

Corrigé du Contrôle d'Informatique INF 311

Promotion 2004

Sujet proposé par François Morain

5 juillet 2005

Les exercices qui suivent sont indépendants et peuvent être traités dans n'importe quel ordre. On attachera une grande importance à la clarté, à la précision et à la concision de la rédaction.

Exercice 1

a) Expliquer en quelques mots ce que le programme suivant affiche à l'écran.

```
class Exo1{  
  
    static int f(int x, int y){  
        if((x == 0) || (y == 0))  
            return 0;  
        else  
            return x + f(x, y-1);  
    }  
  
    public static void main(String[] args){  
        TC.println(f(3, 5));  
    }  
}
```

Corrigé. La fonction f calcule $x \times y$, donc la fonction main affiche 15.

b) Donner toutes les raisons pour lesquelles le programme suivant est incorrect, qu'elles soient détectées par le compilateur, ou bien à l'exécution¹.

```
class Exo1b{  
  
    void g(int[] t){  
        for(int i = t.length-1; i >= 0; i--)  
            t[i] += t[i-1];  
    }  
  
    public static void main(String[] args){  
        int[] t = {1, 2, 3, 4};  
  
        g(t);  
    }  
}
```

¹Pour faciliter la description du programme, nous avons numéroté les lignes du programme, les numéros étant sur la droite de celui-ci.

```

        for(int j = 0; j < t.length; i++)           12
            TC.println(t[j]);                       13
    }                                               14
}                                                  15

```

Corrigé. La fonction `g` est définie comme une méthode d'objet, alors qu'elle est utilisée comme une méthode de classe. Dans cette même fonction `g`, le programme finira par atteindre l'indice `-1` à la ligne 5, ce qui provoquera une erreur à l'exécution. À la ligne 12, `i` n'est pas connu et le compilateur va se plaindre.

Exercice 2

Soit `t` un tableau de `n` entiers (`int`) *distincts*, qui est supposé trié dans l'ordre croissant, comme par exemple le tableau

```
int[] t = {-1, 1, 5, 6};
```

Le but de l'exercice est de déterminer s'il existe un entier `i` de l'intervalle $0..n-1$ tel que `t[i]` soit égal à `i`.

a) Écrire une fonction

```
static int indice(int[] t){...}
```

qui retourne un entier `i` satisfaisant `t[i] = i` s'il en existe un, et `-1` sinon. Votre fonction devra faire $O(n)$ comparaisons.

Corrigé.

```

static int indice(int[] t){
    for(int i = 0; i < t.length; i++)
        if(t[i] == i)
            return i;
    return -1;
}

```

b) Écrire une fonction

```
static int indiceRapide(int[] t){...}
```

qui retourne un tel indice, mais qui utilise seulement $O(\log n)$ comparaisons.

Corrigé. La clef de cette fonction se trouve dans la remarque suivante. Si $t[k] < k$ pour un certain indice `k`, alors aucun indice de l'intervalle $[0..k]$ n'est celui qu'on cherche. En effet, de $t[k-1] < t[k] < k$, on déduit qu'on ne peut avoir $t[k-1] = k-1$, etc.; on doit donc chercher dans $[k+1..n[$. De la même façon, si $k < t[k]$, alors $k < t[k] < t[k+1]$ et on ne peut avoir $t[k+1] = k+1$, et l'on doit chercher dans l'intervalle $[0..k-1]$. Maintenant, on peut combiner cette idée avec une approche diviser-pour-résoudre classique, ce qui conduit au programme :

```

// on cherche dans t[g..d]
static int indRap(int[] t, int g, int d){
    int m;

    if(g > d)
        return -1;
    m = (g+d)/2;
    if(t[m] == m)
        return m;
    else if(t[m] < m)
        return indRap(t, m+1, d);
    else // t[m] > m
        return indRap(t, g, m-1);
}

static int indiceRapide(int[] t){
    return indRap(t, 0, t.length-1);
}

```

Exercice 3

Le but de cet exercice est d'écrire un programme qui simule la gestion d'une blanchisserie commerciale.

Question 1. Définir une classe `Vetement` qui contienne les informations suivantes sur un vêtement : une `String` donnant le nom du vêtement (un pantalon, des chaussettes, un tee-shirt, etc.), un entier donnant son poids (en grammes) et un autre `int` donnant la température (en degrés Celsius) auquel le vêtement doit être lavé. Donner un constructeur explicite pour cette classe.

Corrigé.

```

class Vetement{
    String nom;
    int poids, temp;

    Vetement(String n, int p, int t){
        this.nom = n;
        this.poids = p;
        this.temp = t;
    }
}

```

Question 2. Les vêtements sont rangés sur des étagères. Chaque étagère est numérotée par un entier (un `int`) collé dessus. Ces étagères sont représentées par un tableau `t` de `NMAX` éléments de type `Vetement`, qui doit être une variable de classe d'une nouvelle classe `Blanchisserie`. Donner les descriptions correspondantes en Java et écrire une fonction

```
static void initialiser(){...}
```

qui initialise le tableau. Vous pouvez fixer la valeur de NMAX à 1000. Donner également la fonction main de la classe Blanchisserie.

Corrigé.

```
class Blanchisserie{

    final static int NMAX = 1000;

    static Vetement[] t;

    static void initialiser(){
        t = new Vetement[NMAX];
        for(int i = 0; i < NMAX; i++)
            t[i] = null;
    }

    public static void main(String[] args){
        initialiser();
    }
}
```

Question 3. Quand un client arrive, la procédure suivante est utilisée. On supposera pour simplifier que chaque client apporte un seul vêtement à chaque fois. Le blanchisseur cherche la première étagère vide d'indice minimal `ind`, il range le vêtement dans l'étagère correspondante `t[ind]` et il donne au client un ticket de numéro `ind`.

a) Écrire une fonction

```
static int trouverIndice(){...}
```

qui retourne l'indice recherché, ou `-1` si toutes les étagères sont utilisées.

Corrigé.

```
static int trouverIndice(){
    for(int i = 0; i < NMAX; i++)
        if(t[i] == null)
            return i;
    return -1;
}
```

b) Écrire une fonction

```
static int déposer(Vetement vet){...}
```

qui simule le rangement d'un vêtement sur une étagère, dont le numéro est déterminée par la fonction `trouverIndice()` précédente. Si une étagère libre est trouvée, son indice est retourné, sinon on retourne `-1`. Modifier la fonction principale `main` pour déposer un tee-shirt de 100g devant être nettoyé à 40°.

Corrigé.

```

static int deposer(Vetement vet){
    int ind = trouverIndice();

    if(ind == -1)
        return -1;
    else{
        t[ind] = vet;
        return ind;
    }
}
public static void main(String[] args){
    int ticket;
    initialiser();
    ticket = deposer(new Vetement("tee-shirt", 100, 40));
}

```

c) Écrire une fonction

```

static void retirer(int ticket){...}

```

qui libère l'étagère ticket quand un client vient récupérer son vêtement après nettoyage.

Corrigé.

```

static void retirer(int ticket){
    t[ticket] = null;
}

```

Question 4. Quand le blanchisseur veut nettoyer des vêtements, il sélectionne ceux qui doivent être nettoyés à la même température. Ses machines à laver ont bien sûr une capacité finie.

a) Associer au tableau `t` un autre tableau `s` qui décrit l'état d'un vêtement : il est soit `a_nettoyer`, `en_cours`, ou `fait`. Décrire une façon de représenter cette information, c'est-à-dire donner une description Java de l'état et du type du tableau `s`. Modifier les fonctions `initialiser()` et `deposer()` pour en tenir compte.

Corrigé.

```

final static int a_nettoyer = 0, en_cours = 1, fait = 2;
static int[] s;
static void initialiser(){
    t = new Vetement[NMAX];
    s = new int[NMAX];
    for(int i = 0; i < NMAX; i++){
        t[i] = null;
        s[i] = fait;
    }
}
static int deposer(Vetement vet){
    int ind = trouverIndice();
}

```

```

    if(ind == -1)
        return -1;
    else{
        t[ind] = vet;
        s[ind] = a_nettoyer;
        return ind;
    }
}

```

b) Comme on ne peut prévoir le nombre de vêtements à mettre dans une machine, on décide d'utiliser une liste de Vetement pour résoudre ce problème. Ce type devra stocker un vêtement et l'indice de l'étagère où il se trouve. Définir une classe ListeVetement qui implante cette idée en Java. Donner également un constructeur explicite pour la classe.

Corrigé.

```

class ListeVetement{
    Vetement vet;
    int ind;
    ListeVetement suivant;

    ListeVetement(Vetement v, int i, ListeVetement suiv){
        this.vet = v;
        this.ind = i;
        this.suivant = suiv;
    }
}

```

c) Écrire une fonction

```

static ListeVetement selectionner(int cap, int temp){...}

```

qui remplit une machine à laver de capacité cap avec des vêtements devant être lavés à la température temp. Le poids total des vêtements doit être \leq cap. L'algorithme de sélection fonctionne de la manière suivante : on parcourt le tableau des étagères et on prend le vêtement correspondant s'il tient dans la machine. Si son poids est trop important, on passe au suivant. La fonction doit mettre à jour tous les tableaux concernés et retourner la liste des vêtements trouvés.

Corrigé.

```

static ListeVetement selectionner(int cap, int temp){
    ListeVetement l = null;
    int reste = cap;

    for(int i = 0; i < NMAX; i++){
        if(s[i] == a_nettoyer){
            if((t[i].temp == temp) && (t[i].poids <= reste)){
                l = new ListeVetement(t[i], l);
                s[i] = en_cours;
            }
        }
    }
}

```

```

        reste -= t[i].poids;
    }
}
return l;
}

```

d) Écrire une fonction

```
static void retourner(ListeVetement l){...}
```

qui simule le retour des vêtements lavés, éléments de la liste l, dans leurs étagères respectives.

Corrigé.

```

static void retourner(ListeVetement l){
    while(l != null){
        s[l.ind] = fait;
        l = l.suivant;
    }
}

```