Université Paris-Diderot - L1 Novembre 2012

IF1: Introduction à l'Informatique et à la programmation - Devoir sur Table
(Correction)

Documents non autorisés. Le barème est donné seulement à titre indicatif.

Exercice 1 (10 points)

1. *Un entier* `i` *est* **diviseur propre** *de* `n` *si* `i` *divise* `n` *mais* `i` *est différent de* `n`. *Écrire une fonction Java* `diviseurpropre` *qui prend deux entiers positifs* `i` *et* `n` *en argument et qui renvoie un boolean indiquant si* `i` *est diviseur propre de* `n`.

   Correction :

```
static boolean diviseurpropre(int i,int n){
return (n%i==0);}
```

2. *Un entier positif* `k` *est un* **nombre parfait** *s'il est égal à la somme de tous ses diviseurs propres. Par exemple,* 6 *est parfait car* $6 = 1 + 2 + 3$. *Écrire une fonction Java* `parfait` *qui prend un entier positif* `k` *en argument et qui renvoie un boolean indiquant si* `k` *est un nombre parfait.*

   Correction :

```
static boolean parfait (int n){
int res=0;
    for (int i=1;i<n;i++)
        if (diviseurpropre(i,n)) res=res+i;
    return (n==res);
}
```

3. *Écrire un programme Java* `GenParfait` *qui lit un entier positif* `m` *rentré au clavier par l'utilisateur et qui affiche les* `m` *premiers entiers parfaits.*

   Correction :

```
import java.util.Scanner;

class GenParfait{

static boolean diviseurpropre(int i,int n){
return (n%i==0 && n!=i);
}


static boolean parfait (int n){
int res=0;
    for (int i=1;i<n;i++)
        if (diviseurpropre(i,n)) res=res+i;
    return (n==res);
}
```

```
public static void main(String[] args){
int m;
System.out.print("Rentrez un nombre : ");
Scanner sc = new Scanner(System.in);
m = sc.nextInt();
System.out.print("Les "+m+" premiers nombres parfait sont : ");

int compteur=0;
for (int i=1;compteur<m;i++){
    if (parfait(i)) {compteur++;System.out.print(i+", ");}
}
System.out.println("");
}

}
```

Exercice 2 (10 points) *Un tableau d'entiers de longueur* `n` *est une* **permutation** *s'il contient les entiers* `0`,`1`,...,`n-1`, *dans un ordre quelconque. Par exemple, le tableau* `{3,0,1,2}` *est une permutation, mais le tableau* `{3,4,1,2}` *n'en est pas.*

On s'intéresse à deux permutations particulières:

- **L'identité** *de longueur* `n` *est le tableau* `{0,1,...,n-2,n-1}`. *Par exemple la permutation identité de longueur 4 est le tableau* `{0,1,2,3}`.
- **L'inversion** *de longueur* `n` *est le tableau* `{n-1,n-2,...,1,0}`. *Par exemple la permutation inversion de longueur 4 est le tableau* `{3,2,1,0}`.
- *Une permutation a une* **marche montante** *si elle contient deux entiers consécutifs dans deux positions consécutives. Par exemple, la permutation* `{3,1,2,0}` *a une seule marche montante, la permutation identité de longueur* `n` *a* $(n-1)$ *marches montantes, et la permutation inversion n'en a aucune.*

1. *Écrire une fonction* `estId` *qui prend en argument un tableau d'entiers et qui teste si son argument est une permutation identité.*

   Correction :

   ```
   static boolean estId(int[]  t){
       for (int i=0 ; i<t.length; i++)
           if ( t[i] != i ) return false;
       return true;
   }
   ```

2. *Écrire une fonction* `estInv` *qui prend en argument un tableau d'entiers et qui teste si son argument est une permutation inversion.*

   Correction :

```
static boolean estInv(int[]  t){
      for (int i=0; i<t.length; i++)
          if ( t[i]+i != t.length-1 ) return false;
      return true;
   }
```

3. *Écrire une fonction* `nbMarchesMontantes` *qui prend en argument un tableau d'entiers, censé être une permutation, et qui renvoie le nombre de ses marches montantes. Il n'est pas demandé de vérifier que l'argument soit effectivement une permutation.*

   Correction :

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