Examen du 03/12/2013 (2h)

TOP : Techniques and tOols for Programming – 1A

Documents interdits, à l'exception d'une feuille A4 à rendre avec votre copie.

La notation tiendra compte de la présentation et de la clarté de la rédaction.

★ Questions de cours. (3pt)

▷ **Question 1:** ($\frac{1}{2}$pt) Définissez en français (sans équation) les notations O, Ω et Θ utilisées pour dénoter la complexité algorithmique en insistant sur leurs relations les unes avec les autres.

▷ **Question 2:** ($\frac{1}{2}$pt) Quel est le rapport entre les notations que vous venez de définir et les temps de calcul dans le meilleur des cas, le pire des cas et le cas moyen ?

▷ **Question 3:** (1pt) Définissez le principe des tris (1) par insertion (2) par sélection en explicitant leurs invariants (en français ou avec des écritures mathématiques que vous introduirez).

▷ **Question 4:** ($\frac{1}{2}$pt) Définissez le principe général de deux tris récursifs vu en cours.

▷ **Question 5:** ($\frac{1}{2}$pt) Pourquoi est-il parfois difficile de trouver l'invariant d'une boucle ? Quel est l'intérêt d'exprimer un tel invariant ?

★ Exercice 1: Des îles dans la mer de données (5pt – ACM Programming Contest 2013)

Étant donnée une séquence d'entiers $\{a1, a2, a3...an\}$, une île dans cette séquence est une sous-séquence continue dans laquelle tout élément est supérieur aux éléments immédiatement avant et après la sous-séquence. Dans les 3 exemples ci-dessous, chaque île est délimitée par des parenthèses.

- Exemple 1 : `0 0 1 1 2 2 1 1 0 1 2 2 1 1 0`. Il y a 4 îles : 1 1 2 2 1 1 ; 2 2 ; 1 2 2 1 1 ; 2 2.
  Cela donne `0 0(1 1(2 2)1 1)0(1(2 2)1 1)0`.
- Exemple 2 : `0 1 2 3 4 3 2 1 2 3 4 3 2 1 0`. Il y a 7 îles différentes.
  Cela donne `0(1(2(3(4)3)2)1(2(3(4)3)2)1)0`.
- Exemple 3 : `0 1 0 1 0 1 0 1 0 1 0 1 0 1 0`. Il y a également 7 îles, mais toutes de taille 1.
  Cela donne `0(1)0(1)0(1)0(1)0(1)0(1)0(1)0`

On suppose que tout entier de la liste est positif ou nul et aussi que toutes les îles sont valides, c'est à dire que chaque valeur ne diffère de celle qui la précède que de 1 au plus.

▷ **Question 1:** (2pt) On se donne dans un premier temps l'opération `head(List):Int` qui permet de retrouver la valeur du premier élément d'une liste (ou -1 si la liste est vide), ainsi que `tail(List):List` qui retourne la liste amputée de sa tête (ou la liste vide). La taille de la liste est inaccessible à priori.

Écrire une fonction récursive `islands(List):Int`, retournant le nombre d'îles dans la séquence passée en paramètre.

INDICATION : Le problème est grandement simplifié par le fait que toutes les îles sont supposées correctes.

▷ **Question 2:** (2pt) Dérécursivez l'algorithme développé dans la question précédente en utilisant les techniques vues en cours. Expliquez ce que vous faites, et argumentez pourquoi vous pouvez le faire.

▷ **Question 3:** (1pt). On suppose maintenant que les données sont rangées dans un tableau scala. On peut donc retrouver la valeur de la $i$ieme case du tableau `tab` avec `tab(i)` et la taille du tableau avec `tab.size()`. Proposez un algorithme itératif en $O(n)$ comptant les îles dans une séquence donnée.

★ Exercice 2: Un code mystère (4pt – d'après Maylis Delest).

On considère la fonction ci-contre, avec `swap(tab,a,b)`, qui inverse les valeurs des cases `tab(a)` et `tab(b)`, et le tableau A de dimension 8 contenant la séquence {2,8,7,4,3,6,5,1}.

```
def mystere(tab:Array[Int], cle:Int): Int {
  val d=0
  val f=tab.length-1
  do {
    while (d<tab.length && tab(d) <= cle) {
      d++;
    }
    while (f>=0 && tab(f)>cle) {
      f--;
    }
    if (d<f) {
      swap(tab,d,f);
      d++;
      f--;
    }
  } while (d<f);
  return f;
}
```

▷ **Question 1:** (2pt) Représentez graphiquement l'effet de l'appel `mystere(A, 6)` sur le tableau A en détaillant les différentes étapes et indiquez la valeur retournée par la fonction.

▷ **Question 2:** (1pt) Quelle est la valeur retournée par la fonction si tous les nombres contenus dans le tableau sont strictement inférieurs à la clé ? Même question si tous les nombres sont strictement supérieurs à la clé.

▷ **Question 3:** (1pt) Donnez la complexité maximale de cette fonction en nombre de tests, et justifiez votre réponse.

★ **Exercice 3: Calcul de la racine carrée (5pt** − d'après Hayssam Soueidan).

On souhaite montrer que le programme ci-contre calcule l'approximation entière de la racine carrée du paramètre x dans la variable y_1. Plus formellement, la post-condition souhaitée est la suivante : $Q \equiv \left((y_1)^2 \leq x\right) \wedge \left((y_1+1)^2 > x\right)$

```
1  def f(x:Int):Int= {
2    val y1=0
3    val y2=1
4    val y3=1
5
6    while (y3 <= x) {
7      y1 = y1 + 1
8      y2 = y2 + 2
9      y3 = y3 + y2
10   }
11   return y1
12 }
```

▷ **Question 1:** Explicitez l'invariant de la boucle présente dans ce programme. La difficulté est de lier la valeur des différentes variables de l'algorithme entre elle. Il est probablement utile de dérouler l'algorithme sur quelques cas et quelques étapes pour deviner ces liens entre les variables.

▷ **Question 2:** Calculez la plus faible pré-condition nécessaire pour que cette boucle mène au respect de la post-condition souhaitée. Explicitez vos étapes, et justifiez vos actions. La qualité de la rédaction est aussi importante que la justesse des écritures mathématiques.

▷ **Question 3:** Concluez en calculant la plus faible pré-condition nécessaire pour que l'algorithme entier réponde à la post-condition souhaitée.

Rappel : Règles de calcul des pré-conditions.

- $\mathbf{WP}(nop, Q) \equiv Q$
- $\mathbf{WP}(x := E, Q) \equiv Q[x := E]$
- $\mathbf{WP}(C; D, Q) \equiv \mathbf{WP}(C, \mathbf{WP}(\mathrm{D},\mathrm{Q}))$
- $\mathbf{WP}(\texttt{if}\ Cond\ \texttt{then}\ C\ \texttt{else}\ D, Q) \equiv (Cond = \texttt{true} \Rightarrow \mathbf{WP}(C, Q)) \wedge (Cond = \texttt{false} \Rightarrow \mathbf{WP}(D, Q))$
- $\mathbf{WP}(\texttt{while}\ E\ \texttt{do}\ C\ \texttt{done}\ \{\mathrm{inv\ I\ var\ V}\}, \mathrm{Q}) \equiv I$; Obligations de preuve :
  - $(E = \texttt{true} \wedge I \wedge V = z) \Rightarrow \mathbf{WP}(C, I \wedge V < z))$
  - $I \Rightarrow V \geq 0$
  - $(E = \texttt{false} \wedge I) \Rightarrow Q$

★ **Exercice 4: Fonctions récursives sur les listes d'entiers (3pt).**

On rappelle les opérateurs de base du type liste d'entiers :

$$\begin{cases} \texttt{Nil} \mapsto \text{La liste vide} \\ \textit{list}.\texttt{head} \mapsto \text{Premier caractère de la liste } \textit{list} & \text{(défini ssi } \textit{list} \text{ n'est pas vide)} \\ \textit{list}.\texttt{tail} \mapsto \textit{list} \text{ privée du premier élément} & \text{(défini ssi } \textit{list} \text{ n'est pas vide)} \\ \textit{entier} :: \textit{list} \mapsto \text{Concaténation de l'entier } \textit{entier} \text{ et de la liste } \textit{list} \end{cases}$$

Écrivez les fonctions suivantes en Scala en utilisant ces primitives. Si votre solution ne s'exécute pas en temps linéaire, **vous serez pénalisé**.

▷ **Question 1:** `longueur(li:List[Int]):Int` ⇝ renvoie le nombre d'éléments composant la liste.

▷ **Question 2:** `est_membre(v:Int, li:List[Int]):Bool` ⇝ indique si l'entier fait partie de la liste.

▷ **Question 3:** `somme(li:List[Int]):Int` ⇝ renvoie la somme de tous les éléments de la liste.

▷ **Question 4:** `croissante(li:List[Int]):Bool` ⇝ indique si la liste est triée en ordre croissant.

▷ **Question 5:** `retourne(li:List[Int]):List[Int]` ⇝ renvoie la liste lue en sens inverse.