Chapitre III

Les structures de contrôle

Sommaire

I. Introduction

Si l'on ne pouvait qu'indiquer à l'ordinateur une suite inamovible d'instructions à répéter pour accomplir sa tâche, on écrirait des algorithmes bien peu adaptables.

Par exemple, si l'on souhaite écrire un programme sachant résoudre une équation du premier degré $ax + b = 0$, on doit prévoir le cas où $a = 0$. Or nous ne savons pas encore comment réaliser cela.

De la même façon, si l'on veut calculer la somme des n premiers entiers, n étant donné, il va falloir effectuer un nombre variable d'addition, ce que nous ne savons pas non plus faire.

Les *structures de contrôle* sont là pour permettre à l'ordinateur d'accomplir des actions plus complexes. Ce chapitre va les étudier en détail.

II. Les conditionnelles

La structure la plus simple est la *conditionnelle*. On veut exécuter une action ou une autre selon qu'un test est vrai ou faux.

Par exemple, voici un programme résolvant une équation du premier degré $ax+b = 0$, en envisageant les différents cas possibles :

```
program premier_degre;

var a,b : real;
begin
   writeln('Resolution de l''equation ax+b=0');
   write('Valeur de a : ');
   readln(a);
   write('Valeur de b : ');
   readln(b);
   if a = 0.0
   then        (* equation 0x+b=0 *)
      if b = 0.0
      then     (* equation 0=0 *)
         writeln('Tous les reels sont solutions.')
      else     (* equation b=0 *)
         writeln('Aucun reel n''est solution.')
   else        (* cas general *)
      writeln('Une solution : ',-b/a)
end.
```

On voit dans le code de ce programme qu'on a envisagé l'ensemble des cas possibles : le cas où $a = 0$, qui se subdivise en deux sous-cas : $b = 0$ ou $b \neq 0$, et le cas général $a \neq 0$. Une seule des trois instructions `writeln` est exécutée à chaque lancement du programme.

La structure conditionnelle s'écrit sous deux formes :

Algorithmique	Traduction Pascal
Si <condition> Alors <action>	**if** <condition> **then** <action>;

Dans cette première forme, le bloc d'instruction `<action>` n'est exécuté que si `<condition>` est vrai. Dans le cas contraire, aucune instruction n'est exécutée, on passe directement à l'instruction suivante.

Algorithmique	Traduction Pascal
Si <condition> Alors <actionV> Sinon <actionF>	**if** <condition> **then** <actionV> **else** <actionF>;

Dans cette deuxième forme, l'un des deux blocs `actionV` ou `actionF` est exécuté, selon la valeur de vérité du booléen `<condition>`.

Notons que si l'un des blocs `action` contient plusieurs instructions, il doit être encadré par une paire `begin ... end`, de manière à être vu comme une seule instruction.

Un cas plus général de conditionnelle permet de traiter le cas d'une variable pouvant prendre plusieurs valeurs différentes (par exemple un caractère ou un chiffre lu au clavier) : l'instruction `case`. Je vous invite à en lire la documentation[1] si vous en avez un jour besoin.

III. Les boucles

1. Boucle itérative : `for`

La boucle `for` est la boucle la plus simple et la moins flexible. Elle permet de répéter une séquence d'instructions un nombre fixé de fois.

Algorithmique	Traduction Pascal
Pour <indice> de <depart> à <fin> faire <action>	**for** <indice> := <depart> **to** <fin> **do** <action>;

Ici, l'instruction (ou le bloc d'instruction) `<action>` est répétée pour toutes les valeurs de l'`indice` comprise (inclusivement) entre `depart` et `fin`. `indice` doit être d'un type ordonné, comme `integer`, `char` ou `boolean`[2], et augmente d'une unité à chaque passage dans la boucle.

Voici par exemple un programme demandant un entier n, et affichant la somme des entiers compris entre 1 et n :

```
program somme_entiers;

var i,n,s : integer;

begin
   write('Entrer l''entier n : ');
   readln(n);
   s := 0;
   for i := 1 to n do      (* a l'etape i, s contient 1+2+...+(i-1) *)
      s := s + i;          (* a la sortie, s contient 1+2+...+i
*)
   writeln('La somme des entiers compris entre 1 et ',n,' est : ',s);
end.
```

Comme on peut le voir dans cet exemple, on peut utiliser la valeur de `indice` dans le bloc d'instruction `action` (ici réduit à une seule instruction.

2. Boucles conditionnelles : `repeat` et `while`

Les boucles `repeat... until` (répéter... jusqu'à) et `while... do` (tant que... faire) sont ce qu'on appelle des *boucles conditionnelles*. La condition de sortie (pour la boucle `repeat`) ou d'entrée (pour

[1] `http://wiki.freepascal.org/Case`

[2] Hé oui, il est possible d'écrire quelque chose de la forme `for i from false to true do ...` !

la boucle `while`) de boucle est une conditionnelle dont la valeur de vérité est testé à chaque tour de boucle.

Algorithmique	Traduction Pascal
Répéter <action> Jusqu'à <condition>	**repeat** <action> **until** <condition>
Tant que <condition> faire <action>	**while** <condition> **do** <action>

La différence essentielle entre les deux boucles tient au moment où est effectué le test : avant d'entrer dans la boucle pour la boucle `while`, après le corps de boucle pour la boucle `repeat`. Cela a une conséquence importante sur le nombre de fois qu'est exécuté le corps de la boucle :

- dans le cas d'une boucle `while`, on sort de la boucle si la condition est fausse, ce qui peut se produire dès la première fois, auquel cas le corps de la boucle ne sera jamais exécuté,
- alors que pour une boucle `repeat`, on sort de la boucle si la condition est vraie, mais le corps de boucle est de toute façon exécuté *au moins une fois*.

Tout dépend donc de ce dont on a besoin. En toute rigueur, on pourrait se passer de la boucle `repeat`, en la simulant par une boucle `while` dans laquelle on initialise artificiellement la condition d'entrée dans la boucle à "vrai"[3]. Certains langages ne l'implémente d'ailleurs pas.

Voici par exemple un programme demandant un entier n, et indiquant s'il est premier ou pas, en testant sa divisibilité par tous les entiers[4] compris entre 2 et $n-1$ la somme des entiers compris entre 1 et n :

```
program premier;

var n : integer;

function est_premier(m : integer) : boolean;

var b,i : integer;

begin
   i := 2;
   est_premier := true;
   b := m;
   while (i<b) and est_premier do
      if m mod i = 0 then est_premier := false
      else i := i+1
end;

begin
   write('Entrer l''entier n : ');
   readln(n);
   write('L''entier ',n,' ');
   if est_premier(n) then writeln('est premier.')
   else writeln('n''est pas premier.');
   readln
end.
```

[3] et en inversant la valeur de vérité de la condition, celle-ci devenant une condition pour rester dans la boucle et non plus pour en sortir !

[4] On verra en TD comment améliorer drastiquement les performances de ce programme.