Programmation Fonctionnelle
Cours 2
String, Char, Instructions, Functions Récursives

September 25, 2013

Le type char

- C'est le type des caractères ASCII
  American Standard Code pour Information Interchange
  = essentiellement les caractères qu'on trouve sur un clavier américain, plus des caractères de contrôle
- Les constantes du type `ch r` s'écrivent entre apostrophes '
- En fait il y a 256 valeurs de ce type, mais l'interpréteur OCaml ne sait pas afficher les caractères au-delà du numéro 127 (mais ça marche avec des programmes OCaml compilés).
- Il y a une bibliothèque séparée qui fournit un type de caractères UTF8.
- Le type `ch r` est disjoint du type `int`, mais il y a des fonctions de conversion.

Exemples (examples2.ml)

```
(* caractere ASCII *)
let x = 'a';;

int_of_char 'x';;

char_of_int 78;;

(* pas le bon apostrophe *)
int_of_char `b`;;

(* erreur de conversion *)
char_of_int 56734;;
```


Le type string

- Les chaînes de caractères.
- Les constantes du type `string` s'écrivent entre guillemets ".
- Les caractères non ASCII (au-delà de 127 sont représentés par leur code, mais OCaml ne peut pas les traiter nativement).
- Avec des programmes OCaml compilés on peut afficher des chaînes Unicode sans problème (mais les positions dans les chaînes ne seront pas calculées correctement !)
- Les types `ch r` et `string` sont disjoints.

Quelques fonctions utiles

- `String.length x` : longueur de la chaîne x
- `x^y` : concaténation des chaînes x et y
- `String.make c n` : chaîne qui consiste en n copies du caractère c.

Exemples (examples3.ml)

```
let x = "hello";;

String.length x;;

"abc" ^ "def";;

String.make 10 'a';;

let string_of_char c = String.make 1 c;;

string_of_char 'x';;
```


Positions dans les chaînes de caractères

- Le premier caractère dans une chaîne est à la position 0
- Le dernier caractère dans une chaîne x est à la position `(String.length x) - 1`
- `String.get x n` : caractère à la n-ème position de x
- `String.sub x n l` : sous-chaîne de x de longueur l, à partir de la position n de x.
- Il y a plus de fonctions, voir la description du module `String`.

Exemples (examples31.ml)

```
String.sub "abcdefg" 2 3;;

String.get x 2;;

String.get x 17;;

String.get x 0;;

let last x = String.get x ((String.length x) - 1);;

last x;;
```

Représentation de caractères particuliers

Caml	Caractère représenté
\\	antislash (\)
\n	saut de ligne (line feed)
\r	retour chariot (carriage return)
\t	tabulation
ddd	le caractère avec le code ASCII *ddd* en décimal
\'	apostrophe ('), seulement dans les constantes de caractères
\"	guillemet ("), seulement dans les constantes de chaînes

Le caractère \ est un « caractère d'échappement »
(angl. : escape character).

Voir le manual (module `Ch r`) pour une liste complète.

Exemples (examples4.ml)

```
(* Fonctions de conversion *)

string_of_int 17;;

string_of_float 3.0;;

int_of_string "42";;

float_of_string "4x5";;
```


Exemple: caractères non-ASCII

```
let s = "(e)";;

String.length s;;

let c = String.get s 1;;
```


`string` n'est pas un type basique

- L'opérateur de test = teste l'égalité structurelle (est-ce la même valeur ?)
- Il y a un deuxième opérateur == qui se comporte pareil sur les types basiques.
- Sur les types non-basiques, == fait une comparison physique (égalité de l'adresse mémoire)
- Normalement c'est l'opérateur = que vous voulez utiliser, pas l'opérateur == !

Exemples (equality.ml)

```
1 = 1;;
1 == 1;;
"ab" = "ab";;
"ab" == "ab";;

let s = "ab";;
let t = s in s==t;;
```


Le type `unit`, et les « instructions » d'affichage

- Le type `unit` contient un seul élément, noté `()` et prononcé également *unit*.
- Utile par exemple pour des fonctions qui ne n'envoient pas de résultat intéressant.
- Ces fonctions ont normalement un « effet de bord », en particulier des instructions d'entrée et de sortie.

Instructions d'affichage

- Instructions de sortie vers `stdout` (par défaut, l'écran)
- Instructions typées :
  `print_string: string -> unit` etc.
- Terminer la ligne courante et forcer l'affichage :
  `print_newline: unit -> unit`
- Pourquoi un argument du type `unit` ?
- Il y a aussi des fonctions du type `printf`.

Enchaînement

- Si $e_1, \ldots, e_n$ sont des expressions alors $e_1; \ldots; e_n$ est aussi une expression.
- Toutes les expressions sont évaluées dans l'ordre
- Le résultat est le résultat de l'évaluation de e_n
- Son type est le type de e_n
- Seulement utile pour des expressions ayant un effet de bord

Exemples (sequence.ml)

```
3;4;;

let f a =
  if a>10
  then begin
    print_int a;
    print_endline "␣est␣strictement␣superieur␣a␣10."
  end
  else (  (* On peut utiliser des parentheses au lieu d
    print_int a;
    print_endline "␣est␣inferieur␣ou␣egale␣a␣10."
  );;


f 2;;
```


- Plusieurs définitions simultanées sont séparées par le mot clef `and`.
- En particulier utile pour les définitions de fonctions récursives (voir plus tard)

Exemples (examples5.ml)

```
let a = 1
in let b = a+2
in b;;

let a = 3 and b = 4;;

a+b;;

let c = 2 and d = c+1;;

let c = 2;;
let d = c;;

let x = 2 in
let x = 17 and y = x*3
```


`let rec`

- Si on utilise f dans la définition d'une fonction `let f x = ...`, alors ce f fait référence à la valeur de f avant la définition.
- Le mot clef `rec` permet d'écrire une définition récursive d'une fonction.

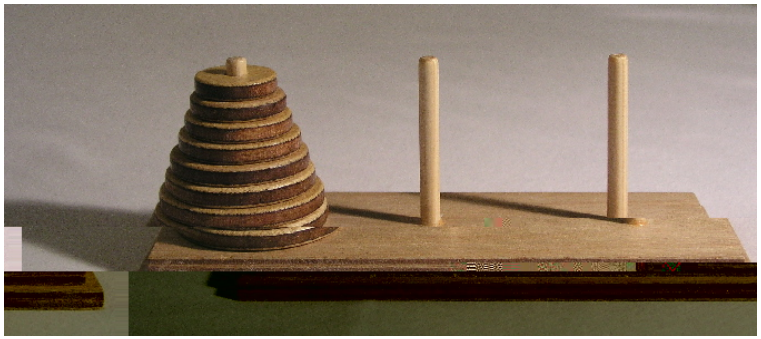
Exemples (examples7.ml)

```
let x = 1
let x = x + 3;;

(* pareil avec les fonctions *)
let f x = x+1;;
let f x = f (f x);;
f 2;;

let fact n = if n <=1 then 1 else n*fact(n-1);;

let rec fact n = if n <=1 then 1 else n*fact(n-1);;

fact 1;;
fact 10;;
```


Exemples (examples8.ml)

```
(* fonctions simultanement recursives *)

let rec pair x = if x=0 then true else impair (x-1)
and impair x = if x=0 then false else pair (x-1);;

pair 42;;
pair 17;;

#trace pair;;
#trace impair;;

pair 10;;
```


Les Tours de Hanoï



- déplacer la tour d'un poteau vers un autre
- déplacer un disque à la fois
- jaimais placer un disque au-dessus d'un disque plus petit


```
let bouge_disque origine destination =
  (* deplace un seul disque de origine vers destination *)
  print_string "Deplacer un disque du poteau ";
  print_int origine;
  print_string " vers le poteau ";
  print_int destination;
  print_endline ".";;
let rec hanoi initial terminal auxiliaire disques =
  (* Tours Hanoi de initial vers terminal en utilisant au
  if disques = 1
  then bouge_disque initial terminal
  else begin
    hanoi initial auxiliaire terminal (disques-1);
    bouge_disque initial terminal;
    hanoi auxiliaire terminal initial (disques-1)
  end;;
hanoi 1 2 3 3;;
```