université PARIS DIDEROT PARIS 7

EXAMEN INTRO À L'INFORMATIQUE & À LA PROGRAMMATION (IF1) 19/12/2013 08^{30}-11^{30}

Aucun document. Aucune machine. Barème indicatif. Exercices tous indépendants.
Une question peut toujours être traitée en utilisant les précédentes (traitées ou non).
Les morceaux de code Java devront être clairement présentés, indentés et commentés.

Exercice 1. (2,5 points) Expliquer ce que produit l'exécution de chacun des programmes suivants :

```
class Flup{
  static int pop(int a){
    if(a>10) return 2*a;
    else{
      int s=0;
      for(int i=0; i<a; i++)
        s=s+i;
      return s;
    }
  }
  public static void main(String[] args){
    int v=pop(12);
    System.out.println("1: v="+v);
    v=pop(4);
    System.out.println("2: v="+v);
  }
}
```

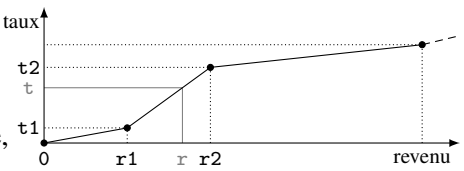
```
class Clike{
  static int ck(int n, boolean b){
    System.out.println("Par ici, n="+n+" et b="+b);
    if(n<3) return n;
    if(b) return ck(n%4==2,n/4);
    else  return ck((3*n+1)/2,n%4==3);
  }
  static int ck(boolean b, int n){
    System.out.println("Par là, b="+b+" et n="+n);
    if(n<3) return n;
    if(b) return ck(n%4==0,(3*n+2)/2);
    else  return ck((3*n+1)/2,n%4==1);
  }
  public static void main(String[] args){
    System.out.println(ck(true,11));
  }
}
```

Exercice 2. (4 points) Un *diviseur strict* d'un entier positif `k` est un entier positif diviseur de `k` et distinct de `k`.

1. Écrire une fonction `estDiviseurStrict` qui prend deux arguments entiers et teste s'ils sont positifs et si le premier divise strictement le second. Ainsi les valeurs de `estDiviseurStrict(k,284)` sont `false`, `true`, `false` et `false` pour `k` valant `-4`, `4`, `5` et `284` respectivement.
2. Écrire une fonction `sommeDiviseursStricts` qui prend un argument entier et renvoie la somme de ses diviseurs stricts s'il est positif et `-1` sinon. Ainsi `sommeDiviseurs(284)` renvoie 220 (somme de `1,2,4,71,142`).
3. Selon Leonhard Euler, deux entiers sont *amicaux* si chacun d'eux est égal à la somme des diviseurs stricts de l'autre. Écrire une fonction `sontAmicaux` qui teste si les deux arguments entiers sont positifs et amicaux. Ainsi, `sontAmicaux(10,14)` renvoie `false` et `sontAmicaux(220,284)` renvoie `true`.
4. Pour tout entier positif `k`, on définit sa suite *aliquote* : le premier terme est `k` et chaque terme suivant est la somme des diviseurs stricts du terme précédent. Si la suite atteint `1`, elle s'arrête car `1` ne possède pas de diviseur strict. Écrire une fonction `afficherAliquote` qui prend deux arguments entiers `k` et `n` et affiche les `n` possibles premiers termes de la suite aliquote de `k`. Ainsi `afficherAliquote(14,3)` affiche `14 10 8` et `afficherAliquote(14,n)` affiche `14 10 8 7 1` pour toute valeur de `n` supérieure à 5.

Exercice 3. (4,5 points) Un choc de simplification en matière fiscale consisterait à utiliser des taux effectifs d'imposition. Pour un nouvel impôt sur le revenu, le taux pourrait ne dépendre que du revenu (brut mensuel individuel en euro) selon une échelle comme celle-ci : *source :* http://www.revolution-fiscale.fr/

revenu	0	1100	2200	5000	10000	40000	100000 et +
taux	0.0%	2.0%	10.0%	13.0%	25.0%	50.0%	60.0%



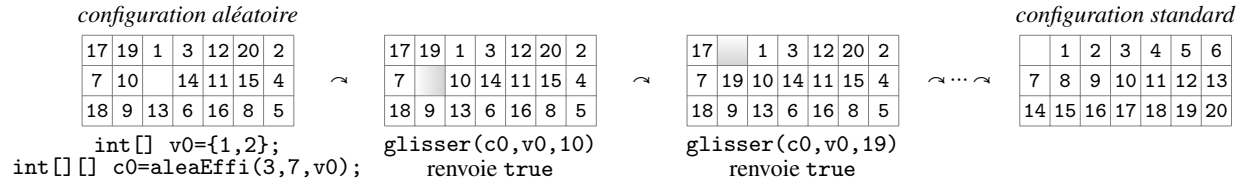
Pour un revenu compris entre deux des sept revenus références de cette échelle, le taux serait proportionnel aux taux correspondants.

1. Écrire une fonction `taux` qui prend en arguments trois revenus `r1`, `r`, `r2` (entiers en euro) et deux taux `t1`, `t2` (réels) et renvoie le taux `t` correspondant au revenu `r` en supposant que `r` est compris entre deux revenus références `r1` et `r2` auxquels correspondent les taux `t1` et `t2`. Par exemple :
   `taux(0, 825, 1100, 0.0, 2.0)` renvoie `1.5`
   `taux(0, 1100, 1100, 0.0, 2.0)` et `taux(1100, 1100, 2200, 2.0, 10.0)` renvoient `2.0`
   `taux(1100, 1650, 2200, 2.0, 10.0)` renvoie `6.0`

2a. Écrire une fonction `impot` qui renvoie l'impôt (arrondi à l'euro inférieur) calculé en appliquant le taux correspondant au revenu en argument (entier en euro) selon l'échelle ci-dessus. Par exemple :
`impot(825)` renvoie 12 `impot(1100)` renvoie 22 `impot(1650)` renvoie 99 `impot(150000)` renvoie 90000

2b. Pour faciliter les tests et comparer différentes échelles, on veut pouvoir passer l'échelle en argument du calcul. Écrire une fonction `impot` qui prend en arguments un revenu `r` (entier en euro) et deux tableaux `revenu` (d'entiers) et `taux` (de réels) de même longueur et qui renvoie l'impôt (arrondi à l'euro inférieur) calculé en appliquant le taux correspondant au revenu `r` selon l'échelle codée par `revenu` et `taux`.

Exercice 4. (9 points) Le taquin est un jeu dont le principe est de reconstituer un objet dans sa configuration standard à partir d'une configuration aléatoire :



L'objet est une grille rectangulaire composée de tuiles numérotées de `1` à `haut*larg-1` qui peuvent glisser les unes par rapport aux autres en utilisant une place vide. Un tel objet sera codé par un tableau bidimensionnel d'entiers (l'entier `0` codant la place vide) dont les valeurs se déplaceront à chaque changement de configuration.

1. Écrire une fonction `estStandard` qui teste si le tableau bidimensionnel d'entiers en argument code une configuration standard.
2. Écrire une fonction `estConfiguration` qui teste si le tableau bidimensionnel d'entiers en argument *peut* coder une configuration, au sens où il doit être rectangulaire (avoir ses lignes d'une même longueur) et contenir chacun des entiers compris entre `0` largement et le produit de sa hauteur par sa largeur strictement.
3. Écrire une fonction `chaine` qui renvoie une chaîne de caractères représentant la configuration en argument.
4. Écrire une fonction `glisser` qui prend en arguments une configuration `c`, un tableau `pos` de deux entiers donnant la position de la place vide dans `c` (pour ne pas avoir à la rechercher) et un entier `num` et qui teste si une des deux, trois ou quatre tuiles adjacentes à la place vide porte le numéro `num`, auquel cas modifie le contenu de `c` et de `pos` de sorte que la tuile numérotée `num` et la place vide soient échangées.
5. Écrire une fonction `jouer` qui prend en arguments une configuration `c` et un tableau `pos` donnant la position de la place vide et qui met le jeu en œuvre : tant que `c` n'est pas standard, on l'affiche et on demande à l'utilisateur quelle tuile glisser (jusqu'à ce qu'une tuile adjacente à la place vide soit entrée).

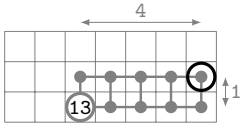
Deux méthodes permettent d'engendrer une configuration aléatoire à partir de laquelle on retrouve la configuration standard par glissements : une méthode naïve qui consiste à prendre une configuration standard et à glisser les tuiles aléatoirement jusqu'à avoir une certaine *taxi-distance* suffisante, une méthode efficace qui utilise la notion de *signature* dont la parité permet de prédire si une configuration aléatoire pourra se ramener à la configuration standard par glissements.

Rappelons que `Math.random()` renvoie un `double` aléatoire compris entre `0` largement et `1` strictement.

6. La taxi-distance entre deux positions de tuiles est définie comme la distance minimum via des déplacements uniquement horizontaux ou verticaux.

6a. Écrire une fonction `abs` qui renvoie la valeur absolue d'un argument entier.

6b. Écrire une fonction `taxi` qui prend en arguments une configuration `c`, un indice `i` de ligne et un indice `j` de colonne et qui renvoie la taxi-distance de la tuile en position `i,j` dans `c` jusqu'à sa position dans la configuration standard.



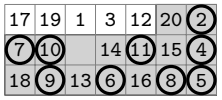
`taxi(c0,2,2)`
renvoie 5 (tuile 13)

6c. Écrire une fonction `taxi` qui prend une configuration en argument et renvoie la somme des taxi-distances de chacune de ses tuiles jusqu'à sa position dans la configuration standard.

6d. Écrire une fonction `aleaNaif` qui prend en arguments trois entiers `haut`, `larg`, `dist` et un tableau `pos` de deux entiers et qui renvoie une configuration aléatoire de hauteur `haut`, de largeur `larg`, de taxi-distance `dist` avec la place vide en position `pos` selon la méthode naïve.

7. La signature d'une tuile *t* est le nombre de tuiles qui sont à droite de *t* sur la ligne de *t* ou sont en dessous de la ligne de *t* et qui portent un numéro inférieur au numéro de *t*.

7a. Écrire une fonction `signature` qui prend une configuration en argument et renvoie la somme des signatures de toutes ses tuiles.



la signature de
la tuile 12 vaut 9

7b. Écrire une fonction `aleaEffi` indépendante de `aleaNaif` qui prend en arguments deux entiers `haut` et `larg` et un tableau `pos` de deux entiers et renvoie une configuration aléatoire de hauteur `haut` et de largeur `larg` avec la place vide en position `pos` et vérifiant la condition suivante : la somme des signatures de toutes ses tuiles est impaire si et seulement si `larg` est paire et `pos[0]` est impair. Si une configuration ne vérifie pas cette condition, il suffit d'en inverser deux tuiles adjacentes.