Université Paris 6
Licence d'informatique

Module de Programmation
ANNÉE 1999/2000

Les seuls documents autorisés sont les documents manuscrits et les documents distribués en cours, TDs et TPs. La durée de l'épreuve est de 3h. Le barême est indicatif.

Exercice 1 Typage (7 points)

1. Typez soigneusement chacune de ces déclarations. La session débute dans l'environnement de typage global `E0`. Les environnements seront ensuite nommés `E1`, `E2`,...

```
#let g =
#   let y = 1 in
#   function x -> if x > y then x else -x;;
```

```
#let h f n x =
#   let rec g1 n1 res =
#     if n1 = n then res
#     else g1 (n1+1) (f res) in
#     g1 0 x;;
```

   Vous donnerez un exemple d'application totale de la fonction `h` précédente en justifiant son typage.

2. Typez soigneusement chacune de ces déclarations. La session débute dans l'environnement de typage global `E0`. Les environnements seront ensuite nommés `E1`, `E2`,...

```
#let d = function y -> function x -> y::x;;
```

```
#let f = function g -> function x -> function y ->
#   if (hd x) then tl (g x) else y;;
```

```
#let m = f (d true);;
```

Exercice 2 Évaluation (5 points)

On considère les déclarations suivantes dans un environnement initial E_0.

```
let x = 1;;

let op y = y + x;;

let rec foo = function f -> function n -> function acc ->
  if (n = 0)
  then acc
  else foo f (n-1) (f acc) ;;

let bar = function f ->
  let rec aux = function n -> function acc ->
    if (n = 0)
    then acc
    else aux (n-1) (f acc)
  in
    aux ;;
```

On notera E_1 l'environnement après la définition de `foo` et E_2 celui après le définition de `bar` ainsi que F et G les fermetures auxquelles ces noms sont liés.

2.1 Quels sont les environnements de définition des fermetures F et G ?

2.2 Détaillez l'évaluation de l'expression `(foo op 2 1)` dans E_2.

2.3 Détaillez l'évaluation de l'expression `(bar op 2 1)` dans E_2.

2.4 Quelle différence voyez vous dans ces deux évaluations ? Dans quelle mesure peut on dire que l'une est "meilleure" que l'autre ?

Exercice 3 Problème (8 points)
On désire implémenter une petite librairie permettant de manipuler beaucoup de nombres de façon compacte. Nous définirons un type `mucho` permettant de représenter soit un entier n soit tous les entiers entre deux entiers m et n. Un `mucho` de la forme (m, n) est dit **bien formé** si $m < n$ et nous ne considérerons que des `mucho` bien formés.

3.1 Définir le type `mucho`. Écrire une fonction `mucho_of_int` (: `int -> mucho`) qui transforme un entier n en un `mucho` qui le représente.

3.2 Écrire une fonction `mucho_of_int_int` (: `int -> int -> mucho`) qui pour deux entiers m et n créée un `mucho` bien formé représentant tous les entiers entre m et n. Écrire un prédicat `int_is_in_mucho` (: `int -> mucho -> bool`) qui teste si un entier n appartient à un `mucho` M.

Écrire une fonction `list_of_mucho` (: `mucho -> int list`) qui retourne la liste ordonnée des entiers que le `mucho` M représente.

3.3 Écrire une fonction `it_mucho` (: $\forall\alpha$ (α `->` `int` `->` α) `->` α `->` `mucho` `->` α) qui pour une fonction F, un élément x et un mucho M représentant les entiers $n_1 < \ldots n_k$ calcule $(F \ldots (F x\, n_1) \ldots n_k)$. Vous écrirez cette fonction de deux façons : soit en utilisant la fonction `list_of_mucho` et une fonctionnelle sur les listes, soit directement. Quelle version vous semble la meilleure ?

Deux `mucho` M_1 et M_2 seront dits **séparés** s'il n'ont aucun entier en commun. Un `mucho` M_1 est dit **précéder** un mucho M_2 si chaque entier de M_1 est plus petit que tous les entiers de M_2. Ainsi si deux `mucho` sont séparés l'un précède l'autre.

3.4 Écrire un prédicat `overlap` (: `mucho -> mucho -> bool`) qui teste si deux `mucho` M_1 et M_2 ont des entiers en commun. Définir une exception `Overlap`. Écrire le prédicat `precede` (: `mucho -> mucho -> bool`) qui pour deux `mucho` M_1 et M_2 retourne `true` si M_1 précède M_2, `false` si M_2 précède M_1. Cette fonction déclenchera l'exception `Overlap` si M_1 et M_2 ne sont pas séparés.

Nous représenterons un ensemble d'entiers par une liste de `mucho` séparés et triés dans l'ordre défini par `precede` et nous nommerons cet ensemble un **paquet** d'entiers.

3.5 Écrire une fonction `merge_mucho` (: `mucho -> mucho list -> mucho list`) qui pour un `mucho` M et un paquet L d'entiers retourne un paquet L' d'entiers contenant tous les entiers de M et tous les entiers de L.

3.6 En déduire une fonction `pack_int` (: `int list -> mucho list`) qui pour une liste quelconque L d'entiers retourne un paquet d'entiers. Déduisez également une fonction `merge_paquet` (: `mucho list -> mucho list -> mucho list`) qui combine deux paquets d'entiers en un seul paquet d'entiers