# CH 6 : Les structures de contrôle
## *a) Les structures conditionnelles*

## I. Structures de contrôle

En informatique, on appelle **structure de contrôle** une commande qui contrôle l'ordre dans lequel les différentes instructions d'un programme sont exécutées. En **Scilab**, on peut distinguer plusieurs types de structures de contrôle :

***a)*** les **structures séquentielles** : présentées dans le « CH 2 : Premiers pas vers la programmation ». Les différentes commandes sont exécutées les unes à la suite des autres *i.e.* dans un ordre séquentiel.

***b)*** les **structures conditionnelles** : qui permettent d'introduire des branchements conditionnels dans le programme. Un programme peut comporter plusieurs branches. Chacune d'elle est associée à une condition. La branche exécutée est la première dont la condition est réalisée.

***c)*** les **structures itératives** : qui permettent d'effectuer la répétition (*i.e.* l'itération) de commandes. Ces répétitions peuvent s'effectuer un nombre de fois fixé explicitement par le programmeur ou peuvent avoir lieu tant qu'une condition n'est pas réalisée.

Dans ce chapitre, nous nous intéressons aux structures conditionnelles. Le prochain sera consacré aux structures itératives.

## II. Structures conditionnelles : le `if`

### II.1. Structure conditionnelle à deux branches

#### II.1.a) Syntaxe de la commande

En programmation, il est parfois utile de réaliser un branchement : **si** une **condition** est réalisée **alors** le programme exécute les instructions de la première branche ; **sinon** le programme exécute les instructions de la deuxième branche. La syntaxe utilisée dans les langages de programmation, basée sur les mots clés `if`, `then`, `else`, est présentée ci-dessous.

```
if condition then
  instruction_1
else
  instruction_2
end
```

Détaillons les éléments de cette syntaxe.

- *condition* : représente une expression **Scilab** dont l'évaluation est soit `%t` soit `%f`. Autrement dit, un objet de type `boolean`.
- $instruction_i$ : représente une séquence de commandes **Scilab**, qu'on appelle aussi un bloc d'instructions.

#### II.1.b) Ordre d'exécution

L'exécution d'une structure conditionnelle à deux branches s'effectue de la manière suivante :

***1)*** **si** *condition* est réalisée, *i.e.* si cette condition est évaluée à `%t` **alors** le bloc $instruction_1$ est exécuté.

***2)*** **sinon**, autrement dit si *condition* n'est pas réalisée (*i.e.* évaluée à `%f`), le bloc $instruction_2$ est exécuté.

## II.2. Structures conditionnelles imbriquées

### II.2.a) Syntaxe

Il faut bien comprendre que les blocs $instruction_1$ et $instruction_2$ sont des séquences d'instructions. Ces blocs peuvent donc comporter des structures conditionnelles. Par exemple, la deuxième branche d'une conditionnelle peut déboucher sur une nouvelle structure à deux branches, créant ainsi une structure conditionnelle à trois branches. La syntaxe d'une telle construction est la suivante.

```
if condition_1 then
  instruction_1
else
  if condition_2 then
    instruction_2
  else
    instruction_3
  end
end
```

On peut itérer ce procédé : le bloc $instruction_3$ peut lui aussi être un conditionnelle et ainsi de suite. Cela permet de créer des structures conditionnelles comportant un nombre (fini) quelconque de branches.

### II.2.b) Ordre d'exécution

L'exécution de la structure imbriquée précédente est la suivante.

*1)* **si** $condition_1$ est réalisée, **alors** le bloc $instruction_1$ est exécuté.

*2)* **sinon**, autrement dit si $condition_1$ n'est pas réalisée :

- ***a)*** soit $condition_2$ est réalisée, auquel cas le bloc $instruction_2$ est exécuté.
  *(cas où $condition_1$ non réalisée et $condition_2$ réalisée)*
- ***b)*** soit $condition_2$ n'est pas réalisée, auquel cas le bloc $instruction_3$ est exécuté.
  *(cas où $condition_1$ et $condition_2$ non réalisées)*

## II.3. Structure conditionnelles à $n$ branches

### II.3.a) Syntaxe

La syntaxe précédente décrivant une conditionnelle à trois branches est un peu lourde. L'écriture de conditionnelles à $n$ branches est possible via la syntaxe allégée suivante.

```
if        condition_1      then
  instruction_1
elseif    condition_2      then
  instruction_2
...
elseif    condition_{n-1}  then
  instruction_{n-1}
else
  instruction_n
end
```

### II.3.b) Ordre d'exécution

L'exécution d'une telle structure à $n$ branches s'effectue comme suit.

*1)* **si** $condition_1$ est réalisée, **alors** le bloc $instruction_1$ est exécuté.

*2)* **sinon**, **si** $condition_2$ est réalisée, **alors** le bloc $instruction_2$ est exécuté.
*(cas où $condition_1$ non réalisée et $condition_2$ réalisée)*

...

***n-1)*** **sinon**, **si** $condition_{n-1}$ est réalisée, **alors** $instruction_{n-1}$ est exécuté.
*(cas où $condition_1$, ..., $condition_{n-2}$ non réalisées et $condition_{n-1}$ réalisée)*

***n)*** **sinon**, le bloc $instruction_n$ est exécuté.
*(cas où $condition_1$, ..., $condition_{n-1}$ non réalisées)*

### II.4. Le mot clé `else`

La dernière branche d'une structure conditionnelle est portée par le mot clé `else` mais n'est pas suivie d'une condition. En conséquence :

- soit au moins l'une des conditions est vérifiée. Le bloc $instruction_i$ correspondant à la première condition réalisée sera alors exécuté.
- soit aucune condition n'est réalisée et le bloc $instruction_n$ est exécuté.

Ainsi, quoiqu'il arrive, l'un des blocs $instruction_i$ sera exécuté. On dit, dans ce cas, que le filtrage est **exhaustif**.

Formellement, il n'y a pas d'obligation, au niveau langage, à n'utiliser que des filtrages exhaustifs. L'utilisation du mot clé `else` peut donc être considéré comme une nouvelle règle de bonne conduite.

## III. Écrire des conditionnelles

### III.1. Écrire des tests à l'aide d'opérateurs de comparaison

Les opérateurs permettant d'écrire des conditions ont déjà été présentés, notamment lors de la description du type `boolean` au « CH 4 : Types de données en **Scilab** ». Essentiellement, on dispose :

- des opérateurs de comparaison `>`, `<`, `>=`, `<=`, `==` qui combinent des objets de type `constant` pour former des expressions de type `boolean`.
- des opérateurs `|`, `&`, $\sim$ qui permettent de combiner des expressions de type `boolean` pour en former de nouvelles.

### III.2. Écrire de structures conditionnelles

Maintenant que nous avons détaillé d'un point de vue théorique le rôle des structures conditionnelles, on s'intéresse à l'intégration de celles-ci dans les programmes **Scilab**. Pour illustrer notre propos, nous considérons de nouveau le programme présenté au « CH 3 : Premiers programmes en **Scilab** » consistant à calculer les racines réelles d'un polynôme du second degré.

Il manquait jusque là un mécanisme de test permettant de s'assurer du caractère positif du discriminant avant d'en calculer la racine carrée. On se propose donc de mettre jour le programme en y incluant une structure conditionnelle.

```
// Le script suivant demande la valeur d'un polynôme
// et affiche ses deux racines réelles si elles existent
// ou un message "Attention..." dans le cas contraire
a = input("Entrez la valeur du coefficient a : ");
b = input("Entrez la valeur du coefficient b : ");
c = input("Entrez la valeur du coefficient c : ");
delta = discrim(a, b, c);
p = string(a)+"X^2 + "+string(b)+"X + "+string(c);
if delta >= 0 then
  xPlus = (-b + sqrt(delta))/(2*a);
  xMoins = (-b - sqrt(delta))/(2*a);
  disp("Voici les deux racines du polynôme " + p);
  disp("La plus grande racine vaut " + string(xPlus));
  disp("La plus petite racine vaut " + string(xMoins));
else
  disp("Attention " + p + " n''a pas de racines réelles")
end
```

Cette modification permet d'obtenir, lors de l'exécution, l'affichage suivant.

```
--> exec('/Info/Exemple_cours/racPDispInp.sce', -1)
Entrez la valeur du coefficient a : 3
Entrez la valeur du coefficient b : 1
Entrez la valeur du coefficient c : 4

  Attention 3X^2 + 1X + 4 n'a pas de racines réelle
```