Les boucles en programmation (algorithmique)

12-15 minutes

http://www.charlie-soft.com/Programmation/Algorithmique/Boucles.php

Les boucles, c'est généralement le point douloureux de l'apprenti programmeur. C'est là que ça coince, car autant il est assez facile de comprendre comment fonctionnent les boucles, autant il est souvent long d'acquérir les réflexes qui permettent de les élaborer judicieusement pour traiter un problème donné.

On peut dire en fait que les boucles constituent la seule vraie structure logique caractéristique de la programmation. Si vous avez utilisé un tableur comme Excel, par exemple, vous avez sans doute pu manier des choses équivalentes aux variables (les cellules, les formules) et aux tests (le `if`…). Mais les boucles, ça, ça n'a aucun équivalent. Cela n'existe que dans les langages de programmation proprement dits.

Donc pour la majorité d'entre vous, c'est quelque chose de totalement neuf, alors, à vos futures - et inévitables - difficultés sur le sujet, il y a trois remèdes : de la patience, de la rigueur, et encore de la patience!

A quoi ça sert ?

Prenons le cas d'une saisie au clavier, où par exemple, le programme pose une question à laquelle l'utilisateur doit répondre par O (Oui) ou N (Non). Mais tôt ou tard, l'utilisateur, facétieux ou maladroit, risque de taper autre chose que la réponse attendue. Dès lors, le programme peut planter soit par une erreur d'exécution (parce que le type de réponse ne correspond pas au type de la variable attendu) soit par une erreur fonctionnelle (il se déroule normalement jusqu'au bout, mais en produisant des résultats fantaisistes).

Alors, dans tout programme un tant soit peu sérieux, on met en place ce qu'on appelle un contrôle de saisie, afin de vérifier que les donnés entrées au clavier correspondent bien à celles attendues par l'algorithme. On pourrait essayer avec un `if`. Voyons voir ce que ça donne :

```
class SaisieOuiNon {
    void principal(){
        char rep;

        rep = SimpleInput.getChar("Voulez-vous un café (O/N) ");
        if (rep != 'O' && rep != 'N') {
             System.out.println("Réponse incorrecte ! ");
           rep = SimpleInput.getChar("Voulez-vous un café (O/N) ");
        }
    }
}
```

C'est impeccable. Du moins tant que l'utilisateur a le bon goût de ne se tromper qu'une seule fois, et d'entrer une valeur correcte à la deuxième demande. Si l'on veut également bétonner en cas de deuxième erreur, il faudrait rajouter un `if`. Et ainsi de suite, on peut rajouter des centaines de `if`, mais on n'en sortira pas, il y aura toujours moyen qu'un acharné flanque le programme par terre.

La solution consistant à aligner des `if` en pagaille est donc une impasse. La seule issue est donc de flanquer une structure de boucle, qui se présente ainsi :

```
while (booléen) {
  ...
  instructions
  ...
}
```

Le principe est simple : le programme arrive sur la ligne du `while`. Il examine alors la valeur du booléen (qui, je le rappelle, peut être une variable booléenne ou, plus fréquemment, une condition). Si cette valeur est `true`, le programme exécute les instructions qui suivent, jusqu'à ce qu'il rencontre l'accolade fermante (}) de la boucle. Il retourne ensuite sur la ligne du `while`, procède au même examen, et ainsi de suite. Le manège enchanté ne s'arrête que lorsque le booléen prend la valeur false.

Mise en pratique

Donner le nombre d'exécution de la boucle des programmes suivants :

```
x = 3;
v =-12;
while (x > v) {
    x = (-2) * x;
}
```

```
x = -3;
v = -2;
while (x < 10) {
    x = x + v;
}
```

```
x = -1;
v = -1;
while (x < 5 et v < 50) {
    x = x + 1;
    v = v * x;
}
```

```
x = -1;
v = -1;
rester = false;
while (rester) {
    rester = (x + 2) > v;
    v = v + 2;
    x = x + 1;
}
```

Où on trouve la solution.

Reprenons notre problème de contrôle de saisie. Une première approximation de la solution consiste à écrire :

```
class SaisieOuiNon {
    void principal(){
        char rep;

        System.out.println("Voulez-vous un café (O/N) ");
        while (rep != 'O' && rep != 'N') {
             System.out.println("Réponse incorrecte ! ");
             rep = SimpleInput.getChar("Voulez-vous un café (O/N) ");
        }
    }
}
```

Là, on a le squelette de l'algorithme correct. Mais de même qu'un squelette ne suffit pas pour avoir un être vivant viable, il va nous falloir ajouter quelques muscles et organes sur cet algorithme pour qu'il fonctionne correctement.

Son principal défaut est de provoquer une erreur à chaque exécution. En effet, l'expression booléenne qui figure après le `while` interroge la valeur de la variable `rep`. Malheureusement, cette variable, si elle a été déclarée, n'a pas été affectée avant l'entrée dans la boucle. On teste donc une variable qui n'a pas de valeur, ce qui provoque une erreur et l'arrêt immédiat de l'exécution. Pour éviter ceci, on n'a pas le choix : il faut que la variable `rep` ait déjà été affectée avant qu'on en arrive au premier tour de boucle. Pour cela, on peut faire une première lecture de `rep` avant la boucle. Dans ce cas, celle-ci ne servira qu'en cas de mauvaise saisie lors de cette première lecture. L'algorithme devient alors :

```
class SaisieOuiNon {
    void principal(){
        char rep;

        rep = SimpleInput.getChar("Voulez-vous un café (O/N) ");
        while (rep != 'O' && rep != 'N') {
             System.out.println("Réponse incorrecte ! ");
             rep = SimpleInput.getChar("Voulez-vous un café (O/N) ");
        }
    }
}
```

Une autre possibilité, fréquemment employée, consiste à ne pas lire, mais à affecter arbitrairement la variable avant la boucle. Arbitrairement ? Pas tout à fait, puisque cette affectation doit avoir pour résultat de provoquer l'entrée obligatoire dans la boucle. L'affectation doit donc faire en sorte que le booléen soit mis à `true` pour déclencher le premier tour de la boucle. Dans notre exemple, on peut donc affecter `rep` avec n'importe quelle valeur, hormis « O » et « N » : car dans ce cas, l'exécution sauterait la boucle, et `rep` ne serait pas du tout lue au clavier.

Remarque

Le danger lorsque l'on fait des boucles, c'est de mal les concevoir. C'est notamment le cas lorsque l'on créé une boucle dont la condition sera toujours fausse. Le programme ne rentre alors jamais dans la superbe boucle sur laquelle vous avez tant sué !

Mais la faute symétrique est au moins aussi désopilante.

Elle consiste à écrire une boucle dans laquelle le booléen ne devient jamais `false`. L'ordinateur tourne alors dans la boucle comme un dératé et n'en sort plus. Seule solution, quitter le programme avec un démonte-pneu ou un bâton de dynamite. La boucle infinie est une des hantises les plus redoutées des programmeurs.
Mise en pratique :

Faites les algorithmes suivants :

1. Écrire un algorithme qui demande à l'utilisateur un nombre jusqu'à ce que ce nombre soit compris entre 1 et 3.
2. Écrire un algorithme qui permette de calculer X à la puissance Y. Pour rappel, X2=X*X, X3=X*X*X, X4=X*X*X*X …
3. Écrire un algorithme qui reproduit le fonctionnement d'une caisse enregistreuse. C'est-à-dire que l'ordinateur va demander à l'utilisateur de saisir le montant des articles tant que le prix est différent de zéro, puis qui affiche la somme du montant de tous les articles.