

Deug MIAS 1^{ère} année : UE SM23
SM 2^{ème} année : module H7
STPI 2^{ème} année
IUP Miage 1^{ère} année

Examen de TP Informatique – 1^{ère} session

2 mai 2001

Durée : 2h00.

Barème indicatif : 1 : 6 points ; 2 : 7 points ; 3 : 7 points. Les problèmes sont indépendants.

Sans documents.

Respecter le lexique des énoncés. Ne pas les recopier.

1 Lecture d'expressions Caml

Donner la réponse de l'interpréteur Caml lors de l'évaluation de chacune des expressions suivantes. Quand l'expression est correcte, donner son type. Quand l'expression est incorrecte, préciser l'origine de l'erreur.

1. `"h";;`
2. `12+4.5;;`
3. `23, 't';;`
4. `17 :: [34;76];;`
5. `let x=(4,3) in [x,x];;`
6. `string_of_char('g');;`
7. `let y = 5 in
 let x = if 4<y then 8 else 10 in
 x+3;;`
8. `int_of_string("7875");;`
9. `let x = 1 in
 3 + let y = if x mod 2 = 1 then x else 3*x in y*2;;`
10. `[12, [1+(if 4>5 then 3 else 5), [2;3]], "false"];;`

2 Représentation d'un nombre décimal dans une base

On veut implanter une fonction calculant la représentation d'un nombre décimal n dans une base b quelconque. On rappelle qu'en base b , les nombres sont composés de chiffres dont la valeur varie entre 0 et $b - 1$. Le résultat sera une séquence d'entiers compris entre 0 et $b - 1$ qui représentera le nombre n .

Exemples :

- $n = 2768$ s'écrit 2768 en base $b = 10$: $base(2768, 10) = [2; 7; 6; 8]$
- $n = 3$ s'écrit 11 en base $b = 2$: $base(3, 2) = [1; 1]$

Une spécification de cette fonction est :

$base$: un entier ≥ 0 , un entier $> 0 \rightarrow$ une séquence d'entiers.

$\{base(n, b)$ est la séquence d'entiers compris entre 0 et $b - 1$ représentant l'entier n dans la base $b\}$

Une réalisation de cette fonction est :

- si $n < b$, il n'y a rien à faire, la représentation de n dans b est n : $base(n, b) = [n]$ (1)
- si $n \geq b$, on ajoute à droite de la représentation du quotient de la division entière de n par b le reste (*mod*) de cette division : $base(n, b) = ajd(base(n/b, b), n \bmod b)$ (2)

2.1 Réalisation de $base$ en Caml

Donner une réalisation en Caml de la fonction $base$ à partir des deux équations ci-dessus et de la fonction ajd dont la réalisation fera l'objet de la question 2.2. Que vaut $base(10, 2)$?

2.2 Réalisation de ajd en Caml

La fonction ajd ajoute à droite d'une séquence d'éléments de type quelconque (ici, ce ne sont plus nécessairement des entiers) un élément. Compléter sa réalisation en Caml ci-dessous :

```
let ..... = fonction
  [] , e -> [e]
  | e'::s, e -> ...
;;
```

2.3 Test de ajd

Pour tester la fonction ajd sur les entiers, on fait évaluer à l'interpréteur Caml l'expression « $ajd([3, 2], 1);;$ ». La réponse est :

```
>ajd([3, 2], 1) ;;
>
Cette expression est de type int, mais est utilisée avec le
type int*int.
```

Expliquer la réponse de l'interpréteur. Corriger l'expression « `ajd([3, 2], 1);;` ». Compléter pour avoir un jeu de tests significatifs de la fonction *ajd* sur les entiers.

2.4 Complexité de la fonction *base*

Pour évaluer la complexité de nos réalisations, on fait appel à la possibilité de tracer leur exécution :

```
La fonction base est dorénavant tracée.
La fonction ajd est dorénavant tracée.
#base(2456, 10);;
base <-- 2456, 10
base <-- 245, 10
base <-- 24, 10
base <-- 2, 10
base --> [2]
ajd <-- [2], 4
ajd <-- [], 4
ajd --> [4]
ajd --> [2; 4]
base --> [2; 4]
ajd <-- [2; 4], 5
ajd <-- [4], 5
ajd <-- [], 5
ajd --> [5]
ajd --> [4; 5]
ajd --> [2; 4; 5]
base --> [2; 4; 5]
ajd <-- [2; 4; 5], 6
ajd <-- [4; 5], 6
ajd <-- [5], 6
ajd <-- [], 6
ajd --> [6]
ajd --> [5; 6]
ajd --> [4; 5; 6]
ajd --> [2; 4; 5; 6]
base --> [2; 4; 5; 6]
- : int list = [2; 4; 5; 6]
```

2.4.1

Quel est la commande à utiliser pour obtenir une telle trace? Combien y-a-t'il d'appels récursifs à *base*? Pour chacun de ces appels, combien y-a-t'il d'appels récursifs à *ajd*? Qu'en déduire?

2.4.2

On décide de se passer de la fonction *ajd*, et donc de construire – dans un premier temps – la représentation de *n* dans la base *b* à l'**envers** en faisant des ajouts

à gauche (fonction `base_envers`). Dans un deuxième temps, on retournera la liste obtenue. Donner la réalisation en Caml de la fonction `base_envers`.

2.4.3

Une spécification de la fonction `retourne` retournant une liste est :

retourne : une séquence d'éléments \rightarrow une séquence d'éléments
 {*retourne*(*l*) est la liste obtenue en écrivant *l* de droite à gauche}

Par exemple : `retourne([1; 2; 3]) = [3; 2; 1]`

Déduire de ce qui précède la réalisation en Caml de la fonction `base`.

2.4.4

Une réalisation naïve de `retourne` fait appel à l'ajout à droite, ce qu'on essaye justement d'éviter. A cette fin, on utilise `acc`, une liste *accumulateur*, de la façon suivante :

<i>l</i>	<i>acc</i>
[1; 2; 3]	[]
[2; 3]	[1]
[3]	[2; 1]
[]	[3; 2; 1]

On introduit donc une fonction auxiliaire `ret_aux` :

ret_aux : deux séquences d'éléments \rightarrow une séquence d'éléments
 {*ret_aux*(*l*, *acc*) est la liste obtenue en accumulant dans *acc* les éléments successifs de *l*, ce qui a pour effet de retourner *l*}

Ainsi, la réalisation en Caml de la fonction `retourne` sera :

```
let (retourne : int list -> int list) = function
  l -> ret_aux(l, [])
;;
```

Compléter la réalisation suivante de `ret_aux` :

```
let (ret_aux : ..... ) = function
  [e], acc -> e::acc
  | ..., acc -> ...
;;
```

3 Banque d'investissement

On modélise une banque d'investissement de la manière suivante : la banque possède différents types de comptes : compte chèque, compte rémunéré, plan épargne logement, ... Chaque type de compte a un intitulé et un taux annuel qui est un entier représentant un pourcentage (par exemple 5 représente un taux de 5%). Un exemple de séquence de types de comptes est :

```
[("PEL", 5); ("Codevi", 3); ("Compte Chèques", 0);
 ("Livret A", 3)]
```

D'autre part la banque possède une séquence de comptes clients. Pour chaque compte on a le nom de son propriétaire, son solde, le nombre de jours écoulés depuis le dernier calcul d'intérêts et l'intitulé du type de compte. Un exemple de séquence de comptes clients est le suivant:

```
[("Durand", "Codevi", 12000, 123); ("Dupont", "PEL", 35000, 57);
 ("Dugenou", "Compte Chèques", 847, 12)]
```

On définit les types Caml suivant:

```
type nomClient == string;;
type duree == int;;
type solde == int;;
type intitulecompte == string;;
type taux == int;;
```

3.1 Comptes et comptes clients

Définir les types Caml des types de comptes (`typecomptes`) et comptes clients (`comptesclients`).

On définit en outre les types suivants :

```
type seqClient == comptesclients list;;
type seqTypeCompte == typecomptes list;;
```

3.2 Exemples

Donner l'expression Caml représentant les informations suivantes:

1. Un compte chèques au nom de Mr Ducoude dont les intérêts n'ont pas été réévalués depuis 77 jours, de solde 3243.
2. Un plan épargne populaire ("PEP") ayant un taux de 4%.

3.3 Montant d'un compte client

Soient `lclient` une séquence de clients et `ltypecompte` une séquence de types de comptes. On désire définir une fonction qui calcule le montant actuel d'un compte en incluant les intérêts qui n'ont pas encore été versés. Pour cela il faut retrouver le taux correspondant au type de compte. Précondition: il faut que l'intitulé du compte soit dans la séquence des comptes fournie.

Compléter les fonctions suivantes, et proposer des jeux de test significatifs :

```
let rec (tauxdutype :
      seqTypeCompte * intitulecompte -> taux) = fonction
  ((j,t)::l,i) -> ....
;;

let (credit_actuel :
      comptesclients * seqTypeCompte -> int) = fonction
  (... , ...) -> ....
;;
```

3.4 Montant des comptes d'un client

On désire pouvoir calculer la somme totale d'argent qu'un client possède dans cette banque, tous types de comptes confondus. Compléter la fonction suivante :

```
let rec (somme_totale :
    nomClient * seqTypeCompte * seqClient -> int) = function
    (n, ltype, [])          -> ...
  | (n, ltype, (nc,i,s,d)::l) -> ...
;;
```