Corrigé du contrôle continu n° 2

Informatique Fondamentale (IF121) — groupe B13

Contrôle du 17 décembre 2003

Barème : 5 points pour l'exercice 1a, 3 points pour 1b et 1c, 6 points pour l'exercice 2. 7 points pour les bases de la programmation (note « prog. ») ; 7/7 signifie que les bases sont acquises (structure générale, méthodes, variables, boucles, parité d'un entier, entrées-sorties). Le total sur 21 est compté sur 20.

Exercice 1

La justification du programme est donnée sous forme de commentaires dans le programme. La démarche générale est de traduire sous forme mathématique les raisons qui ont poussé à écrire le programme de cette façon. Un invariant relie en général les valeurs des variables que le programme modifie avec les données du problème initial.

```
import fr.jussieu.script.Deug;
class LogarithmeBinaire {
    static int lbe(int n) {
        int k = 0;
        int p = 1;
        // p = 1 = 2^0 = 2^k
        while (p <= n) {
            // Invariant : p = 2^k
            k = k + 1;
            p = 2 * p;
            // p = 2 * ancien_p = 2 * 2^ancien_k = 2^(ancien_k + 1) = 2^k
            // Terminaison : parce que p est entier, augmente, et reste <= n
        }
        // Sortie de la boucle ==> n < p = 2^k
        // Si n = 0 : la boucle est exécutée 0 fois, on a k = 0. Ok.
        // Si n > 0 : la boucle est exécutée au moins une fois ;
        //            au début du dernier tour, on a p <= n, donc
        //            le k final est le plus petit k tel que n < 2^k.
        return k - 1;
    }
    static void main (String[] args) {
        Deug.print("Logarithme binaire entier de ");
        int n = Deug.readInt();
        int k = lbe(n);
        Deug.println(k);
    }
}
```

Exercice 2

```
import fr.jussieu.script.Deug;
class Syracuse {
    static void main (String[] args) {
        Deug.print("Suite de Syracuse partant de ");
        int x = Deug.readInt();
        int n = 0;
        int max = x;
        while (x != 1) {
            n++;
            if (x % 2 == 0) {
                x = x / 2;
            } else {
                x = 3 * x + 1;
                if (x > max) max = x;
            }
        }
        Deug.println("Durée de vol : " + n);
        Deug.println("Hauteur de vol : " + max);
    }
}
```

Remarques générales

- Répartition des tâches (déjà dit en TP) : le premier exercice demande d'écrire une méthode qui effectue un calcul (calcul de x^n à partir de x et de n). Cette méthode n'a donc pas à lire les données du problème (elles sont passées en argument) ni à afficher le résultat (il doit être retourné par la méthode). C'est la méthode `main` (non demandée) qui se charge de la lecture des entrées et de l'affichage du résultat.
- Test de parité (déjà vu en TP). Soit n un entier. Par définition, n est pair si et seulement si il est multiple de 2, c'est-à-dire qu'il existe un entier k tel que $n = 2 \times k$. On ne peut pas exprimer cette propriété directement en Java. Une caractérisation équivalente est que n est pair si et seulement si le reste de la division de n par 2 est nul. Ceci s'écrit en Java : `n % 2 == 0`. Donc pour tester si le contenu de la variable `n` est pair en Java :
  ```
  if (n % 2 == 0) { /* code pour n pair */ } else { /* code pour n impair */ }
  ```
- Dans une conditionnelle `if (`*`condition`*`) ... else ...` , la condition est forcément fausse quand on exécute la branche « else » (« else » signifie « sinon »). Donc plutôt que d'écrire
  ```
  if (condition) { ... } else if (condition contraire) { ... }
  ```
  on peut écrire directement
  ```
  if (condition) { ... } else { ... }
  ```
- Les deux exercices appelaient l'écriture d'une boucle while. On ne pouvait pas s'en sortir avec une boucle for (ou alors, une boucle for plus compliquée que `for (int i=...; i<=...; i++)`).