Université Montpellier 2, Sciences et Techniques , Département d'Informatique
Licence Informatique, semestre 3
Module: « Programmation applicative GLIN302 »

Enseignant: Stefano Cerri
Autre enseignants: Yannick Francillette, Emmanuel Hermellin, Fabien Priotto

Sujet d'examen écrit: le vendredi 17/01/2014 à 13h , Amphi Dumontet, Rang A

Etudiant: Prénom, nom, n. étudiant:
<n'oublier pas de remplir dans la copie anonyme, MAINTENANT svp>

Attention:

Les livres et les notes personnelles sont admises, ordinateur et portables non admises.

La totalité des exemples exploités dans ce sujet étaient disponibles dans le Labo Interactif duquel vous avez pris connaissance depuis le début.

Les questions posés ont pour but d'évaluer la compréhension des concepts, modèles et techniques *avant* de les exploiter en produisant du code.

Une seule question vous demande d'écrire du code (B22), les autres questions sont concernés avec votre compréhension du code d'autrui – activité majeure dans l'Informatique de demain - .

Les réponses les plus *succinctes* mais *complètes* seront mieux appréciés.

Le cas où il y ait (selon vous) une erreur : ne paniquez pas, ni forcez une réponse de la part des collègues en surveillance, au contraire: expliquez votre raison sur la copie.

Temps: 2 heures.

Résumé et barèmes des questions:

A. Fonctions de 1ère classe, portée des variables, fonctions et procédures (5 points)
B. Récursion terminale et enveloppée (5 points)
C: Flots, Dataflow, Concurrence declarative, Agents (client-serveur), Etat, Objets (5 points)
D: Sémantique et paradigmes (5 points)

**Bon courage, calme, concentration et bonne chance !!**

## A. Fonctions de 1ère classe, portée des variables, fonctions et procédures (5 points)

*% Une fonction peut prendre d'autres fonctions comme arguments et/ou renvoyer une fonction.*

```
declare
fun{Compose F G}
   fun{$ X}   {F {G X}}   end
end
fun{Sqr X}  X*X   end
fun{Double X}  X*2  end

MyF={Compose Sqr Double}
{Browse {MyF 4}}
```

On dit qu'un élément du langage est « de première classe » quand quatre conditions sont vérifiés: i) on peut donner un nom à l'élément, ii) on peut stocker l'élément dans une structure de données et le récupérer dynamiquement -pendant l'exécution-, iii) ??? , iv) ??? .
En Oz, il y a trois élément fondamentaux (type) du langage (noyeau) : a. les nombres, b. ??? , c. ???

Question A11: Quelles sont les conditions iii) et iv) ?
Question A12: Quels sont les deux autres éléments b. et c. ?
Question A13: En Oz les nombres sont de première classe. Est-ce que en Oz les fonctions sont de première classe? Quel autre élément fondamental est de première classe en Oz?
Question A14: Quelles fonctions sont passés en paramètre et quelle(s) sont restitués par valeur dans l'exemple de Compose (écrire les noms des deux types de fonctions) ?

*% Les identificateurs utilisés dans un procédure mais n'étant pas des paramètres (variables « liées ») sont des références externes (variables « libres »).*

```
local Y LB in
   Y=10
   proc {LB X ?Z}
      if X>=Y then Z=X else Z=Y end
   end
   local Y=15 Z in
      {LB 5 Z}
      {Browse Z}
   end
end
```

Question A21: Quelle est la valeur de Z affichée?
Question A22: Pourquoi?

*% Une fonction est simplement une procédure dont le dernier argument est une variable à laquelle le résultat est lié.*

```
declare
fun {MaxF X Y}
   if X>=Y then X else Y end
end
proc {MaxP X Y R}
```

```
   R = if X>=Y then X else Y end
end
{Browse {MaxP 2 3}}
local Z in
   {MaxF 2 3 Z}
   {Browse Z}
end
```

Question A31: Qu'est-ce qui est affiché dans les deux cas?

*% Les appels imbriqués sont définis à partir des appels simples en calculant les paramètres avant de faire l'appel extérieur. Les deux blocs suivants sont équivalents.*

```
declare
fun{F}  4  end
fun{G} 5  end
fun{S A B}   A+B   end

{Browse {S {F} {G}}}

local X Y Z in
   {F X}
   {G Y}
   {S X Y Z}
   {Browse Z}
end
```

Question A32: Est-ce que F et G sont des « vraies » fonctions d'un point de vue mathématique?
Question A33: Qu'est-ce qui est affiché dans les deux cas?

**B. Récursion terminale et enveloppée (5 points)**

*% On peut très simplement définir une fonction qui multiplie tous les éléments d'une liste.*

```
declare
fun{MultList L}
   case L
   of nil then 1 %neutre de la multiplication
   [] H|T then H*{MultList T}
   end
end

{Browse {MultList [1 2 3 4]}}
```

*% On peut définir une fonction qui multiplie tous les éléments d'une liste efficacement en utilisant un accumulateur.*

```
declare
fun{MultList2 L}
```

```
   fun{MultListAcc L Acc}
      case L
      of nil then ???1???
      [] H|T then {MultListAcc T ????2???}
      end
   end
in
   {MultListAcc L 1}
end

{Browse {MultList2 [1 2 3 4]}}
```

Question B1.1: Compléter le code de la fonction Multilist2 (écrire seulement ce qui doit remplacer ???1??? et ???2???)
Question B1.2: Quelle est la valeur affichée des deux Browse?
Question B1.3: Quel est l'intérêt de la transformation de Multilist en Multilist2 ?
Question B1.4: Est-ce que c'est nécessaire? Est-ce que c'est opportun? Dans quel cas nécessaire et/ou opportun?

*% De suite des définitions connues de Append et Reverse (en récursion enveloppée):*

```
declare
fun {Append Ls Ms}
   case Ls of nil then Ms
   [] X|Lr then X|{Append Lr Ms} end
end
{Browse {Append [1 2 3] [4 5 6]}}

declare
fun {Reverse Xs}
   case Xs of nil then nil
   [] X|Xr then {Append {Reverse Xr} [X]} end
end
{Browse {Reverse [a b c d]}}
```

Question B21: Quel sont les deux valeurs affichés par les deux Browse ?
Question B22: Ecrire l'algorithme Reverse qui exploite IterReverse , une fonction a récursion terminale avec accumulateur.

**C: Flots, Dataflow, Concurrence declarative, Agents (client-serveur), Etat, Objets (5 points)**

*% La correspondance de forme est définie à partir de la correspondance avec une seule forme simple. La création de paire de liste est effectuée avant le calcul de ses arguments.*

```
declare
fun {MapF Xs F}
   case Xs
   of nil then nil
   [] X|Xr then {F X}|{Map Xr F}
   end
```

```
end

proc {MapP Xs F ?Ys}
   case Xs
   of nil then Ys=nil
   else case Xs
          of X|Xr then
            local Y Yr in
               Ys=Y|Yr
               {F X Y}
               {MapP Xr F Yr}
            end
          end
   end
end
fun {Double X} 2*X end

{Browse {MapF [1 2 3] Double}}
{Browse {MapP [1 2 3] Double}}
```

Question C11: MapF et MapP sont elles équivalentes ?
Question C12: L'instruction Ys=Y|Yr dans MapP affecte Ys mais quelle est la valeur de Y à ce point ? Et quelle est la valeur de Yr ?
Question C13: Quels sont les valeurs affichés par les deux Browse?
Question C14: Qu'est-ce qu'est un flot?

*%Entre le producteur et le consommateur, d'autres fils peuvent exister qui transforment un flot (d'entrée) en un autre flot (de sortie). Ici, nous calculons la somme des entiers impairs entre 0 et 150.*

```
declare
fun {Generate N Limit}
   if N<Limit then N|{Generate N+1 Limit}
   else nil end
end
fun{Filter L F}
   case L
   of nil then nil
   [] H|T andthen {F H} then H|{Filter T F}
   [] H|T then {Filter T F}
   end
end
fun {Sum Xs A}
   case Xs of X|Xr then {Sum Xr A+X}
   [] nil then A end
end
fun {IsOdd X} X mod 2 \= 0 end

local Xs Ys S in
   thread Xs={Generate 0 150} end
   thread Ys={Filter Xs IsOdd} end
```

```
   thread S={Sum Ys 0} end
   {Browse S}
end
```

Question C21: La programmation Dataflow représente une vision complémentaire à la programmation objets. Quelles sont les fonctionnalités typiques de la programmation dataflow?
Question C22: Quel est le principe de base des variables dataflow qui rend la concurrence déclarative un modèle simple et puissant de système multi-agents (production, transformation et consommation de flots de données)?
Question C23: Si on ajoute l'instruction {Delay 1000} au corps de la fonction Generate, quelle est la conséquence?

*% L'état explicite et les fonctions*
*Cet exercice explore l'utilisation des cellules dans les fonctions. Nous définissons une fonction {Accumulate N} qui accumule toutes ses entrées, c'est-à-dire qu'elle additionne les arguments de tous ces appels. Voici un exemple :*

```
 {Browse {Accumulate 5}}
 {Browse {Accumulate 100}}
 {Browse {Accumulate 45}}
```

*Ces appels afficheront 5, 105 et 150, en supposant que l'accumulateur contient zéro au début. Voici une manière erronée d'écrire Accumulate :*

```
 declare
 fun {Accumulate N}
 Acc in
    Acc={NewCell 0}
    Acc:=@Acc+N
    @Acc
 end
```

Question C31: Quelle est l'erreur dans cette définition ? Comment la corrigeriez-vous ?

L'abstraction de données peut être non agrégée (constructeurs, selécteurs et prédicats sont séparée de la structure) ou agrégée (par exemple les objets); protégée ou non.
*% Pile sécurisée agrégée avec état et envoi procédural. La structure de donnée contenue dans une cellule est protégée et les opérations sont indissociables de la structure. On peut ici déjà parler d'objet.*

```
declare
fun {NewStack}
   C={NewCell nil}
   proc {Push E} C:=E|@C end
   fun {Pop} case @C of X|S1 then C:=S1 X end end
   fun {IsEmpty} @C==nil end
in
   proc {$ Msg}
      case Msg of push(X) then {Push X}
      [] pop( ?E) then E={Pop}
```

```
     [] isEmpty( ?B) then B={IsEmpty} end
   end
end

S1={NewStack}
{S1 push(a)}
{S1 push(b)}
{Browse {S1 pop($)}}
```

Question C41: Quel est le résultat du Browse?
Question C42: Quel est le type de S1 (type de base en Oz) ?
Question C43: Le corps de la procédure S1 contient des références, respectivement, à push et à Push; à pop et à Pop, à isEmpty et à IsEmpty. Quelle est la différence entre les premiers (push, pop, isEmpty) et les deuxièmes (Push, Pop, IsEmpty)?
Question C44: Où sont mémorisés les trois : Push, Pop, IsEmpty dans l'objet S1?
Question C45: Pourquoi on parle de pile sécurisée?

**D. Sémantique et paradigmes (5 points)**

A partir de la instruction sémantique suivante:

([({P Z}, {P → *p*, Y → *y*, Z → *z*})],
{ *p* = (proc {$ X} Y=X end, {Y → *y*}), *y*, *z*=1})

Question D1: Le processus continue à partir de cette instruction sémantique jusqu'à quand il n'y a plus aucune instruction sur la pile.
D11: Marquer la pile sémantique et la mémoire (les pas d'exécution) jusqu'à l'état final.
D12: Justifier.

Question D2: Dans le langage noyau il n'y a que des procédures.
D21: Pourquoi?
D22: Comment on représente les fonctions alors ?

Question D3: On a étudié 3 paradigmes de programmation:
le fonctionnel (ou applicatif),
le paradigme à concurrence déclarative et
les objets.
Pour obtenir la concurrence d'un coté, et les objets de l'autre on a rajouté exactement deux concepts dans le langage noyau.

D31: Quel concept doit-on ajouter au fonctionnel pour la concurrence déclarative?
D32: Quel concept pour les objets ?