**Université Paris 7 - Denis Diderot**
**IF1 - Partiel du 26 novembre 2005 - Durée : 2 heures 30 minutes**
**Documents, calculettes, ordinateurs, téléphones non autorisés**

---

**Remarques préliminaires**

- Il y a 3 exercices indépendants (un par feuille) que vous pouvez traiter dans l'ordre que vous voulez.
- Lorsqu'il est demandé d'écrire du code, le langage utilisé est Java. Le correcteur n'attachera pas une grande importance à l'absence de quelques points-virgules. Il n'en sera pas de même, par exemple, en ce qui concerne la rigueur apportée à la déclaration des variables, la syntaxe des structures de contrôle et l'utilisation des accolades { et }. La présentation (indentations) et les commentaires explicatifs seront particulièrement appréciés.

---

**Exercice 1.**

Cet exercice est une suite de questions indépendantes auxquelles il est demandé de répondre brièvement mais de façon justifiée.

**1.1.** Quelle est la représentation en base 8 du nombre s'écrivant `7B4E60A8` en base 16 ?

**1.2.** Soit le nombre $n$ égal à $35 + \frac{1}{2} + \frac{1}{8} + \frac{1}{32} + \frac{1}{64} + \frac{1}{1024}$

**1.2.1.** Le nombre $n$ est-il finiment représentable en base 2?
Si oui, quelle est cette représentation et, si non, pourquoi?

**1.2.2.** Même question avec le nombre $n + \frac{7}{10}$.

**1.3.** Éxécuter à la main la séquence Java suivante :

```
int m = 3, n = 2, p;
float f, g;
p = 2 * m + n;     m = n + 3;     n = 5;
Deug.println(p); Deug.println(m); Deug.println(n);
m = p / n;      f = p / n;
Deug.println(m); Deug.println(f);
g = n;    f = p / g;
Deug.println(f);
```

**1.4.** Déterminer, pour chacune des combinaisons de valeurs des variables $a$ et $b$, la valeur de la variable $d$ après exécution de la séquence suivante et en déduire une séquence plus simple équivalente :

```
boolean a, b, c, d;
  .....
if (!b)
    c = !a;
else
    c = true;
if (c)
   if (!a)
      d = !b;
   else
      d = true;
else
   d = false;
```

**Exercice 2.**

On dispose de $m$ pièces de 50 centimes, de $n$ pièces de 20 centimes et de $p$ pièces de 10 centimes (dans tout cet exercice, on supposera que $p$ est non nul).
On s'intéresse au paiement de sommes sans rendu de monnaie avec un tel assortiment de pièces.

**2.1.** Exprimer, sous forme d'une expression arithmétique Java, la somme maximale que l'on peut espérer payer avec cet assortiment de pièces.

**2.2.** Ne disposant pas de pièces de 1, 2 et 5 centimes, un certain nombre de sommes ne sont pas payables a priori (celles non multiples de 10), en plus de celles supérieures à la somme maximale précédente.
Exprimer au moyen d'une expression logique, combinant ces conditions, une condition *nécessaire* pour qu'une somme donnée soit *susceptible* d'être payée.

**2.3.** Exprimer, sous forme algorithmique simple, une stratégie de paiement d'une somme $s$ avec un nombre minimum de pièces pour un assortiment donné de pièces de 50, 20 et 10 centimes si le paiement est possible et qui détecte l'impossibilité de réaliser le paiement.

*Indication.* On prendra par ordre décroissant des valeurs des pièces, le maximum de pièces possibles. On admettra que cette stratégie donne une solution si le paiement est possible (cela en raison de l'hypothèse qu'on dispose d'au moins une pièce de 10 centimes).

**2.4.** Écrire un **programme Java complet** (c'est-à-dire dont la compilation conduit à du code exécutable par la commande `java`) :

- qui lit les nombres $m$, $n$ et $p$ et la somme $s$ à payer ;
- qui contient la définition d'une fonction (méthode `static`) `verif`
  - renvoyant la valeur booléenne `true` si les paramètres $m$, $n$, $p$ et $s$ sont des entiers positifs ou nuls (sauf $p$ qui doit être non nul) qui satisfont les conditions nécessaires pour que le paiement soit envisageable ;
  - renvoyant la valeur `false` sinon ;
- qui utilise la stratégie décrite en 2.3 pour chercher une solution (si la vérification précédente est correcte) ;
- qui affiche une solution s'il en existe et un message particulier s'il n'y en a pas.

**Exercice 3.**

À un tableau d'entiers donné (qu'on dira être sous *forme étendue*)

$$\underbrace{val_0 \cdots val_0}_{nb_0}\ \underbrace{val_1 \cdots val_1}_{nb_1} \cdots \underbrace{val_i \cdots val_i}_{nb_i} \cdots$$

on associe la suite de couples

$$(nb_0, val_0), (nb_1, val_1), ..., (nb_i, val_i), ...$$

dite *forme factorisée*.

Ainsi, si le tableau $t$ contient

| 3 | 4 | 4 | 1 | 1 | 1 | 3 | 3 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|

il lui correspond la suite de couples :

- $(1, 3)$ : le tableau $t$ commence par une séquence de longueur 1 de la valeur 3 ;
- $(2, 4)$ : $t$ contient ensuite une séquence de longueur 2 de la valeur 4 ;
- $(3, 1)$ : $t$ contient ensuite une séquence de longueur 3 de la valeur 1 ;
- $(2, 3)$ : $t$ contient ensuite une séquence de longueur 2 de la valeur 3 ;
- $(4, 1)$ : $t$ contient finalement une séquence de longueur 4 de la valeur 1.

Une représentation possible de cette suite de couples correspond au tableau suivant dans lequel les couples sont mis brutalement bout à bout :

| 1 | 3 | 2 | 4 | 3 | 1 | 2 | 3 | 4 | 1 |
|---|---|---|---|---|---|---|---|---|---|

Un tableau $t2$ représentant une forme factorisée vérifie les propriétés suivantes :

- sa taille est un nombre pair ;
- les éléments d'indice pair (c'est-à-dire $t[0]$, $t[2]$, ...) sont strictement positifs ;
- des éléments d'indices impairs consécutifs sont différents
  (c'est-à-dire $t[2 * i - 1] \neq t[2 * i + 1]$).

**3.1.** Écrire une fonction `altern` qui, étant donné un tableau $t$ d'entiers, renvoie le nombre de changements de valeurs observés lors du parcours du tableau de son premier à son dernier élément (pour notre exemple cette valeur est 4).

**3.2.** Écrire une fonction `taille` qui, étant donné un tableau $t$ d'entiers, vérifie si le tableau $t$ correspond à la représentation factorisée d'un tableau et renvoie alors la taille du tableau étendu qui lui correspond.
Si le tableau $t$ ne correspond pas à une forme factorisée, la fonction renverra $-1$.

**3.3.** Écrire une fonction `ecrireVal`, ne renvoyant pas de valeur et qui, étant donnés un tableau $t$ et des entiers *deb*, *val* et $n$, écrit consécutivement, dans le tableau $t$, $n$ fois la valeur *val* à partir de l'indice *deb*. Aucun contrôle ne sera fait dans la fonction : on supposera qu'elle est appelée avec des valeurs de *deb* et de $n$ compatibles avec des valeurs correctes d'indices dans le tableau $t$.

**3.4.** Écrire une fonction `factoriser` qui, étant donné un tableau d'entiers $t1$, renvoie le tableau d'entiers $t2$ correspondant à sa forme factorisée. On pourra utiliser la fonction `altern`, écrite en 3.1, pour déterminer la taille du tableau $t2$.

**3.5.** Écrire une fonction `etendre` qui, étant donné un tableau $t2$, renvoie le tableau $t1$ sous forme étendue qui lui correspond si $t2$ est une forme factorisée de tableau et renvoie la valeur `null` dans le cas contraire.