

Maple

TP n°2 : Tests et boucles

I. Les expressions logiques

On suppose que l'on a : $a:=2$: $b:=3$: $c:=4$: $d:=5$:

Les expressions logiques sont des expressions dont le résultat est un booléen (`true` ou `false`).

```
a<b;  
evalb(a<b);
```

On peut faire intervenir les opérateurs logiques `and`, `or` ou `not` sur les résultats de type booléen.

```
(a<b) and (c>d);  
(a<b) or (c>d);  
not(a<b) or (c>d);
```

II. Les structures de contrôles

2.1 La structure de contrôle `if .. then`

L'instruction `if .. then` est appelée une instruction conditionnelle.

Sa syntaxe est la suivante :

```
if «condition1» then «instruction1»  
    elif «condition2» then «instruction2»  
    elif «condition3» then «instruction3»  
    :  
    elif «conditionN» then «instructionN»  
    else «instructionP»  
fi;
```

`elif` et `else` sont facultatifs et le nombre de champs `elif` est «illimité».

`elif` signifie "autrement si".

`fi`; est nécessaire et signale la fin de la structure.

Cette suite d'instructions effectue un test selon certaines conditions : `condition1`, `condition2`, ..., `conditionN` qui sont des expressions logiques.

Si la condition `condition1` est vérifiée, l'instruction `instruction1` est exécutée et on sort de la structure. Sinon, on effectue les différents tests optionnels : `condition2`, ..., `conditionN` dans l'ordre. Si l'une de ces conditions est vérifiée, on exécute l'instruction correspondante et on sort de la structure.

Si aucune condition n'est vérifiée et que la commande `else` est présente, c'est l'instruction suivant cette commande qui est exécutée.

S'il n'existe que deux alternatives de renvoi de valeur dans une instruction conditionnelle on peut utiliser une forme raccourcie de l'instruction `if .. then` dont la syntaxe est la suivante :

```
`if`(condition, val1, val2) ;
```

L'instruction `val1` est renvoyée si la condition est vrai, sinon c'est `val2`.

```
x:=Pi;  
a:=`if`(is(x,fraction),0,1);
```

Exercice 1

Résolution de l'équation du premier degré $ax + b = 0$ pour quelques valeurs de a et b .

```
a:=7:b:=3:  
if a<>0 then print(`Une solution : x `=-b/a)  
  elif b=0 then print(`Tout x est solution`)  
  else print(`Pas de solution`)  
fi;
```

1. Entrer et valider les lignes précédentes.
2. Modifier les valeurs de a et de b et valider les choix.

2.2 La structure de contrôle `for .. to`

La structure `for .. to` est une structure répétitive.

Sa syntaxe est la suivante :

```
for «var» from «init» to «fin» by «step» do  
  «calculs»  
od;
```

La commande `for .. to` exécute la ou les instructions de `calculs` pour une variable `var` allant d'une valeur initiale `init` à une valeur finale `fin`, avec un pas `step`.

`od;` signale la fin de la structure.

Les commandes `from` et `by` peuvent être omises ou écrites dans un ordre quelconque. Si l'on omet la commande `from`, la variable `init` vaut 1. Si l'on omet `by`, `step` vaut 1.

Par exemple, si l'on veut calculer de la somme des 100 premiers entiers naturels :

```
somme:=0:  
for k from 1 to 100 do  
  somme:=somme+k  
od:  
`somme `=somme;
```

Si l'on veut uniquement la somme des entiers impairs inférieurs à 100, il suffit d'introduire un pas de 2.

```
somme:=0:
for k to 100 by 2 do
    somme:=somme+k
od:
`somme `=somme;
```

On peut aussi utiliser une liste pour les différentes valeurs de la variable. La syntaxe est alors la suivante :

```
for «variable» in «expression» do «calculs» od;
```

Par exemple si on veut les nombres premiers de la liste [31,39,47,105], on peut utiliser :

```
for k in [31,39,47,105] do
    if isprime(k) then print(k,`est premier`) fi
od;
```

2.3 La structure de contrôle `while .. do`

La structure `while .. do` est une autre structure répétitive. Sa syntaxe est la suivante :

```
while «test» do «calculs» od;
```

Exécute la suite d'instructions `calculs`, tant que la condition `test` est vraie.
`od;` signale la fin de la structure.

Cette instruction peut remplacer l'instruction `for .. to`. De plus, elle peut créer des boucles sans connaître à l'avance le nombre d'itérations.

Par exemple, si l'on veut calculer la somme des 100 premiers entiers naturels, on peut aussi utiliser :

```
somme:=0:k:=-1:
while k<100 do
    k:=k+1:
    somme:=somme+k
od:
`somme `=somme;
```

Ou pour la somme des entiers impairs inférieurs à 100 :

```
somme:=0:k:=-1:
while k<99 do
    k:=k+2:
    somme:=somme+k
od:
`somme `=somme;
```

Remarques

La commande `break` permet de sortir de la structure de contrôle en cours. Cela évite des itérations inutiles.

Dans le même ordre d'idée, la commande `next` permet de passer à la valeur suivante dans une itération.

Exercice 2

On cherche à déterminer si un élément x fait partie d'une liste L .

```
L := [1, 2, 3, 4, 5]:
x := 2:
k := 1:
while k <= nops(L) do
  if L[k] = x then
    break;
  fi;
  k := k + 1:
od:
if k > nops(L) then
  print("x n'est pas un élément de L");
else
  print("x est un élément de L");
fi;
```

Changez la valeur de x et réévaluez.

Exercice 3

Inspirez vous de l'exercice 1 pour écrire une suite d'instructions gérant la résolution de l'équation du second degré $ax^2 + bx + c = 0$.

Exercice 4

1. Donner une suite d'instructions qui renvoie la somme $\sum_{k=1}^{1000} \frac{1}{k}$ sous forme exacte.
2. Donner une suite d'instructions qui renvoie le même résultat sous forme approchée (float).
3. Donner une suite d'instructions qui renvoie la somme des inverses des 1000 premiers entiers impairs.

Exercice 5

Déterminer les valeurs approchées des 15 premiers termes des suites suivantes :

- a. La suite u définie par $u_0 = -1$ et $u_{n+1} = \cos(u_n)$.
- b. Les suites u et v définies par $u_0 = 1$, $v_0 = 4$, $u_{n+1} = \sqrt{u_n \times v_n}$ et $v_{n+1} = \frac{u_n + v_n}{2}$.
- c. La suite u définie par $u_0 = 0$, $u_1 = 4$ et $u_{n+2} = \sqrt{u_n + u_{n+1}}$.

Exercice 6

Ecrire une suite d'instructions qui construise un tableau à deux dimensions $n \times m$ dont les valeurs sont le produit de l'indice de ligne et de l'indice de colonne.

Exercice 7

Déterminer le plus petit entier n tel que $n^2 - 3$ soit un multiple du carré d'un entier plus grand que 1.