# TP 7 - Programmation sous environnement Maple

## Syntaxe if, boucles for et while

Voici la syntaxe pour écrire une instruction conditionnelle :

**if** *condition1* **then** *instruction1*
**elif** *condition2* **then** *instruction2*
**else** *instruction3*
**fi**;

Pour les boucles **for** et **while**, voici la syntaxe :

**for** i **from** a **to** b **do**
*instructions*
**od**;

**while** *condition* **do**
*instructions*
**od**;

## Les procédures et fonctions

Maple dispose de deux moyens d'associer un résultat à des paramètres : les procédures et l'opérateur flèche (fonction, cf TP5). Voici la syntaxe pour les procédures :

$f :=$ **proc**(*paramètres*)
**local** *variables*;
*instructions*
**end**;

Les *paramètres* sont simplement les paramètres de la procédure, ils sont séparés par des virgules lorsqu'ils sont plusieurs. Les *variables* sont des variables internes à la procédure, distinctes des paramètres, qu'il faut déclarer après la commande **local**. Ce sont par exemple les variables de boucles, compteurs, variables auxiliaires que vous utilisez pour calculer le résultat mais qui ne font pas partie des paramètres.
Lors de l'appel de la procédure, i.e. $f(p_1, \ldots, p_n)$, Maple commence par évaluer tous les paramètres $p_1, \ldots, p_n$, puis il exécute les instructions avec ces valeurs et renvoie en fait la dernière valeur déposée sur la pile d'évaluation interne de la procédure, à moins que l'exécution n'ait exigé de renvoyer une expression à l'aide de la fonction **return**.
Si on ne sait pas a priori sur combien de paramètres comporte la procédure, le plus simple est de déclarer une séquence vide de paramètres, puis d'utiliser dans le corps de la procédure les fonctions **nargs** (qui renvoie le nombre de paramètres) et **args** (qui renvoie la suite des paramètres).

**Exercice 1.** Écrire une procédure qui renvoie l'élément maximal parmi un nombre quelconque de paramètres.

**Exercice 2.** Écrire une procédure **divise**$(a, b)$ qui effectue la division euclidienne de $a$ par $b$ et renvoie le quotient et le reste de cette division. À l'aide de cette procédure, écrire une procédure **modulo**$(a, b)$ qui calcule $a$ modulo $b$.

**Exercice 3.** Un tableau est une structure de données qui permet de repérer ses éléments par des indices entiers. Ainsi pour un tableau $T$ à une dimension, $T[i]$ désigne le $i$-ème élément du tableau. Dans cet exercice, nous travaillerons uniquement sur des tableaux à une dimension de taille $n$. Pour cela, commencer par importer la bibliothèque *ArrayTools* :

```
> with(ArrayTools)
[AddAlongDimension, Alias, AllNonZero, AnyNonZeros, Append, BlockCopy, CircularShift, ComplexAsFloat, Compress, Concatenate, Copy, DataTranspose, Diagonal,
  Dimensions, ElementDivide, ElementMultiply, ElementPower, Extend, Fill, FlipDimension, GeneralInnerProduct, GeneralOuterProduct, HasNonZero, HasZero, Insert,
  IsEqual, IsMonotonic, IsZero, Lookup, LowerTriangle, MultiplyAlongDimension, NumElems, Partition, Permute, PermuteInverse, RandomArray, ReduceAlongDimension,
  RegularArray, Remove, RemoveSingletonDimensions, Replicate, Reshape, Reverse, ScanAlongDimension, SearchArray, Size, SuggestedDatatype, SuggestedOrder,
  SuggestedSubtype, Uncompress, UpperTriangle]
```

Pour initialiser un tableau de longueur $n$ avec que des zéros, on utilise la commande $Array(1..n)$.

```
> Array(1..10)
                    [0 0 0 0 0 0 0 0 0 0]
```

Écrire quatre procédures qui effectuent les instructions suivantes :

- Initialiser un tableau aléatoirement avec des valeurs entre 0 et 100 (on utilisera la fonction *rand*).
- Calculer la moyenne d'un tableau.
- Rechercher l'élément maximal du tableau.
- Effectuer la somme de deux tableaux (supposés de même taille) dans un troisième.

**Exercice 4.** Écrire une procédure qui effectue le tri à bulle d'un tableau, et qui affiche le tableau à chaque étape. On rappelle le pseudo-code de ce tri ci-dessous.

```
tri_à_bulles(Tableau T)
   pour i allant de (taille de T)-1 à 1
       pour j allant de 0 à i-1
           si T[j+1] < T[j]
               (T[j+1], T[j]) ← (T[j], T[j+1])
```