes, il ne permet que de comparer leur contenu. Il faut utiliser l'opérateur == pour comparer l'adresse de deux cellules.

```
# [5] = [5];;
- : bool = true
# [5] == [5];;
- : bool = false
```

3 Références

Une variable CAML est un nom associé à une cellule mémoire qui contient la valeur de l'expression associée à la variable.

CAML *ne permet pas* de modifier en place la valeur d'une variable.

Une *référence* est un pointeur sur une cellule mémoire. On crée une référence vers une cellule mémoire contenant l'entier 10 de la manière suivante :

```
let x = ref 10
```

`x` est une variable ordinaire de CAML dont la valeur (non modifiable) est l'adresse d'une cellule.

On peut modifier la valeur contenue dans la cellule avec l'opérateur `:=`, appelé opérateur d'*affectation*, et la lire à l'aide de l'opérateur `!`, appelé opérateur de *déréférencement*.

```
# x:=11;;
- : unit = ()
# !x;;
- : int = 11
```

 Une référence introduit une indirection (ce n'est pas la variable `x` qui est modifiée mais la cellule sur laquelle elle pointe) et peut donc se révéler moins efficace qu'une variable ordinaire lorsque le compilateur de fait pas d'optimisation.