TP de programmation en Java

Christian Rinderknecht

4 février 2015

1 Conversion en binaire

Écrivez un programme `Binary.java` qui saisit sur la ligne de commande Unix un nombre entier positif en base dix et imprime sa représentation binaire. La méthode (on parle d'*algorithme*) à suivre est la suivante :

1. saisissez l'entier sur la ligne de commande ;
2. convertissez l'entier en un tableau de booléens tel que l'élément d'index i vaille `true` si le i-ème bit vaut 1 et `false` sinon ;
3. imprimez ce tableau de telle sorte que si l'élément d'index i vaut `true` alors on affiche `1` et `0` sinon.

Réalisez la première étape dans la méthode `main` de votre classe publique `Binary` (voyez la classe statique `Integer` pour convertir une chaîne de caractère en entier). Pour la seconde étape, écrivez une méthode publique et statique (dans `Binary`) nommée `convert`. Pour la dernière étape, définissez une méthode publique et statique `print`.

Avant d'entamer la dernière étape écrivez complètement `main` et n'écrivez que des méthodes `convert` et `print` qui ne font rien. Si vous et le compilateur êtes satisfait des types de ces méthodes, passez à la rédaction de leur contenu (appelé *corps*).

Dans le corps des méthodes, servez-vous uniquement des boucles, de la conditionnelle et des tableaux. La classe statique `java.lang.Math` fournit des méthodes correspondant aux opérations mathématiques usuelles. Le catalogue de la bibliothèque Java de Sun est consultable en ligne à l'adresse `http://java.sun.com/j2se/1.4.2/docs/api/`.

2 Factorielle

On rappelle que la fonction factorielle en mathématique se définit ainsi :

$$0! = 1$$
$$n! = n \times (n-1)!$$

En d'autres termes, si l'on renomme la factorielle en *fact*, on écrit de façon équivalente :

$$\text{fact}(0) = 1$$
$$\text{fact(n)} = n \times \text{fact(n-1)}$$

Ce type de définition de fonctions (ici c'est une suite entière) est dite *récursive* (ou *récurrente* pour les suites). En général, vous connaissez déjà depuis le lycée les suites de la forme $u_n = f(u_{n-1})$. Cela ne pose aucun problème à Java, la seule contrainte étant que vous devez vous assurer vous-même que l'appel de méthode récursif se fait avec des arguments plus petits pour garantir la terminaison du programme. Dans l'exemple de la suite, la méthode serait u et les arguments n et $n-1$.

Le programme correspondant à la factorielle s'écrit simplement

```
public static int fact (int n) {
  if (n == 0)
    return 1;
  else return n * fact (n-1);
}
```

3 Sommes

Le but est de programmer $\Sigma_{i=0}^{n} i^2$. Écrivez un programme `Sommes.java` contenant une classe Sommes qui contient une méthode publique et statique nommée carres qui prend l'entier correspondant à n et retourne l'entier Java correspondant à $\Sigma_{i=1}^{n} i^2$.

- Il faut que carres utilise une boucle pour calculer son résultat.
- Ensuite écrivez une méthode rec_carres qui calcule la même chose que carres mais de façon récursive (donc sans boucle). Pour y parvenir, cherchez uen fonction f telle que $u_n = f(u_{n-1})$ et $u_n = \Sigma_{i=0}^{n} i^2$.

4 Suite de Fibonacci

Programmez dans un fichier `Fibonacco.java` la suite de Fibonacci en suivant de près la définition mathématique usuelle

$$F_0 = F_1 = 1$$
$$F_{n+2} = F_{n+1} + F_n$$

5 Binôme de Newton

Programmer dans un fichier `Newton.java` le calcul du binôme de Newton :

$$C_n^0 = C_n^n = 1$$
$$C_n^p = C_{n-1}^p + C_{n-1}^{p-1}$$

6 Algorithme d'Euclide

Le mathématicien grec Euclide a proposé une méthode pour déterminer le plus grand diviseur commun (PGCD) de deux entiers. Soient a et b ces entiers et q et r le quotient de a/b et le reste de a/b, respectivement. Autrement dit :

$$a = b \times q + r \text{ où } r < b$$

Tout diviseur commun de a et b divise donc aussi r, car $r = a - b\,q$, et tout diviseur de b et r divise aussi a, car $a = b\,q + r$. Par conséquent, le PGCD de a et b égale le PGCD de b et r. Le reste étant strictement inférieur au diviseur, la répétition du procédé se terminera (la suite des restes est strictement décroissante). Le dernier reste non nul (ou le dernier diviseur) est alors le PGCD recherché. Par exemple, si $a = 96$ et $b = 81$, nous avons la suite de divisions

$$\begin{aligned} 96 &= 1 \times 81 + 15 \\ 81 &= 5 \times 15 + 6 \\ 15 &= 2 \times 6 + 3 \\ 6 &= 2 \times 3 + 0 \end{aligned}$$

Donc le PGCD de 96 et 81 vaut 3.

Il faut prendre garde que $a \geqslant b$. Si ce n'est pas le cas, il faut intervertir a et b. De plus, par définition, si $b = 0$ alors le PGCD est a.

Écrivez un programme `Euclide.java` qui contient une classe publique Euclide dont la méthode main saisit deux nombres entiers (le premier est considéré comme a et le second comme b), ainsi qu'une méthode publique et statique pgcd qui calcule le PGCD de deux entiers.

1. Dans un premier temps, utilisez une boucle pour calculer le PGCD.
2. Dans un second temps, définissez une méthode publique et statique nommée rec_pgcd qui calcule *récursivement* le PGCD (c'est-à-dire sans boucle). Pour cela elle suivra la définition mathématique du PGCD :
   - Si $b > a$ alors pgcd(a, b) = pgcd(b, a)
   - sinon si $b = 0$ alors pgcd(a, b) = a
   - sinon pgcd(a, b) = pgcd(b, a mod b), où *mod* est le reste de la division entière.

   Affichez le PGCD calculé par pgcd (avec une boucle) et par rec_pgcd (sans boucle).

7 Tri à bulles

Le but de cet exercice de vous faire programmer un algorithme de tri simple, dit *tri à bulles*. Pour comprendre le principe, supposons que nous voulions trier par ordre croissant une suite d'entiers.

Le principe est de comparer deux à deux les éléments du tableaux, de la gauche vers la droite. Si les deux éléments sont ordonnés l'un par rapport à l'autre, on passe à la paire suivante, sinon on les échange avant de passer à la paire suivante. Au bout de cette étape, la dernière case du tableau (la plus à droite) contient donc le plus grand élément. Par exemple 3 5 7 2 devient 3 5 2 7. Il faut alors recommencer avec le sous-tableau allant de la gauche jusqu'avant le plus grand élément précédemment trouvé. Dans notre exemple, il faut recommencer avec le sous-tableau 3 5 2, puisque 7 est bien placé. Cela donne 3 2 5 7. Maintenant 5 et 7 sont bien placés et l'on recommence avec le sous-tableau 3 2, ce qui donne 2 3 5 7. On s'arrête là car le sous-tableau restant est réduit à 2, qui est forcément trié.

Écrivez d'abord un fichier `Bubble.java` contenant une classe publique Bubble qui contient une méthode publique et statique print qui prend un tableau d'entier et l'affiche de gauche à droite à l'écran. Dans la fonction main, lisez les arguments passés en ligne de commande par l'usager, stockez-les dans un tableau d'entiers et affichez-les tels qu'ils ont été lus. Par exemple, si l'usager tape

```
java Bubble 3 5 7 2
```

alors l'affichage est

```
3 5 7 2
```

Ensuite, ajoutez à la classe Bubble une méthode publique et statique sort qui prend en paramètre un tableau d'entiers et ne renvoie rien (c'est-à-dire le type void). En effet, contrairement aux types primitifs de Java (tels int, boolean etc.), lorsque l'on modifie un paramètre de type tableau dans une méthode, la modification sera observable par l'appelant (de la méthode). Par exemple,

```
public class Toto {
  public static void sept_en_tete (int[] t) {
    if (t.length >= 1) t[0] = 7;
  }
  public static void main (String[] args) {
    int[] t = {1, 2, 3};
    sept_en_tete (t);
    System.out.println (t[0]); // Affiche 7 et non 1.
  }
}
```

Il n'est donc pas nécessaire que le type de retour de sort soit `int[]`, et void fait l'affaire.

Pour programmer ce tri il faut donc dans la méthode sort deux boucles imbriquées (c'est-à-dire l'une dans l'autre) portant sur deux indices (usuellement i et j).