TP 4 – Structures conditionnelles

1 A quoi ça sert ?

En mathématiques et en algorithmique, il est courant de devoir distinguer les cas pour conclure à un résultat (par exemple pour calculer la valeur absolue d'un réel, ou pour déterminer le nombre de racines d'un trinôme, ...). On utilise pour cela des **structures conditionnelles** ou **structures if**. Il en existe trois sortes sous Scilab, nous allons les découvrir.

Avant toute chose, lorsqu'on veut utiliser une structure conditionnelle, on a besoin d'une **condition**. En algorithmique, cette condition est basée sur le **résultat d'un test**.

2 Faire un test sous Scilab

Les conditions, ou tests, sous Scilab s'expriment toujours grâce aux **opérateurs de comparaison**. Par exemple on peut tester :

- si la variable k est supérieure à 1,
- si le fonction ln calculée en la variable x est égale à 12,
- ou même si la variable t est supérieure à 2 et inférieure à 7.

Pour écrire un test sous ce logiciel, on utilisera donc :

- les variables, fonctions, constantes, etc, que nous avons déjà étudiées.
- les opérateurs de comparaison suivants :

  ==, <>, >, <, >=, <=

  Les 4 derniers sont faciles à comprendre, mais attention l'égalité se note == (et non = qui sert à l'affectation). Et la non égalité se note <> (et équivaut donc à $\neq$).
- et en plus, si on en a besoin, les connecteurs logiques : le **ET logique se note** & et le **OU logique se note** |.

Lors d'un test, Scilab renvoie F si le test est faux ou T si le test est vrai.

Taper les tests suivants et observer la réponse de Scilab :

```
--> a=2
--> a<3
--> a<=3
--> a==3

--> a=4
--> a<3
--> a<=3
--> a==3
```

```
--> a=2
--> a<3
--> a<=3
--> a==3
--> a<>3

--> b=3
--> a<=3 & b>4
--> a<=3 & b<4
--> a<=3 | b>4
```

Attention à ne pas confondre `a=b` (**affectation**) avec `a==b` (**test**). Ainsi `1==a` est possible mais pas `1=a`.

Maintenant que vous savez écrire des conditions, voyons comment utiliser les structures if.

3 La structure « if ... then ... end »

Cette structure est la plus simple. On comprend aisément comment Scilab va l'interpréter : **si** la condition est vérifiée **alors** il fera l'action demandée. Sa syntaxe est la suivante :

```
if condition then
   bloc d'instructions
 end
```

Tester cette structure avec le programme trivial suivant :

```
x=1;
if x<=2 then
     disp('le nombre est inferieur à 2')
end
```

Que se passe-t-il si on remplace la première ligne par $x = 3$? Pourquoi ?

Exercice 1. Ecrire un programme qui demande à l'utilisateur de rentrer les coefficients a, b et c d'un trinôme, qui calcule le discriminant et qui, si $\Delta > 0$, calcule les racines et affiche la phrase "Ce trinôme à deux racines, ce sont $x = ...$ et $y = ...$

4 La structure « if ... then ... else ... end »

Cette structure est un peu plus évoluée que la précédente. Elle permet de donner une action alternative lorsque la condition n'est pas vérifiée. Sa syntaxe est la suivante :

```
if condition then
     bloc d'instructions 1
     else
     bloc d'instructions 2
end
```

Tester cette structure avec le programme trivial suivant :

```
x=-3;
if x<0 then
     disp('x est un nombre négatif')
        else
     disp('x est un nombre positif')
end
```

Exercice 2. Ecrire un programme qui demande à l'utilisateur de rentrer un réel x et qui en donne sa valeur absolue (sans utiliser la commande *abs*).

5 La structure « if ... then ... elseif ... else ... end »

C'est la structure la plus évoluée sous Scilab. Elle permet, dans le cas où la première condition n'est pas vérifiée, de faire un nouveau test dont le résultat impliquera le choix d'une action 2 ou d'une action 3. On peut itérer cette commande autant de fois qu'on le souhaite.

Sa syntaxe est la suivante :

```
if condition 1 then
     bloc d'instructions 1
          elseif condition 2 then
              bloc d'instructions 2
....
          elseif condition k-1 then
              bloc d'instructions k-1
          else
              bloc d'instructions k
end
```

Tester cette structure avec le programme trivial suivant :

```
x=-3;
if x<0 then
     disp('x est un nombre négatif')
        elseif x>0 then
           disp('x est un nombre positif')
        else
           disp('x est nul')
end
```

Exercice 3. Améliorer le programme de l'Exercice 1 pour que l'ordinateur donne les racines du trinôme dans tous les cas de figure.

6 A vous de jouer !

Exercice 4. Ecrire un programme qui demande à l'utilisateur d'entrer deux nombres a et b et qui renvoie le plus petit des deux et le plus grand des deux.

Améliorer le programme précédent pour que l'utilisateur puisse entrer 5 nombres différents et que l'ordinateur renvoie le plus grand nombre et le plus petit.

Exercice 5. Ecrire un script qui demande à l'utilisateur s'il veut lancer un dé ou deux, lui affiche le résultat du lancé et lui dit « gagné » s'il n'obtient que des 6.

Vous pouvez vous renseigner sur la commande grand et l'utiliser.

Exercice 6. Ecrire un programme qui prend en entrée la note m d'un élève à un devoir surveillé et donne en sortie le message :

1. « Très bien », si $m \geq 13$
2. « Bien », si $8 \leq m < 13$.
3. « Courage, vous pouvez y arriver en travaillant », si $m < 8$.

Exercice 7. Ecrire un script qui simule la saisie du code confidentiel d'une carte bancaire, l'utilisateur ayant le droit à trois essais et le code étant une donnée initiale du programme.

Exercice 8. Ecrire un programme qui compte le nombre d'éléments divisibles par 7 entre 1 et n pour un nombre choisit par l'utilisateur.

Vous pouvez vous renseigner sur la commande modulo et l'utiliser.