# ALGORITHMIQUE ET PROGRAMMATION

## PARTIE A : INTRODUCTION A L'ALGORITHMIQUE

### 1) INTRODUCTION

L'algorithmique est un terme d'origine arabe, comme algèbre.
*Un algorithme, c'est une suite d'instructions, qui une fois exécutée correctement, conduit à un résultat donné.* Si l'algorithme est juste, le résultat est le résultat voulu. Si l'algorithme est faux, le résultat est, disons, aléatoire.
La maîtrise de l'algorithmique requiert deux qualités complémentaires :

- il faut avoir une certaine **intuition**, car aucune recette ne permet de savoir a priori quelles instructions permettront d'obtenir le résultat voulu. C'est là qu'intervient la forme « d'intelligence » requise pour l'algorithmique. Les réflexes, cela s'acquiert. Et ce qu'on appelle l'intuition n'est finalement que de l'expérience tellement répétée que le raisonnement, au départ laborieux, finit par devenir « spontané ».
- il faut être **méthodique** et **rigoureux**. En effet, chaque fois qu'on écrit une série d'instructions qu'on croit justes, il faut **systématiquement** se mettre mentalement à la place de la machine qui va les exécuter, *armé d'un papier et d'un crayon*, afin de vérifier si le résultat obtenu est bien celui que l'on voulait.

### 2) ALGORITHMIQUE ET PROGRAMMATION

Pourquoi apprendre l'algorithmique pour apprendre à programmer ? Pourquoi a-t-on besoin d'un langage spécial, distinct des langages de programmation compréhensibles par les ordinateurs ?

Parce que l'algorithmique exprime les instructions résolvant un problème donné **indépendamment des particularités de tel ou tel langage.** Apprendre l'algorithmique, c'est apprendre à manier la **structure logique** d'un programme informatique. Cette dimension est présente quelle que soit le langage de programmation ; mais lorsqu'on programme dans un langage (en Maple, en C, en Visual Basic, etc.) on doit en plus se colleter les problèmes de syntaxe, ou de types d'instructions, propres à ce langage. Apprendre l'algorithmique de manière séparée, c'est donc sérier les difficultés pour mieux les vaincre.

## Partie B : LES VARIABLES

### 1. A QUOI SERVENT LES VARIABLES ?

Dans un programme informatique, on va avoir en permanence besoin de stocker provisoirement des valeurs. Il peut s'agir de données fournies par l'utilisateur (frappées au clavier), ou Il peut s'agir de résultats obtenus par le programme, intermédiaires ou définitifs... on utilise donc une **variable**.

### 2. DECLARATION DES VARIABLES

La première chose à faire avant de pouvoir utiliser une variable est sa **déclaration**. Ceci se fait tout au début de l'algorithme, avant même les instructions.

Le **nom** de la variable peut comporter des lettres et des chiffres. Un nom de variable correct commence également impérativement par une lettre.

### 3. L'INSTRUCTION D'AFFECTATION

#### 3.1 Syntaxe et signification

La seule chose qu'on puisse faire avec une variable, c'est **l'affecter,** c'est-à-dire **lui attribuer une valeur**.

En pseudo-code, l'instruction d'affectation se note avec le signe ←

Ainsi :

C ← 24

Attribue la valeur 24 à la variable C.

Ceci, soit dit en passant, sous-entend impérativement que C soit une variable de type numérique.

On peut en revanche sans aucun problème attribuer à une variable la valeur d'une autre variable, telle quelle ou modifiée. Par exemple :

CC ← C

Signifie que la valeur de CC est maintenant celle de C.

Notez bien que cette instruction n'a en rien modifié la valeur de C : *une instruction d'affectation ne modifie que ce qui est situé à gauche de la flèche.*

CC ← C + 4

Si C contenait 12, CC vaut maintenant 16. De même que précédemment, C vaut toujours 12.

C ← C + 1

Si C valait 6, il vaut maintenant 7. La valeur de C est modifiée, puisque C est la variable située à gauche de la flèche.

## 3.2 Ordre des instructions

Il va de soi que l'ordre dans lequel les instructions sont écrites va jouer un rôle essentiel dans le résultat final. Considérons les deux algorithmes suivants :

Exemple 1

**Variable** A **en Numérique**
**Début**
A ← 34
A ← 12
**Fin**

Exemple 2

**Variable** A **en Numérique**
**Début**
A ← 12
A ← 34
**Fin**

Il est clair que dans le premier cas la valeur finale de A est 12, dans l'autre elle est 34 .

Tous les éléments sont maintenant en votre possession pour que ce soit à vous de jouer !

## 4. EXPRESSIONS ET OPERATEURS

Si on fait le point, on s'aperçoit que dans une instruction d'affectation, on trouve :

- à gauche de la flèche, un nom de variable, et uniquement cela. Si on voit à gauche d'une flèche d'affectation autre chose qu'un nom de variable, on peut être certain à 100% qu'il s'agit d'une erreur.
- à droite de la flèche, ce qu'on appelle une **expression**. En informatique, le terme d'**expression** ne désigne qu'une seule chose très précise :

*Une expression est un ensemble de valeurs, reliées par des opérateurs, et équivalent à une seule valeur*

On va maintenant détailler ce que l'on entend par le terme d' **opérateur**.

*Un opérateur est un signe qui relie deux valeurs, pour produire un résultat.*

Les opérateurs possibles dépendent du type des valeurs qui sont en jeu.

## 4.1 Opérateurs numériques :

Ce sont les quatre opérations arithmétiques tout ce qu'il y a de classique.

+ : addition

- : soustraction

* : multiplication

/ : division

Mentionnons également le ^ qui signifie « puissance ». 45 au carré s'écrira donc 45 ^ 2.

Enfin, on a le droit d'utiliser les parenthèses, avec les mêmes règles qu'en mathématiques. La multiplication et la division ont « naturellement » priorité sur l'addition et la soustraction. Les parenthèses ne sont ainsi utiles que pour modifier cette priorité naturelle.

Cela signifie qu'en informatique, 12 * 3 + 5 et (12 * 3) + 5 valent strictement la même chose, à savoir 41. En revanche, 12 * (3 + 5) vaut 12 * 8 soit 96.

### 4.2 Opérateur alphanumérique : &

Cet opérateur permet de *concaténer*, autrement dit d'agglomérer, deux chaînes de caractères. Par exemple :

**Variables** A, B, C en Caractère
**Début**
A ← "mpsi"
B ← "trois"
C ← A & B
**Fin**

La valeur de C à la fin de l'algorithme est "mpsitrois"

### 4.3 Opérateurs logiques (ou booléens) :

Il s'agit du ET, du OU, du NON . Nous les laisserons provisoirement de côté.

## EXERCICES ELEMENTAIRES :

**Exercice 1 :**

Quelles seront les valeurs des variables A et B après exécution des instructions suivantes ?

```
Variables A, B en Entier
Début
A ← 5
B ← A + 4
A ← A + 1
B ← A - 4
Fin
```

**Exercice 2 :**

Quelles seront les valeurs des variables A et B après exécution des instructions suivantes ?

```
Variables A, B en Entier
Début
A ← 5
B ← 2
A ← B
B ← A
Fin
```

**Exercice 3 : « Déjà fait avec MAPLE »**

Ecrire un algorithme permettant d'échanger les valeurs de deux variables A et B, et ce quel que soit leur contenu préalable.
**N.B :** Rappelons l'importance de l'introduction d'une variable intermédiaire.

**Exercice 4 :**

On dispose de trois variables A, B et C. Ecrivez un algorithme transférant à B la valeur de A, à C la valeur de B et à A la valeur de C , et ce quels que soient les contenus préalables de ces variables.

**Exercice 5 :**

Considérons l'algorithme suivant :

```
Variables A, B, C en Caractères
Début
A ← "423"
B ← "12"
C ← A & B
Fin
Quel est alors le contenu la variable C ?
```

# PARTIE C : LECTURE ET ECRITURE

## 1. DE QUOI PARLE-T-ON ?

Il existe des d'instructions pour permettre à la machine de dialoguer avec l'utilisateur .

Dans un sens, ces instructions permettent à l'utilisateur de rentrer des valeurs au clavier pour qu'elles soient utilisées par le programme. Cette opération est la **lecture**.

Dans l'autre sens, d'autres instructions permettent au programme de communiquer des valeurs à l'utilisateur en les affichant à l'écran. Cette opération est **l'écriture**.

## 2. LES INSTRUCTIONS DE LECTURE ET D'ECRITURE

Pour que l'utilisateur entre la valeur de C , on mettra :

```
Lire C
```

**Dès que le programme rencontre une instruction Lire, l'exécution s'interrompt, attendant la frappe d'une valeur au clavier.**

Dans le sens inverse, pour écrire quelque chose à l'écran, c'est aussi simple que :

```
Ecrire cc
```

Exemple :

```
Ecrire "Entrez votre classe : "
Lire mpsi3
```

## EXERCICES ELEMENTAIRES

**Exercice 1:**

Quel résultat produit le programme suivant ?

```
Variables val, double : numériques
Début
Val ← 25
Double ← Val * 2
Ecrire  Val
Ecrire  Double
Fin
```

**Corrigé :** On verra apparaître à l'écran 25, puis 50

**Exercice 2 :**

Ecrire un programme qui demande un nombre à l'utilisateur, puis qui calcule et affiche le carré de ce nombre.

**Corrigé :**

```
Variables nb, car en Entier
Début
Ecrire "Entrez un nombre :"
Lire nb
car ← nb * nb
Ecrire "Son carré est : ", car
Fin
```

# PARTIE C : LES TESTS

## STRUCTURE D'UN TEST

Il n'y a que **deux formes possibles** pour un test ; la première est la plus simple, la seconde la plus complexe.

## 1. Tests simples :

```
Si condition Alors
  Instructions
Finsi
```

Exemple : Pour la température de l'eau.

```
Variable Temp en Entier
Début
Ecrire "Entrez la température de l'eau :"
Lire Temp
Si Temp =< 0 Alors
  Ecrire "C'est de la glace"
FinSi
Fin
```

Ou :

```
Si condition Alors
  Instructions 1
Sinon
  Instructions 2
Finsi
```

Exemple : encore pour la température de l'eau.

```
Variable Temp en Entier
Début
Ecrire "Entrez la température de l'eau :"
Lire Temp
Si Temp =< 0 Alors
  Ecrire "C'est de la glace"
SINON
  Ecrire "C'est liquide ou gazeux"
Finsi
Fin
```

## 2. TESTS IMBRIQUES

EXEMPLE :

```
Variable Temp en Entier
Début
Ecrire "Entrez la température de l'eau :"
Lire Temp
Si Temp =< 0 Alors
  Ecrire "C'est de la glace"
FinSi
Si Temp > 0 Et Temp < 100 Alors
  Ecrire "C'est du liquide"
Finsi
Si Temp > 100 Alors
  Ecrire "C'est de la vapeur"
Finsi
Fin
```

Vous constaterez que c'est un peu trop détaillé. Les conditions se ressemblent plus ou moins, et surtout on oblige la machine à examiner trois tests successifs alors que tous portent sur une même chose, la température de l'eau (la valeur de la variable Temp). Il serait ainsi bien plus rationnel d'**imbriquer** les tests de cette manière :

```
Variable Temp en Entier
Début
Ecrire "Entrez la température de l'eau :"
Lire Temp
Si Temp =< 0 Alors
  Ecrire "C'est de la glace"
Sinon
  Si Temp < 100 Alors
    Ecrire "C'est du liquide"
  Sinon
    Ecrire "C'est de la vapeur"
  Finsi
Finsi
Fin
```

**Dans le cas de tests imbriqués, le Sinon et le Si peuvent être fusionnés en un SinonSi. On considère alors qu'il s'agit d'un seul bloc de test, conclu par un seul FinSi.**

Exemple : ( c'est le plus commode à utiliser )

```
Variable Temp en Entier
Début
Ecrire "Entrez la température de l'eau :"
Lire Temp
Si Temp =< 0 Alors

  Ecrire "C'est de la glace"
SinonSi Temp < 100 Alors
  Ecrire "C'est du liquide"
Sinon
  Ecrire "C'est de la vapeur"
Finsi
Fin
```

Exercice :

Ecrire un algorithme qui demande l'âge d'un enfant à l'utilisateur. Ensuite, il l'informe de sa catégorie :

- "**Poussin**" de 6 à 8 ans (6 et 8 compris)
- "**Pupille**" de 8 à 10 ans (8 non compris et 10 compris)
- "**Minime**" de 10 à 12 ans (10 non compris et 12 compris)
- "**Cadet**" après 12 ans (12 non compris)

# PARTIE D : LES BOUCLES

## 1. La boucle « pour »

Nous donnons la formulation d'une structure « **Pour** », sa syntaxe générale est :

```
Pour Compteur ← ValInit à ValFin ( par Pas) faire

Instructions

Finpour
```

Il s'agit de répéter les instructions pour compteur allant de `ValInit à ValFin.`

Les structures **Pour** sont employées dans les situations où l'on doit procéder à un traitement systématique sur les éléments d'un ensemble dont le programmeur connaît d'avance la quantité.

Exemple : pour afficher mpsi3 5 fois :

```
Pour i ← 1 à 5 faire
afficher mpsi3
Finpour
```

## 2. La boucle « tant que »

Pour la formulation d'une structure « **tant que** », sa syntaxe générale est :

```
Tant que condition faire
 Instructions
ftq
```

Il s'agit de répéter les instructions tant que la condition est satisfaite, quand elle n'est plus remplie, les instructions ne s'exécutent plus.

*Exemple :* Considérons l'exemple suivant :

```
n←0:
Tant que  n² ≤ 8  faire
     n←n+1
ftq
écrire n
```

Question : Que vaut la valeur de n après son affichage ?

Remarque 1: Les structures **TantQue** sont employées dans les situations où l'on doit procéder à un traitement systématique sur les éléments d'un ensemble dont on ne connaît pas d'avance la quantité, comme par exemple :

Remarque 1: Si la condition est toujours vraie, alors le programme ne s'arrêtera jamais ; exemple :

```
C ← 1
Tant que C > 0 faire
  C ← C+1
ftq
```

## EXEMPLES DIVERS

« IL EST RECOMMANDE DE REFLICHIR AVANT DE SONSULTER LA REPONSE ! »

**Exemple 1:** Ecrire un algorithme qui demande un nombre au départ, et qui ensuite affiche les onze nombres suivants.

**Réponse :**

```
Variables N, i en Entier
Début
Ecrire "Entrez un nombre : "
Lire N
Ecrire "Les 11 nombres suivants sont : "
Pour i ← N + 1 à N + 11  faire
    Ecrire i
fin pour
Fin
```

**Exemple 2:** Ecrire un algorithme qui demande un nombre de départ, et qui ensuite écrit la table de multiplication de ce nombre, présentée comme suit : « cas où l'utilisateur entre le nombre 3 »

```
La table de multiplication de ce nombre est :
3 x 1 = 3
3 x 2 = 6
3 x 3 = 9
  ……
3 x 10 = 30
```

**Réponse :**

```
Variables N, i en Entier
Debut
Ecrire "Entrez un nombre : "
Lire N
Ecrire "La table de multiplication de ce nombre est : "
Pour i ← 1 à 10
  Ecrire N, " x ", i, " = ", N*i
fin pour
Fin
```

**Exemple 3 : « Déjà fait avec MAPLE »** Ecrire un algorithme qui demande un nombre de départ, et qui calcule la somme des entiers jusqu'à ce nombre. Par exemple, si l'on entre 5, le programme doit calculer :

1 + 2 + 3 + 4 + 5 = 15

NB : on souhaite afficher uniquement le résultat, pas la décomposition du calcul.

**Réponse :**

```
Variables N, i, Som en Entier
Début
Ecrire "Entrez un nombre : "
Lire N
Som ← 0
Pour i ← 1 à N  faire
  Som ← Som + i
Fin pour
Ecrire  "La somme est : ", Som
Fin
```

**Exemple 4 : « Déjà fait avec MAPLE »** Ecrire un algorithme qui demande un nombre de départ, et qui calcule sa factorielle.

**Réponse :**

```
Variables N, i, F en Entier
Debut
Ecrire "Entrez un nombre : "
Lire N
F ← 1
Pour i ← 2 à N
  F ← F * i
fin pour
Ecrire "La factorielle est : ", F
Fin
```

Remarque : Noter que cet algorithme couvre les cas : N=0 et N=1 !!!

# EXERCICES

Exercice 1 : **« Déjà fait avec MAPLE »** Ecrire un algorithme demandant deux entiers naturels non nuls m et n, et permettant de calculer et afficher le produit $m \times n$ en utilisant juste l'addition.

Exercice 2 : **« Déjà fait avec MAPLE »**

1) Ecrire un algorithme qui demande deux nombres et qui calcule et affiche leur maximum.
2) Ecrire un algorithme qui demande deux nombres et qui calcule et affiche leur minimum.

Exercice 3 : **« Déjà fait avec MAPLE »**
Ecrire un algorithme qui demande un nombre et calcule et affiche sa valeur absolue.

Exercice 4 : **« Déjà fait avec MAPLE »**

Ecrire un algorithme qui demande un nombre réel x et un entier naturel n et qui calcule et affiche la puissance $x^n$.

Exercice 5 : **« Déjà fait avec MAPLE »** Ecrire un algorithme qui demande un réel x et un entier « assez grand » n, et qui calcule et affiche une valeur approchée de $e^x$ grâce au fameux résultat d'analyse : $\lim_{n\to+\infty}\left(1+x+\frac{x^2}{2!}+...+\frac{x^n}{n!}\right)=e^x$.

Exercice 6 : **« Déjà fait avec MAPLE »** Soit la suite récurrente $(u_n)_{n\in\mathbb{N}}$ définie par $\begin{cases}u_0=1\\ u_{n+1}=\sin(u_n)\ \ \forall n\in\mathbb{N}\end{cases}$

Ecrire un algorithme qui calcule une valeur approchée de $u_{50}$. Modifier-le pour un entier n quelconque.

Exercice 7 : **« Déjà fait avec MAPLE »** Considérons la suite $u_n=n+e^n$, on sait que $\lim_{n\to+\infty}u_n=+\infty$.

Alors pour $A=10^9\succ 0$, $\exists N\in\mathbb{N},\forall n\geq N,\ u_n\succ 10^9$ ( c'est la définition de $\lim_{n\to+\infty}u_n=+\infty$ )

Ecrire un algorithme qui détermine le $1^{er}$ entier n vérifiant $u_n\succ 10^9$.

Exercice 8 : **« Déjà fait avec MAPLE »** Considérons la suite $u_n=\frac{n-1}{2n+3}$, on sait qu'elle converge vers $\frac{1}{2}$.

Alors pour $\varepsilon=0.000045\succ 0$, $\exists N\in\mathbb{N},\forall n\geq N,|u_n-\frac{1}{2}|\prec 0.000045$ (Définition de $\lim_{n\to+\infty}u_n=\frac{1}{2}$).

Ecrire un algorithme qui détermine le $1^{er}$ entier n vérifiant $|u_n-\frac{1}{2}|\prec 0.000045$.

Exercice 9 : **« Déjà fait avec MAPLE »** Soit la suite récurrente $(u_n)_{n\in\mathbb{N}}$ définie par $\begin{cases}u_0=1\\ u_{n+1}=\sin(u_n)\ \ \forall n\in\mathbb{N}\end{cases}$. On peut montrer qu'elle converge vers zéro « exercice classique d'analyse ».

Ecrire un algorithme qui détermine alors le $1^{er}$ indice n tel que $|u_n|\prec 10^{-1}$.

## AUTRES EXEMPLES

**Exemple 1:** Ecrire un algorithme qui demande à l'utilisateur un nombre réel compris entre 10 et 30 jusqu'à ce que la réponse convienne ; dans ce cas, il lui indique que son nombre convient.

Réponse :

```
Variable x en Entier
Début
Ecrire "Entrez un nombre entre 10 et 30"
lire x
Tant que x < 10 ou x > 30 faire
      Ecrire " Entrez un nombre entre 10 et 30"
      lire x
ftq
Ecrire "ton nombre convient maintenant"
Fin
```

**Exemple 2:** **« variante de l'exemple 1 »** Ecrire un algorithme qui demande un nombre compris entre 10 et 30 , jusqu'à ce que la réponse convienne. En cas de réponse supérieure à 30, on fera apparaître le message : « Plus petit que ça ! », en cas de réponse inférieure à 10, « Plus grand que ça ! » , et quand la réponse convient « ton nombre convient maintenant ».

Réponse :

```
Variable x en Entier
Début
Ecrire "Entrez un nombre entre 10 et 30"
lire x
Tant que x < 10 ou x > 30 faire
      si x < 10 alors Ecrire " plus grand que ça !"
        sinonsi  x > 30 alors Ecrire " plus petit que ça !"
      Finsi
      Ecrire " Entrez un nombre entre 10 et 30"
      lire x
Ftq
Ecrire "ton nombre convient maintenant"
Fin
```

# SOLUTIONS DES EXERCICES

Solution de l'exo 1 :

**Variables** m,n,x,i **: entiers**
**Début**
**Ecrire** ″entrer deux entiers non nuls ″
**Lire** m,n
x ← 0
**pour** i ← 1 à n **faire**
x ← x+m
**finpour**
**écrire** ″leur produit est : ″ , x
**fin**

Solution de l'exo 2 : pour 1°) ; 2°) est analogue.

**Variables** a,b,x **: numérique**
**Début**
**Ecrire** ″entrer deux nombres ″
**Lire** a,b
**si** a>b **alors** x ← a **sinon** x ← b
**finsi**
**écrire** ″leur max est : ″ , x
**fin**

Solution de l'exo 3 :

**Variables** a,x **: numérique**
**Début**
**Ecrire** ″entrer un nombre ″
**Lire** a
**si** a ≥ 0 **alors** x ← a **sinon** x ← - a
**finsi**
**écrire** ″sa valeur absolue est : ″ , x
**fin**

Solution de l'exo 4 :

**Variables** i, n **: entier**
**c,**x : **numérique**
**Début**
**Ecrire** ″entrer un nombre réel ″
**Lire** x
**Ecrire** ″entrer un entier naturel ″
**Lire** n
c ← 1
**pour** i ← 1 à n **faire**
c ← c * x
**finpour**
**écrire** ″la puissance $x^n$ est : ″ , c
**fin**

Solution de l'exo 5 :

Solution de l'exo 6 :

**Variables** n , i **: entier**
x , c : **numérique**
**Début**
**Ecrire** ″entrer un nombre réel ″
**Lire** x
**Ecrire** ″entrer un entier naturel assez grand ″
**Lire** n
c ← 1
**pour** i ← 1 à n **faire**
c ← c + $\frac{x^i}{i!}$
**finpour**
**écrire** ″une valeur approchée de $e^x$ est : ″ , c
**fin**

**Variables** i **: entiers**
c : **numérique**
**Début**
c ← 1
**pour** i ← 1 à 50 **faire**
c ← sin (c)
**finpour**
**écrire** ″une valeur approchée de $u_{50}$ est : ″, c
**fin**

*Pour l'btention d'une valeur approchée de $u_n$ pour n quelconque cette fois-ci :*
**Variables** n ,i **: entiers**
c : **numérique**
**Début**
**Ecrire** ″entrer un entier n ″
**Lire** n
c ← 1
**pour** i ← 1 à n **faire**
c ← sin (c)
**finpour**
**écrire** ″une valeur approchée de $u_n$ est : ″ , c
**fin**

Solution de l'exo 7 :
**Variables** n **: entiers**
u : **numérique**
**Début**
n ← 0
u ← $n + e^n$
**tant que** $u \le 10^9$ **faire**
n ← n+1
u ← $n + e^n$
**ftq**
**écrire** ″ le $1^{er}$ entier n vérifiant $u_n \succ 10^9$ .est : ″ , n
**fin**

Solution de l'exo 8 :
**Variables** n **: entiers**
u : **numérique**
**Début**
n ← 0
u ← $\frac{n-1}{2n+3}$
**tant que** $|u - \frac{1}{2}| \ge 0.000045$ **faire**
n ← n+1
u ← $\frac{n-1}{2n+3}$
**ftq**
**écrire** ″ le $1^{er}$ entier n vérifiant $|u - \frac{1}{2}| \prec 0.000045$ .est : ″ , n
**fin**

Solution de l'exo 9 :
**Variables** n **: entiers**
u : **numérique**
**Début**
n ← 0
u ← 1
**tant que** $|u| \ge 10^{-1}$ **faire**
n ← n+1
u ← sin(u )
**ftq**
**écrire** ″ le $1^{er}$ entier n vérifiant $|u_n| \prec 10^{-1}$ est : ″ , n
**fin**

# LES TABLEAUX

## 1. LES TABLEAUX A UNE DIMENSION :

Imaginons que dans un programme, nous ayons besoin simultanément de 12 valeurs ; par exemple, des notes pour calculer leur moyenne. On a :

```
Moy ← (N0+N1+N2+N3+N4+N5+N6+N7+N8+N9+N10+N11)/12
```

Un tableau doit être déclaré tout en précisant le nombre et le type de valeurs qu'il contiendra.

Lors de la déclaration d'un tableau, on précise la plus grande valeur de l'indice (dans notre exemple c'est 11 pas 12) !

N.B : *Les indices des tableaux commencent généralement à 0, et non à 1 ; c'est le cas en langage C et en Visual Basic, par contre en Maple, un tableau commence par l'indice 1.*

*Déclaration du tableau pour notre exemple ci-dessus :*

```
Tableau Note(11) en Entier          # Déclaration du tableau ; son nom est : « Note »
```

Ainsi, une note **Ni** est **Note(i)** et que :

$$Moy \leftarrow \left(\sum_{i=0}^{11} Note(i)\right)/12$$

L'énorme avantage des tableaux, c'est qu'on va pouvoir les traiter en faisant des boucles. Par exemple, pour effectuer notre calcul de moyenne, voici un algorithme :

```
Tableau Note(11) en Numérique          # déclaration du tableau
Variables Moy, Som en Numérique        # déclaration des variables Moy et Som
Début
Pour i ← 0 à 11 faire
  Ecrire "Entrez la note n°", i
  Lire Note(i)
finpour
Som ← 0
Pour i ← 0 à 11 faire
   Som ← Som + Note(i)
finpour
Moy ← Som / 12
Fin
```

# EXEMPLES :

---

**Exemple 1:**

Ecrivons un algorithme qui déclare et remplit un tableau de 7 valeurs numériques en leur attribuant toutes la valeur 0 .

```
Tableau tabNul(6) en Numérique
Variable i en entier
Début
Pour i ← 0 à 6  faire
   tabNul(i) ← 0
Finpour
Fin
```

---

**Exemple2 :**

1) Ecrivons un algorithme qui déclare un tableau de 9 notes, dont on fait ensuite saisir les valeurs par l'utilisateur.

*Réponse :*

```
Tableau Notes(8) en Numérique
Variable i en entier
Pour i ← 0 à 8    faire
  Ecrire "Entrez la note numéro ", i + 1
  Lire Notes(i)
Finpour
Fin
```

2) Complétons cet algorithme afin que le calcul de la moyenne des notes soit effectué et affiché à l'écran.

*Réponse :*

```
tableau Notes(8) en Numérique
Variable     i en entier
        s, Moy en numérique
Pour i ← 0 à 8    faire
  Ecrire "Entrez la note numéro ", i + 1
  Lire Notes(i)
Finpour
s ← 0
Pour i ← 0 à 8  faire
  s ← s + Notes(i)
Finpour
Moy ← s/9
Ecrire "La moyenne est:", Moy
Fin
```

# EXERCICES :

**Exercice 1 :**

1) Que produit l'algorithme suivant ?

```
Tableau Nb(5) en Entier
Variable i en Entier
Début
Pour i ← 0 à 5   faire
  Nb(i) ← i * i
Finpour
Pour i ← 0 à 5  faire
  Ecrire Nb(i)
Finpour
Fin
```

2) Simplifier cet algorithme ( avec la même boucle « pour » )

**Exercice 2 :**

Que produit l'algorithme suivant ?

```
Tableau fibonacci(7) en Entier
Variable i en Entier
Début
fibonacci(0) ← 1
fibonacci(1) ← 1
Pour i ← 2 à 7  faire
  fibonacci (i) ← fibonacci (i-1) + fibonacci (i-2)
Finpour
Pour i ← 0 à 7  faire
  Ecrire fibonacci (i)
Finpour
Fin
```

**Exercice 3 :**

Ecrivez un algorithme permettant à l'utilisateur de saisir 10 valeurs, qui devront être stockées dans un tableau. Enfin, une fois la saisie terminée, le programme affichera le nombre de valeurs négatives et le nombre de valeurs positives.

**Exercice 4 :**

Ecrivez un algorithme permettant à l'utilisateur de saisir 10 valeurs, qui devront être stockées dans un tableau. Enfin, une fois la saisie terminée, le programme recherche la plus grande valeur au sein de ce tableau, et détermine sa position dans celui-ci.

## 2. TABLEAUX A DEUX DIMENSIONS :

L'informatique nous offre la possibilité de déclarer des tableaux dans lesquels les valeurs ne sont pas repérées par une seule, mais par **deux coordonnées**.

Un tel tableau se déclare de la manière suivante :

```
Tableau exemple(7, 7) en Numérique      # « exemple » est le nom du tableau
```

Cela veut dire : réserve-moi un espace de mémoire pour 8 x 8 entiers, et quand j'aurai besoin de l'une de ces valeurs, je les repèrerai par deux **indices**. « Rappel : chaque indice commence de 0 »

### EXEMPLES :

**Exemple 1 :**

Écrivons un algorithme remplissant un tableau de 6 sur 13, avec des zéros.

*Réponse :*

```
Tableau tab(5, 12) en Entier
Debut
Pour i ← 0 à 5
  Pour j ← 0 à 12
     tab(i, j) ← 0
  Finpour
Finpour
Fin
```

**Exemple 2 :**

Considérons l'algorithme suivant :

```
Tableau X(1, 2) en Entier
Variables i, j, val  en Entier
Début
Val ← 1
Pour i ← 0 à 1
  Pour j ← 0 à 2
     X(i, j) ← Val
     Val ← Val + 1
  Finpour
Finpour
Pour i ← 0 à 1
  Pour j ← 0 à 2
     Ecrire X(i, j)
  Finpour
Finpour
Fin
```

« Remplissons en crayon les vides : »

Cet algorithme remplit un tableau de la manière suivante:

```
X(0, 0) =
X(0, 1) =
X(0, 2) =
X(1, 0) =
X(1, 1) =
X(1, 2) =
```

Cet algorithme écrit ensuite ces valeurs à l'écran, dans **cet ordre.**

### EXERCICES

**Exercice 1 : « similaire à l'exemple 2 ci-dessus »**

Quel résultat produira cet algorithme ?

```
Tableau T(3, 1) en Entier
Variables k, m, en Entier
Début
Pour k ← 0 à 3
  Pour m ← 0 à 1
    T(k, m) ← k + m
 Finpour
Finpour
Pour k ← 0 à 3
  Pour m ← 0 à 1
    Ecrire T(k, m)
 Finpour
Finpour
Fin
```

---

**Exercice 2 :**

Soit un tableau T à deux dimensions (12, 8) préalablement rempli de valeurs numériques.
Écrire un algorithme qui recherche la plus grande valeur au sein de ce tableau, et qui détermine sa position dans celui-ci.

---

# QUELQUES TECHNIQUES

Détaillons quelques techniques de programmation dont la connaissance est parfaitement indispensable .

Le but de la manœuvre est de **trier** un tableau, par exemple de 12 éléments, dans ***l'ordre croissant***.

## 1. LE TRI PAR SELECTION

On abordera ici le **tri par sélection**, dont la technique est la suivante :

On met en bonne position l'élément numéro 1, c'est-à-dire le plus petit. Puis on met en bonne position l'élément suivant. Et ainsi de suite jusqu'au dernier. Par exemple, si l'on part de :

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **45** | 122 | 12 | **3** | 21 | 78 | 64 | 53 | 89 | 28 | 84 | 46 |

On commence par rechercher, parmi les 12 valeurs, quel est le plus petit élément , et où il se trouve. On l'identifie en quatrième position (c'est le nombre **3**), et on l'échange alors avec le premier élément (le nombre 45). Le tableau devient ainsi :

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **3** | 122 | 12 | **45** | 21 | 78 | 64 | 53 | 89 | 28 | 84 | 46 |

On recommence à chercher le plus petit élément, mais cette fois, **seulement à partir du deuxième** (puisque le premier est maintenant correct, on n'y touche plus). On le trouve en troisième position (c'est le nombre **12**).

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **3** | **122** | **12** | 45 | 21 | 78 | 64 | 53 | 89 | 28 | 84 | 46 |

On échange donc le deuxième avec le troisième , on obtient :

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 3 | **12** | **122** | 45 | 21 | 78 | 64 | 53 | 89 | 28 | 84 | 46 |

On recommence à chercher le plus petit élément à partir du troisième (puisque les deux premiers sont maintenant bien placés).On le place correctement en l'échangeant avec **122**, etc.

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 3 | 12 | **21** | 45 | **122** | 78 | 64 | 53 | 89 | 28 | 84 | 46 |

Et en progressant de plus en plus, jusqu'à l'avant dernier.

**Voici ce qu'on peut écrire sur un algorithme, c'est sa boucle principale :**

Appelons ***tab*** le nom du tableau ci-dessus. Son plus grand indice est 11 (il possède 12 valeurs).

```
Pour i ← 0 à 10  faire
    ii ← i
    Pour j ← i  à 11 faire
        Si tab(j) < tab(ii) Alors
            ii ← j
        Finsi
    Finpour       # on sait maintenant où est le plus petit élément entre i et 11 ; son indice est ii
    c ← tab(ii)   # on effectue maintenant la permutation entre tab[i] et tab[ii]
    tab(ii) ← tab(i)
    tab(i) ← c    # On a placé correctement l'élément numéro i, on passe à présent au suivant.
Finpour
```

## 2. LA RECHERCHE DANS UN TABLEAU

Illustrons l'idée à travers un exemple extrêmement fréquent : *la recherche de l'occurrence d'une valeur dans un tableau*.

Considérons le tableau **Tab(11)** ci-dessus « comportant 12 valeurs » . On doit écrire un algorithme saisissant un nombre au clavier, et qui informe l'utilisateur de la présence ou de l'absence de la valeur saisie dans ce tableau.

La première étape, évidente, consiste à écrire les instructions de lecture / écriture, et la boucle de parcours du tableau :

```
Variable N en Numérique
Début
Ecrire "Entrez la valeur à rechercher "
Lire N
Pour i ← 0 à 11  faire
?????????
finpour
Fin
```

Il nous reste à combler les points d'interrogation de la boucle **Pour**. Évidemment, il va falloir comparer N à chaque élément du tableau ; s'il existe un i entre 0 et 11 tel que N=tab(i), alors N fait partie du tableau, sinon alors N n'y appartient pas.

L'idée est l'introduction d'une variable intermédiaire ***inter***, initialisée en 0 et qui prend 1 à chaque fois que N=tab(i) pour tout i=0..11

Ainsi, si après avoir balayé tous les i=0..11 ***inter*** vaut toujours 0, alors cela veut dire que N ne figure pas parmi les éléments du tableau, et si ***inter***=1, alors N fait partie du tableau.

```
Variable N en Numérique
         i en entier
Début
Ecrire "Entrez la valeur à rechercher"
Lire N
inter ← 0
Pour i ← 0 à 11  faire
   Si N = Tab(i) Alors
      inter ← 1
   FinSi
finpour
Si inter = 1  Alors
  Ecrire "cette valeur fait partie du tableau"
Sinon
  Ecrire "cette valeur ne fait pas partie du tableau"
FinSi
Fin
```

### REMARQUE : AUTRE FAÇON D'ECRIRE CET ALGORITHME :

Avant la nouvelle écriture, prenons cette « recréation » en Maple :

```
> inter:=true:
 if inter then print(`cette valeur fait partie du tableau`) fi;
> inter:=false:
 if inter then print(`cette valeur fait partie du tableau`) fi;
> inter:=true:
 if inter then print(`cette valeur fait partie du tableau`) ;
    else print(`cette valeur ne fait pas partie du tableau`);
```

```
fi;
> inter:=false:
 if inter then print(`cette valeur fait partie du tableau`) ;
    else print(`cette valeur ne fait pas partie du tableau`);
 fi;
```

*Fin recréation! Passons à notre nouvelle façon d'écrire l'algorithme.*

```
Variable N en Numérique
         i en entier
Début
Ecrire "Entrez la valeur à rechercher"
Lire N
inter ← faux
Pour i ← 0 à 11  faire
   Si N = Tab(i) Alors
      inter ← vrai
   FinSi
finpour
Si inter  Alors
  Ecrire "cette valeur fait partie du tableau"
Sinon
  Ecrire "cette valeur ne fait pas partie du tableau"
FinSi
Fin
```

## 3. TRI DE TABLEAU + RECHERHCE DANS UN TABLEAU = TRI A BULLES

L'idée de départ du tri à bulles consiste à se dire qu'un tableau trié en ordre croissant, c'est un tableau dans lequel **tout élément est plus petit que celui qui le suit**.

Prenons chaque élément d'un tableau, et comparons-le avec l'élément qui le suit. Si l'ordre n'est pas bon, on permute ces deux éléments. Et on recommence jusqu'à ce que l'on n'ait plus aucune permutation à effectuer. *Les éléments les plus petits remontent ainsi peu à peu vers les premières places, ce qui explique sa dénomination de « tri à bulle ».*

A l'intérieur de la boucle, on prend alors les éléments du tableau, du premier jusqu'à l'avant-dernier, et on procède à un échange si nécessaire :

```
Pour i ← 0 à 10            # on rappelle que notre tableau contient 12 valeurs, donc i varie entre 0 et 11
  Si t(i) > t(i+1) Alors
    temp ← t(i)             # ici on permute t(i) et t(i+1)
    t(i) ← t(i+1)
    t(i+1) ← temp
  Finsi
Finpour
```

On introduit une variable, appelée **test** par exemple, qui prend la valeur **Vrai** dès qu'une permutation a été faite lors du balayage (il suffit qu'il y en ait eu une seule pour qu'on doive tout recommencer encore).

Il faut la remettre à **Faux** à chaque tour de la boucle principale. Comme ça, si après le balayage la variable **test** contient toujours la valeur **Faux**, alors cela veut dire qu'aucune permutation n'a été faite, ce qui implique que le tri est accompli.

Un dernier point à mentionner ; il ne faut pas oublier de lancer la boucle principale, et pour cela de donner la valeur **Vrai** à la variable **test** au tout départ de l'algorithme. Voici en fin l'algorithme au complet :

```
Variable   test en Booléen ;  i en entier ;   temp en numérique
Début
test ← Vrai
TantQue test  faire
  test ← Faux
  Pour i ← 0 à 10 faire
    Si t(i) > t(i+1) alors
      temp ← t(i)
      t(i) ← t(i+1)
      t(i+1) ← temp
      test ← Vrai
    Finsi
  Finpour
FinTantQue
Fin
```

# QUELQUES EXEMPLES ET ALGORITHMES CLASSIQUES :

## 1) Algorithme d'Euclide :

Rappelons qu'il consiste à chercher le pgcd de deux entiers en effectuant une succession de divisions euclidiennes. Le pgcd cherché est alors le dernier reste non nul . Voici l'algorithme correspondant :

**Variables** a, b , r en entiers
**Début**
**Ecrire**(" entrer deux entiers non nuls ")
**Lire** (a, b)
r ← a mod b
**tantque** r ≠ 0 **faire**
    a ← b
    b ← r
    r ← a mod b
**Fintantque**
**écrire** (" leur pgcd est:", b)
**Fin**

Traduction en **C++** :

```
#include<stdio.h>
#include<stdlib.h>
main()
{
    int a,b,r;
    printf("entrer deux entiers non nuls\n");
    scanf("%d%d",&a,&b);
    r=a%b;
    while(r!=0)
    {
            a=b;
            b=r;
            r=a%b;
    }
    printf("leur pgcd est: %d\n",b);
    system("pause");
    return(0);
}
```

## 2) Résolution d'une équation numérique « f(x)=0 » : Méthode de **dichotomie**.

Soit f une fonction définie et **continue** sur un intervalle [a,b] telle que f(a) et f(b) soient de signes **contraires.** Le TVI (thm des val intermidiaires) assure l'existence d'une solution de l'équation f(x)=0 . Supposons que celle-ci est unique ; notons-la $\alpha$ ( ce qui est rempli si f est par exemple strictement monotone sur [a,b]).

Description de la méthode de dichotomie :

*On prend le milieu c de [a,b], si f(a) et f(c) sont de signes contraires, alors* $\alpha$ *appartient à [a , c], sinon,* $\alpha$ *appartient à [c , b]. Notons [a1 , b1] le nouveau intervalle contenant* $\alpha$ *.Remarquons que sa longueur est* $\frac{b-a}{2}$ *.*

*On reprend la même chose avec l'intervalle [a1 , b1], on obtient un nouveau intervalle [a2,b2] contenant* $\alpha$ *de longueur* $\frac{b-a}{2^2}$ *.*

*De proche en proche, on construit un intervalle[an , bn], de longueur* $\frac{b-a}{2^n}$ *et contenant* $\alpha$ *. On a* $\lim_{x\to\infty}\left(\frac{b-a}{2^n}\right)=0$ *,*

*alors, pour un entier n assez grand, on peut considérer a (ou b) comme une approximation de* $\alpha$ *.*
*Dans l'algorithme, on a besoin d'un test d'arrêt : Pour une précision epsilon donnée, le programme doit s'arrêter dès que* $|a-b| \prec epsilon$ . L'erreur en fait est inférieure à |a-b| *puisque* $\alpha$ *appartient à [a , b]* « voir algorithme » :

**Variable** precision,a,b,c,err **en numérique**
**Ecrire** (entrer les deux bornes a et b)
**Lire** (a,b)
**Ecrire** (entrer une precision)
**Lire** (precision)

err ← |b-a|
**tantque** err>precision **faire**
**c** ← (a+b)/2
**si** f(a).f(c) ≤0 **alors** **b** ← c **sinon** a ← c
**finsi**
err ← |b-a|
**fintanque**
**ecrire** (une valeur approchée à precision prés de la solution est , a )
**fin**

**Traduction en C++ :**

Exemple 1 : une valeur approchée de $\sqrt{2}$ .

```
#include<math.h>
float f(float x)
{
     float c;
     c=pow(x,2)-2; // on cherche une val approchée de racinecarré(2)
     return c;
     }
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
main()
{
    float precision,a,b,c,err;
    printf("entrer les deux bornes a et b\n");
    scanf("%f%f",&a,&b);
    printf("entrer precision\n");
    scanf("%f",&precision);
    err=b-a;
    while(err>precision)
    {
            c=(a+b)/2;
            if(f(a)*f(c)<=0) b=c; else a=c;
            err=b-a;
    }
    printf("une valeur approchee de la solution cherchee a %f pres est: %f\n",precision,a);
    system("pause");
    return(0);
 }
```

Exemple 2: une valeur approchée de la solution de :sin(x)=ln(x) dans l'intervalle [1,3] .

```
#include<math.h>
float f(float x)
{
     float c;
     c=sin(x)-log(x); // on cherche une val approchée de racinecarré(2)
     return c;
 }
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
main()
{
    float precision,a,b,c,err;
    printf("entrer les deux bornes a et b\n");
    scanf("%f%f",&a,&b);
    printf("entrer precision\n");
    scanf("%f",&precision);
    err=b-a;
    while(err>precision)
    {
            c=(a+b)/2;
            if(f(a)*f(c)<=0) b=c; else a=c;
            err=b-a;
    }
    printf("une valeur approchee de la solution cherchee a %f pres est: %f\n",precision,a);
    system("pause");
```

```
    return(0);
}
```

## 3) Méthode du pivot de Gauss pour la résolution d'un système de Cramer :

Pour la mise en œuvre de la méthode, voir le cahier de cours ; chapitre sur les systèmes linaires.

*Voici un algorithme permettant la résolution d'un système de Cramer :*

```
Variable i, j, n en entiers
         c, s en numérique
tableau  T(n-1,n)  en numérique     // tableau composé des coeff du syst ainsi que le second membre du système
tableau  x(n-1) en numérique        // tableau composé des coeff du vecteur inconnu
Début
Ecrire (entrer la taille du syst de Cramer à résoudre)
Lire n
Pour i ← 0 à n-1 faire       // la saisie de la matrice du systeme
      Pour j ← 0 à n-1 faire
            Ecrire (entrer le (i,j) ème coefficient du système)
            Lire T(i,j)
     Finpour
Finpour
Pour i ← 0 à n-1 faire       // la saisie du second membre
      Ecrire (entrer le i ème coefficient du second membre du système)
      Lire T(i,n)
Finpour
Pour i ← 0 à n-1 faire
      j ← i                          // repérer j tel que T(j,i)  ≠ 0
      tantque T(j,i)=0  faire
            j ← j+1
      fintantque
      pour k ← i  à n  faire      // permutation des lignes L_i et L_j ; notons qu'avant i, tout est nul
            c ← T(i,k)
            T(i,k)  ← T(j,k)
            T(j,k)  ← c
      Finpour
      Pour j ← i+1 à n-1 faire   // l'opération :  L_j ← T(i,i).L_j − T(j,i).L_i
            c ← T(j,i)
            pour k ← i à n faire
                  T(j,k)  ← T(i,i).T(j,k)-c.T(i,k) // noter l'importance de c ← T(j,i)
            Finpour
      Finpour
Finpour
x(n-1)  ← T(n-1,n)/T(n-1,n-1)       // à ce stade, le système est triang supérieur, résolvons-le maintenant par remontée
pour i ← n-2 à 0 faire
      s ← 0
      pour j ← i+1 à n-1 faire
            s ← s+T(i,j).x(j)
      finpour
      x(i)  ← (T(i,n)-s)/T(i,i)
finpour
ecrire (la solution du système est :)   // affichage de la solution du système
pour i ← 0 à n-1 faire
       ecrire (x(i))
finpour
Fin.
```

**Traduction en C++ :**

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
main()
{
    int n,i,j,k;
    float s,c;
    printf("entrer l'ordre du systeme de Cramer a resoudre:\n");
```

```
    scanf("%d",&n);
    float T[n][n+1];   //matrice du système & second membre du système
    float x[n];        // vecteur inconnu
    for(i=0;i<=n-1;i++)
    {
                for(j=0;j<=n-1;j++)
                {
                        printf("entrer le (%d,%d)eme coeff du systeme\n",i,j);
                        scanf("%f",&T[i][j]);
                }
    }
    for(i=0;i<=n-1;i++)
    {
                printf("entrer le %d eme coeff du second membre du systeme\n",i);
                scanf("%f",&T[i][n]);
    } // le second terme du syst est entré
   for(i=0;i<=n-1;i++)
    {
                j=i;
                while(T[j][i]==0)  j=j+1; // le j vérifie T[j][i] différent de 0
               for(k=i;k<=n;k++)
                {
                        c=T[i][k];
                        T[i][k]=T[j][k];
                        T[j][k]=c;
                }         // les lignes Li et Lj sont permutées, et donc le pivot Tii<>0
                for(j=i+1;j<=n-1;j++)
                {
                            c=T[j][i];
                            for(k=i;k<=n;k++)  T[j][k]=T[i][i]*T[j][k]-c*T[i][k];
                } // on annule ce qui est en bas du (i,i) éme pivot
    }
    x[n-1]=T[n-1][n]/T[n-1][n-1]; // le système est riang sup; resolvons-le par remontée
    for(i=n-2;i>=0;i=i-1)
    {
                s=0;
                for(j=i+1;j<=n-1;j=j+1)  s=s+T[i][j]*x[j];
                x[i]=(T[i][n]-s)/T[i][i];
    } // le syst est maintenant résolu
    printf("la solution du systeme est:\n");
    for(i=0;i<=n-1;i=i+1)    printf("%f  ",x[i]);
    printf("\n"); // pour retourner à la ligne une fois les solutions affichées
    system("pause");
    return 0;
}
```

## LA PROGRAMMATION RECURSIVE

Pour expliquer de quoi il s'agit, nous allons reprendre le fameux exemple: le calcul d'une factorielle « déjà programmé avec une boucle Pour ».

Une autre manière de faire repose sur le fait que quel que soit le nombre n, on a n ! = n x (n-1) !. Autrement dit, la factorielle d'un nombre est ce nombre multiplié par la factorielle du nombre précédent.

Si l'on doit programmer cela, on peut alors imaginer une fonction **Fact**, chargée de calculer la factorielle. Cette fonction effectue la multiplication du nombre passé en argument par la factorielle du nombre précédent. Et cette factorielle du nombre précédent va bien entendu être elle-même calculée par la fonction Fact.

Ainsi, on va créer une fonction qui pour fournir son résultat, va **s'appeler elle-même** un certain nombre de fois. C'est cela, la **récursivité**.

Une question se pose naturellement : *quand ces auto-appels de la fonction **Fact** vont-ils s'arrêter* ?. Ça s'arrête quand on arrive au nombre 0 dont la factorielle est par définition 1.

Cela produit l'écriture suivante :

```
Fonction Fact (N en Numérique)
Si N = 0 alors
  Renvoyer 1
```

```
Sinon
  Renvoyer Fact(N-1) * N
Finsi
Fin Fonction
```

**Traduction en C++ :**

```
#include<stdio.h>
#include<stdlib.h>
int Fact(int N)
{
     if (N==0) return 1;
       else return N*Fact(N-1);
 }
/* vérifions maintenant la fonction Fact */
#include<stdio.h>
#include<stdlib.h>
main()
{
    int n;
    printf("entrer un entier N\n");
    scanf("%d",&n);
    printf("sa factorielle est %d\n",Fact(n));
    system("pause");
    return(0);
 }
```

**Traduction en Maple :**

```
>  Fact:=proc(N)
   if N=0 then 1 else N*Fact(N-1);fi;
   end:
> Fact(5);
```

## Autres exemples :

### 1. Puissance entière

L'idée est juste d'appliquer le fait que : $x^n = x.x^{n-1}$ et que $x^0 = 1$.

Cela produit l'écriture suivante :

```
Fonction puiss (x en Numérique, n en entiers)
Si n = 0 alors  Renvoyer 1
Sinon  Renvoyer x* puiss (x ,n-1)
Finsi
Fin Fonction
```

**Traduction en C++ :**

```
#include<stdio.h>
#include<stdlib.h>
float puiss(float x,int n)
{
     if (n==0) return 1;
       else return  x*puiss(x,n-1);
     }  /* verification de la function puiss */
#include<stdio.h>
#include<stdlib.h>
main()
{
    int n;
    float x;
    printf("entrer un reel x non nul\n");
```

```
        scanf("%f",&x);
        printf("entrer une puissance entiere\n");
        scanf("%d",&n);
        printf("x puissance n est %f\n",puiss(x,n));
        system("pause");
        return(0);
    }
```

**Traduction en Maple :**

```
> puiss:=proc(x,n)
  if n=0 then 1 else x*puiss(x,n-1); fi;
  end:
> puiss(2.3,4);
```

2. pgcd récursif

On applique le lemme d'Euclide : « si a=bq+r alors pgcd(a,b)=pgcd(b,r) » et le résultat : $b/a \Rightarrow pgcd(a,b)=b$.

On obtient :

```
Fonction pgcd(a en entier, b en entier)
Si a mod b = 0 alors  Renvoyer b
Sinon  Renvoyer pgcd( b,a mod b)
Finsi
Fin Fonction
```

**Traduction en C++ :**

```
#include<stdio.h>
#include<stdlib.h>
int pgcd(int a,int b)
{
     if (a%b==0) return b;
        else return pgcd(b,a%b);
     }
#include<stdio.h>
#include<stdlib.h>
main()
{
    int a,b;
    printf("entrer deux entiers non nuls\n");
    scanf("%d%d",&a,&b);
    printf("leur pgcd est: %d\n",pgcd(a,b));
    system("pause");
    return(0);
 }
```

**Traduction en Maple :**

```
> pgcd:=proc(a,b)
  if a mod b =0 then b;
       else pgcd(b, a mod b);
  fi;
  end:
> pgcd(360,270);  igcd(360,270);
```