

# Programming Paradigms

---

# What is a Programming Language?

- Programming Languages 2nd edition Tucker and Noonan
- Online resources

# What is a Programming Language?

- Formal notation for specifying computations, independent of a specific machine
  - Example: a factorial function takes a single non-negative integer argument and computes a positive integer result
  - Mathematically, written as  $\text{fact: nat} \rightarrow \text{nat}$
- Set of imperative commands used to direct a computer to do something useful
  - Print to an output device: `printf("hello world\n")`

# Principal Paradigms

- Programming paradigms are the result of people's ideas about how programs should be constructed
  - ... and formal linguistic mechanisms for expressing them
  - ... and software engineering principles and practices for using the resulting programming language to solve problems
- A programming paradigm is a fundamental style of computer programming.
- Compare with a software development methodology, which is a style of solving specific software engineering problems.

# Principal Paradigms

- A programming paradigm can be understood as an abstraction of a computer system, for example, the von Neumann model used in traditional sequential computers.
- Prevalent computer processing model used is the von Neumann model, invented by John von Neumann in 1945,
  - Data and programs are residing in the memory.
  - Control unit coordinates the components sequentially following the program's instructions.
  - Arithmetic Logical Unit performs the calculations.
  - Input/output provides interfaces to the exterior.
- The program and its data are what is abstracted in a programming language and translated into machine code by the compiler/interpreter.

# Principal Paradigms

- There are four main programming paradigms:
  - Imperative
  - Object-oriented
  - Functional
  - Logic (declarative)
- Very few languages are “pure”. Most combine features of different paradigms
  - The design goal of multi-paradigm languages is to allow programmers to use the best tool for a job, admitting that no one paradigm solves all problems in the easiest or most.

# Imperative Paradigms

- Often thought as a synonym for imperative programming.
  - Specifying the steps, the program must take to reach the desired state.
  - Based upon the concept of the procedure call.
  - Procedures, also known as routines, subroutines, methods, or functions that contain a series of computational steps to be carried out.
  - Any given procedure might be called at any point during a program's execution, including by other procedures or itself.
- Using a procedural language, the programmer specifies language statements to perform a sequence of algorithmic

# Imperative Paradigms

- Follows the classic von Neumann-Eckert model:
  - Program and data are indistinguishable in memory
  - Program = a sequence of commands
  - State = values of all variables when the program runs
  - Large programs use procedural abstraction

• Example imperative languages:

- Cobol, Fortran, C, Ada, Perl,  
...

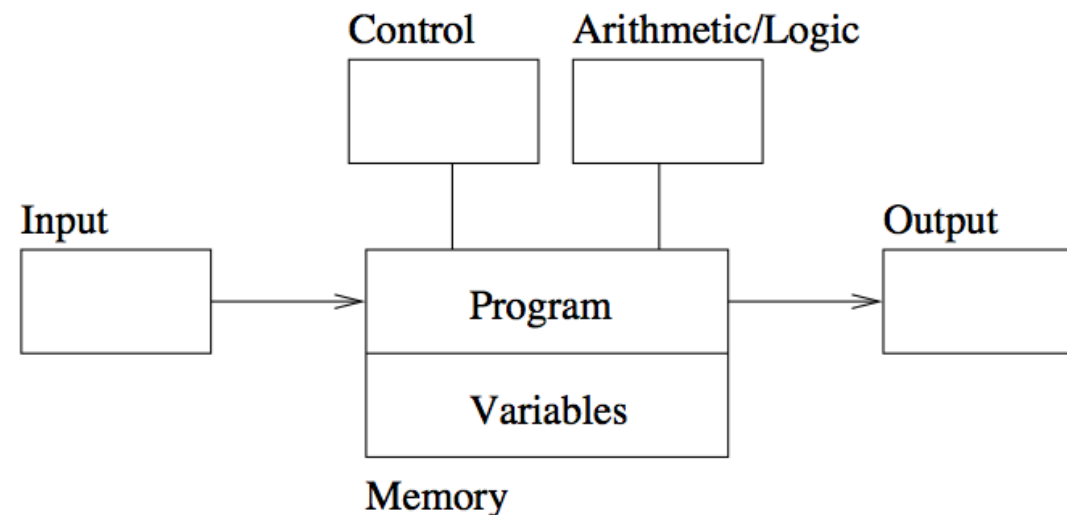


Figure 1.1: The von Neumann-Eckert Computer Model



# Imperative Paradigms

- Possible benefits:
  - Often a better choice than simple sequential or unstructured programming in many situations which involve moderate complexity or require significant ease of maintainability.
  - The ability to reuse the same code at different places in the program without copying it.
  - An easier way to keep track of program flow than a collection of "GOTO" or "JUMP" statements (which can turn a large, complicated program into spaghetti code).
  - The ability to be strongly modular or structured.

# Object-oriented programming paradigm

- Object-oriented programming (OOP) is a programming paradigm that uses "objects" – data structures encapsulating data fields and procedures together with their interactions – to design applications and computer programs.
- The most important distinction is whereas procedural programming uses procedures to operate on data structures, object-oriented programming bundles the two together, so an "object" operates on its "own" data structure.
- Associated programming techniques may include features such as data abstraction, encapsulation, modularity, polymorphism, and inheritance.

# Object-oriented programming paradigm

- An OO Program is a collection of objects that interact by passing messages that transform the state.
- When studying OO, we learn about:
  - Sending Messages
  - Inheritance
  - Polymorphism
- Example OO languages:
  - Smalltalk, Java, C++, C#, and Python

# Functional Paradigm

- Functional programming models a computation as a collection of mathematical functions.
  - Input = domain
  - Output = range
- Functional languages are characterized by:
  - Functional composition
  - Recursion
- Example functional languages:
  - Lisp, Scheme, ML, Haskell, ...

# Functional Paradigm

- Compute Factorial Function
- In Imperative Programming Languages:

```
fact(int n) {  
    if (n <= 1) return 1;  
    else  
        return n * fact(n-1); }
```

- In Functional Programming Languages: Scheme

```
(define (fact n)  
  (if (< n 1) 1 (* n (fact (- n 1))))  
))
```

# A Brief History

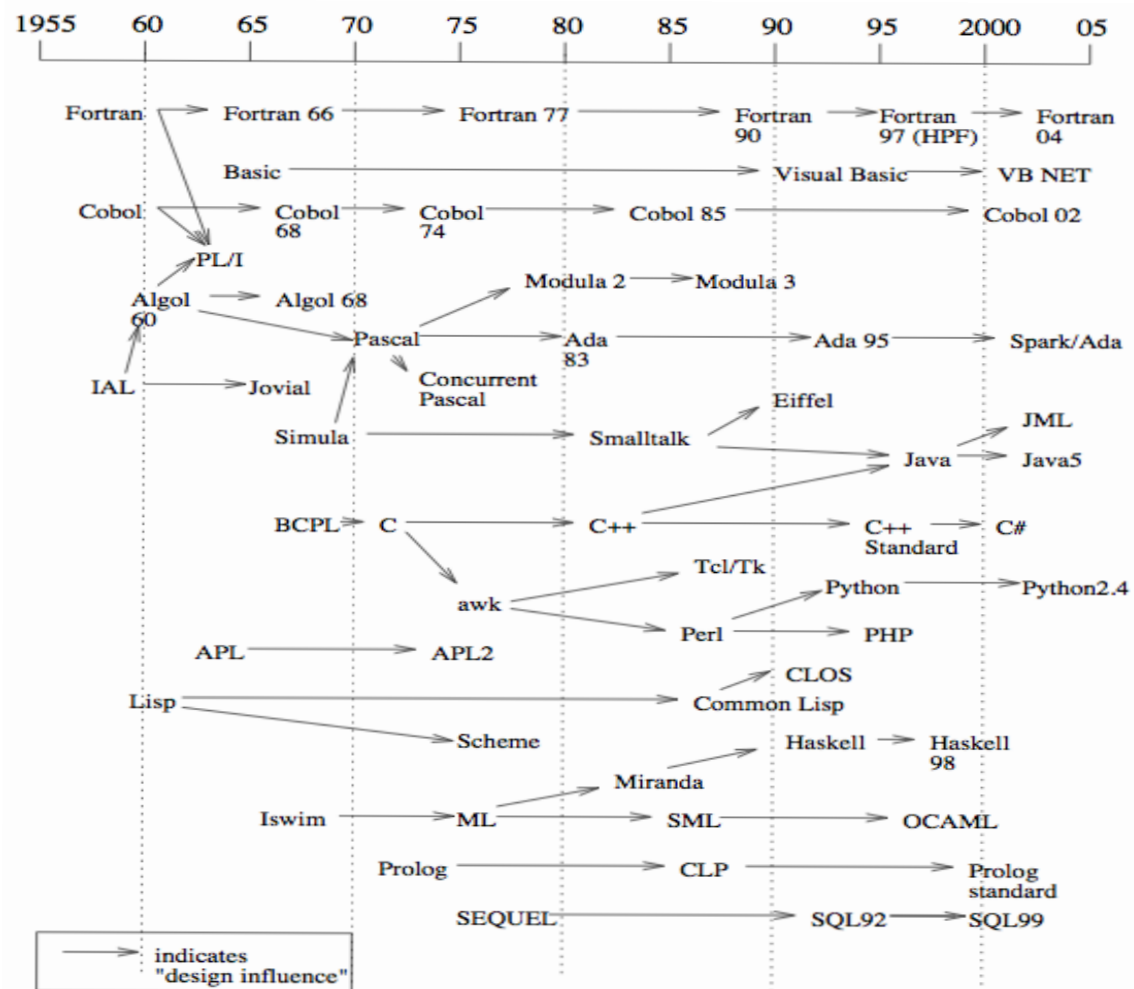


Figure 1.2: A Snapshot of Programming Language History

# A Brief History

- What makes a successful language?
- Key characteristics:
  - Simplicity and readability
  - Clarity about binding
  - Reliability
  - Support
  - Abstraction
  - Orthogonality
  - Efficient implementation

# A Brief History

- **Simplicity and Readability**
- Small instruction set
  - Is a smaller number of instructions better?
  - Simple syntax
- More than one way to accomplish a particular op,
  - A single op has more than one meaning
- Benefits:
  - Ease of learning
  - Ease of programming



# A Brief History

- **Reliability**
- A language is reliable if:
- Program behavior is the same on different platforms
  - E.g., early versions of Fortran
- Type errors are detected
  - E.g., C vs Haskell
- Semantic errors are properly trapped
  - E.g., C vs C++
- Memory leaks are prevented
  - E.g., C vs Java

# A Brief History

- Language support
- Accessible (public domain) compilers/interpreters
- Good texts and tutorials
- Wide community of users
- Integrated with development environments (IDEs)

# A Brief History

- Abstraction in Programming
- Data
  - Programmer-defined types/classes
  - Class libraries
- Procedural
  - Programmer-defined functions
  - Standard function libraries

# A Brief History

- **Compilers and Virtual Machines**
- Compiler – produces machine code
- Interpreter – executes instructions on a virtual machine
- Example compiled languages:
  - Fortran, Cobol, C, C++
- Example interpreted languages:
  - Scheme, Haskell, Python
- Hybrid compilation/interpretation
  - The Java Virtual Machine (JVM)

# A Brief History

- Compilation process

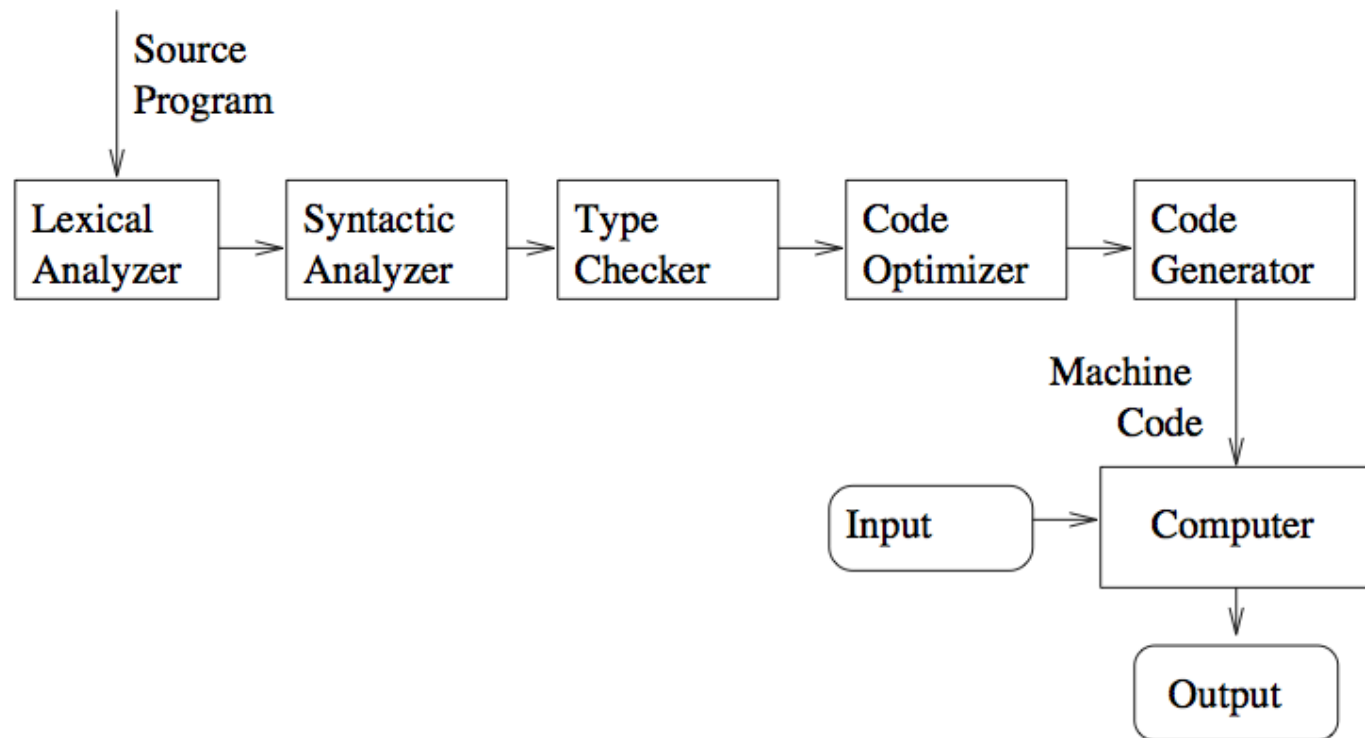


Figure 1.4: The Compile-and-Run Process

# A Brief History

- Interpretation process

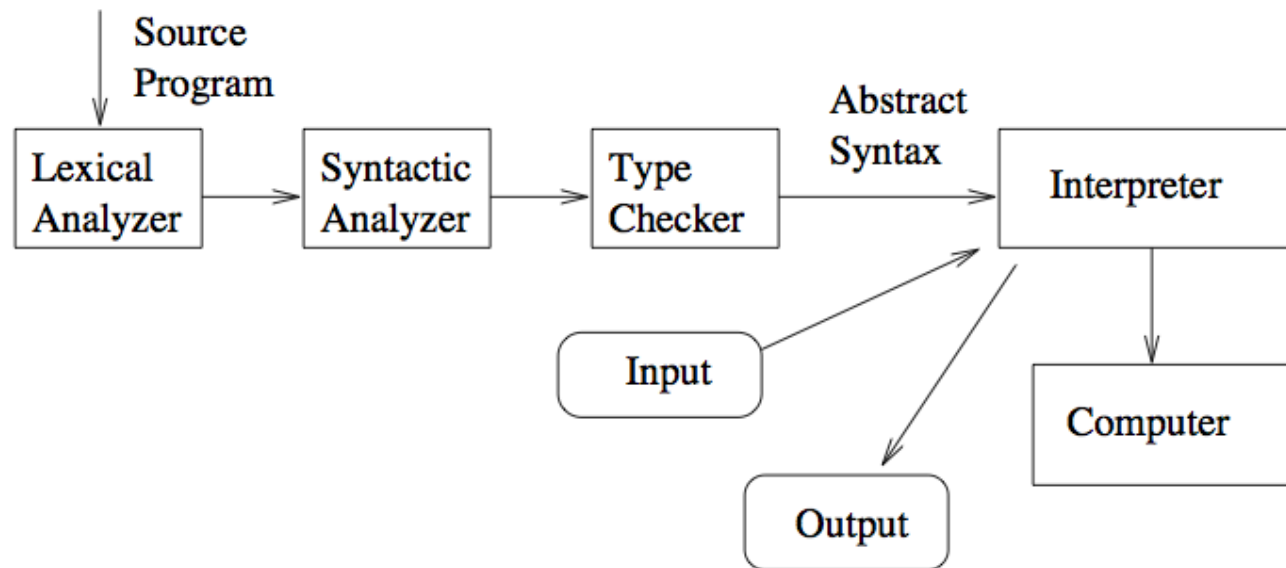


Figure 1.5: Virtual Machines and Interpreters

# Introduction à l'algorithmique

---

# Références

- Structures de Données et Algorithmes, A. Aho, J. Hopcroft, J. Ullman. InterEditions.
- Types de Données et Algorithmes, C. Froidevaux, M-C. Gaudel, M. Soria. Editions McGRAW-Hill.
- <http://www.pise.info/algo/introduction.htm>
- <http://cours.univ-nancy2.fr/course/view.php?id=89>
- Cours en ligne sur internet



# Plan

---

- Introduction
- Algorithmique
- Structure d'un algorithme
- Programme et sous-programme

# Introduction

- Mais c'est quoi un algorithme?

Abu Abdullah Muhammad bin Musa al-Kharazmi  
ca. 780 - 850



# Introduction

- Mais c'est quoi un algorithme?

**Définition 1:** Une description des différentes étapes permettant de résoudre un problème quelconque

**Définition 2 :** Une suite d'instructions, qui une fois exécutée correctement, conduit à un résultat donné

**Définition 3:** L'algorithmique exprime les instructions résolvant un problème donné indépendamment des particularités de tel ou tel langage

# Introduction

- Mais c'est quoi un algorithme?

En conclusion : un algorithme est une description complète et détaillée des actions à effectuer et de leur séquençement pour arriver à un résultat donné

- Intérêt: séparation analyse/codage (pas de préoccupation de syntaxe)
- Qualités: exact (fournit le résultat souhaité), efficace (temps d'exécution, mémoire occupée), clair (compréhensible), général (traite le plus grand nombre de cas possibles), ...

L'algorithmique désigne aussi la discipline qui étudie les algorithmes et leurs applications en Informatique

Une bonne connaissance de l'algorithmique permet d'écrire des algorithmes exacts et efficaces

# Introduction

- Définition d'un algorithme

Exemple:

Résolution d'une équation du 2<sup>nd</sup> degré :  $ax^2 + bx + c = 0$

Etapas de résolution :

1. Connaître les valeurs de  $a$ ,  $b$  et  $c$
2. Calculer le discriminant  $\Delta = b^2 - 4ac$
3. Si  $\Delta < 0$  alors pas de solution
4. Si  $\Delta = 0$  alors solution double =  $-b/2a$
5. Si  $\Delta > 0$  alors deux solutions

# Introduction

- Définition d'un algorithme
  - Pour écrire un algorithme, on utilise le langage de description d'algorithme ou le pseudo-code
  - Un algorithme est composé de trois parties :
    - Entête de l'algorithme
    - Déclarations des données et fonctions utilisées par l'algorithme
    - Le corps de l'algorithme

# Structure d'un algorithme

- Structure d'un algorithme



Entête

- Il permet l'identification de l'algorithme
- Mot-clé : Algorithme

Déclarations

- Liste exhaustive des variables, constantes, des structures, des fonctions et des procédures utilisées dans le corps de l'algorithme
- Mots-clés : variable, constante, structure, fonction, ...

Corps

- C'est dans cette partie que les tâches (instructions, opérations, ...) de l'algorithme sont placées
- Mots-clés : Début, Fin

# Structure d'un algorithme

- Exemple d'un algorithme

Identificateur de  
l'algorithme

Algorithme somme  
variable X, Y: Entier

Début

$X \leftarrow 4$

Ecrire("Donner la valeur de Y ")

Lire(Y)

Ecrire(X+Y)

Fin

2 variables entières  
sont déclarées

4 instructions  
forment le corps  
de l'algorithme



# Structure d'un algorithme

- Exemple d'un algorithme

Identificateur de  
l'algorithme

Algorithme moyenne

variable n, somme, moyenne: Réel

Début

Ecrire("Donner le nombre n")

Lire(n)

...

somme  $\leftarrow$  ....

moyenne  $\leftarrow$  somme / n

Ecrire(moyenne)

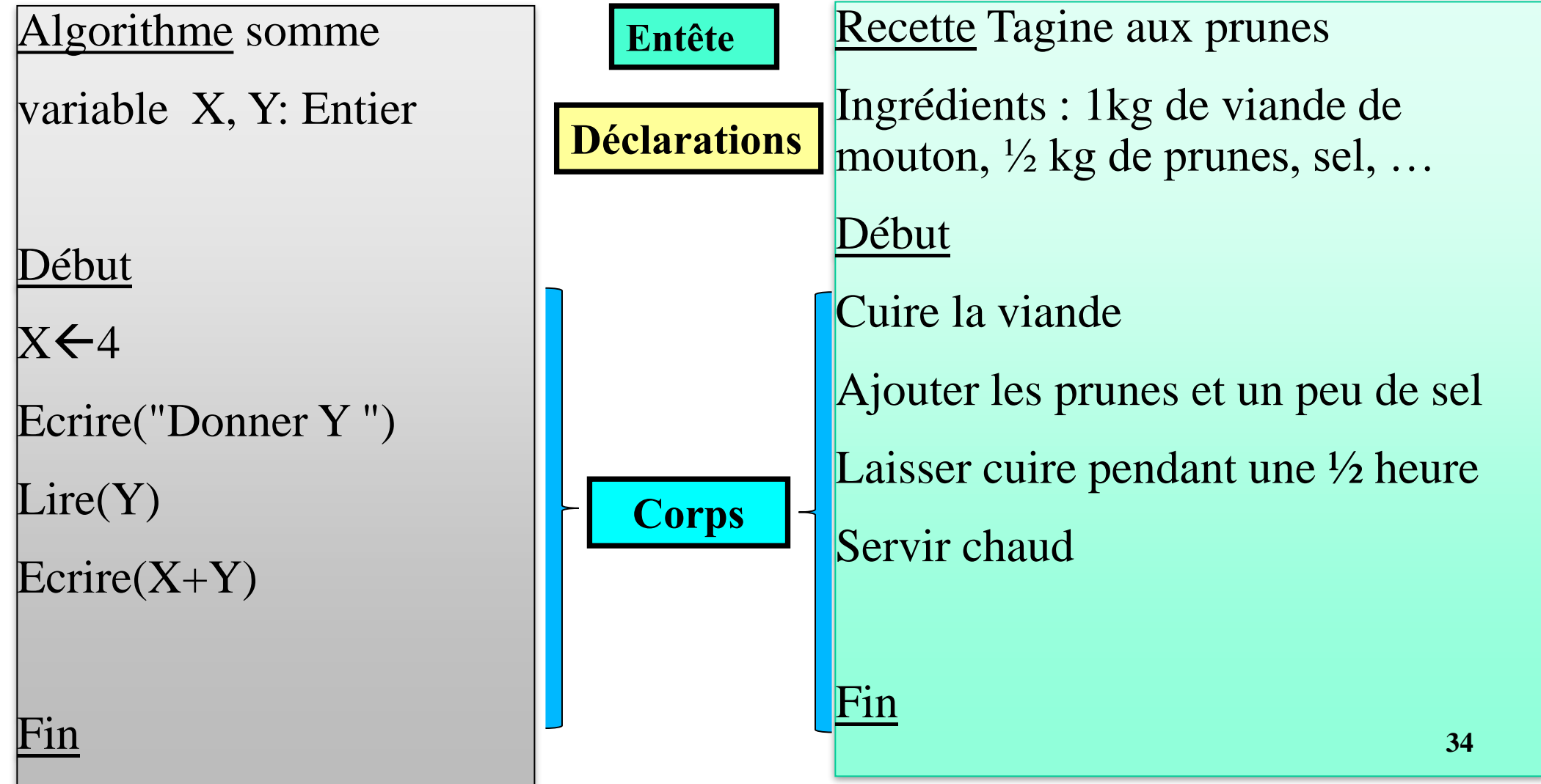
Fin

3 variables réels sont  
déclarées

Instructions  
forment le corps  
de l'algorithme

# Structure d'un algorithme

- Algorithme et recette de cuisine



# Entête

- Entête



Entête

- Il permet l'identification de l'algorithme
- Mot-clé : Algorithme

Déclarations

- Liste exhaustive des variables, constantes, des structures, des fonctions et des procédures utilisées dans le corps de l'algorithme
- Mots-clés : variable, constante, structure, fonction, ...

Corps

- C'est dans cette partie que les tâches (instructions, opérations, ...) de l'algorithme sont placées
- Mots-clés : Début, Fin

# Entête

- Identification d'un algorithme

On a dit que l'entête d'un algorithme permet d'identifier (nommer) un algorithme

Syntaxe :

Dans le langage de description d'algorithme ou le « pseudo-code », le mot réservé **Algorithme** précède le nom de l'algorithme. Le choix du nom est laissé au programmeur

Exemple :

Si je décide d'écrire un algorithme qui réalise la somme de deux réels ayant comme nom « somme », alors l'entête est :

**Algorithme** somme

# Déclarations

- Déclarations



Entête

- Il permet l'identification de l'algorithme
- Mot-clé : Algorithme

Déclarations

- Liste exhaustive des variables, constantes, des structures, des fonctions et des procédures utilisées dans le corps de l'algorithme
- Mots-clés : variable, constante, structure, fonction, ...

Corps

- C'est dans cette partie que les tâches (instructions, opérations, ...) de l'algorithme sont placées
- Mots-clés : Début, Fin

# Déclarations

- **Déclarations**

La partie déclaration dans un algorithme contient une liste exhaustive des déclarations:

- des variables,
- des constantes;
- des structures,
- des fonctions,
- des procédures,

utilisées dans le corps de l'algorithme

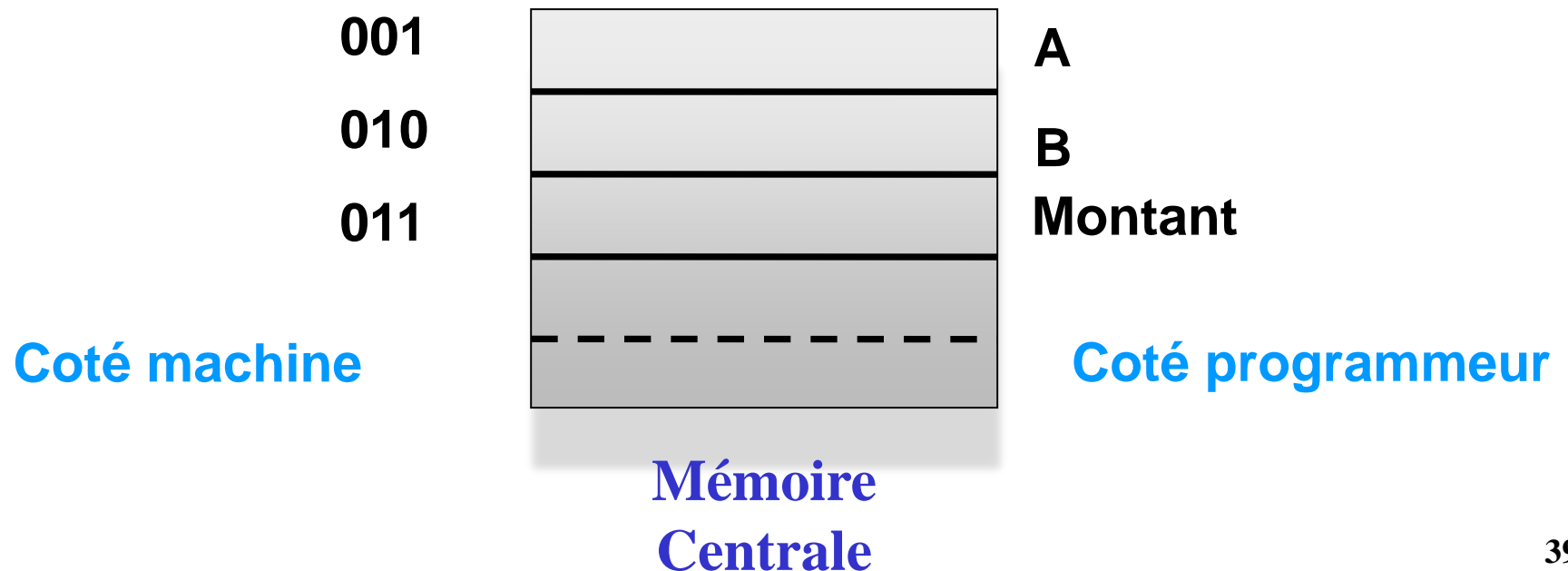
Les variables, constantes, structures, fonctions et procédures utilisées doivent avoir fait l'objet d'une déclaration préalable.

# Déclarations

- Variables : notion de variable**

Les variables servent à « nommer » des emplacements ou adresses de la mémoire

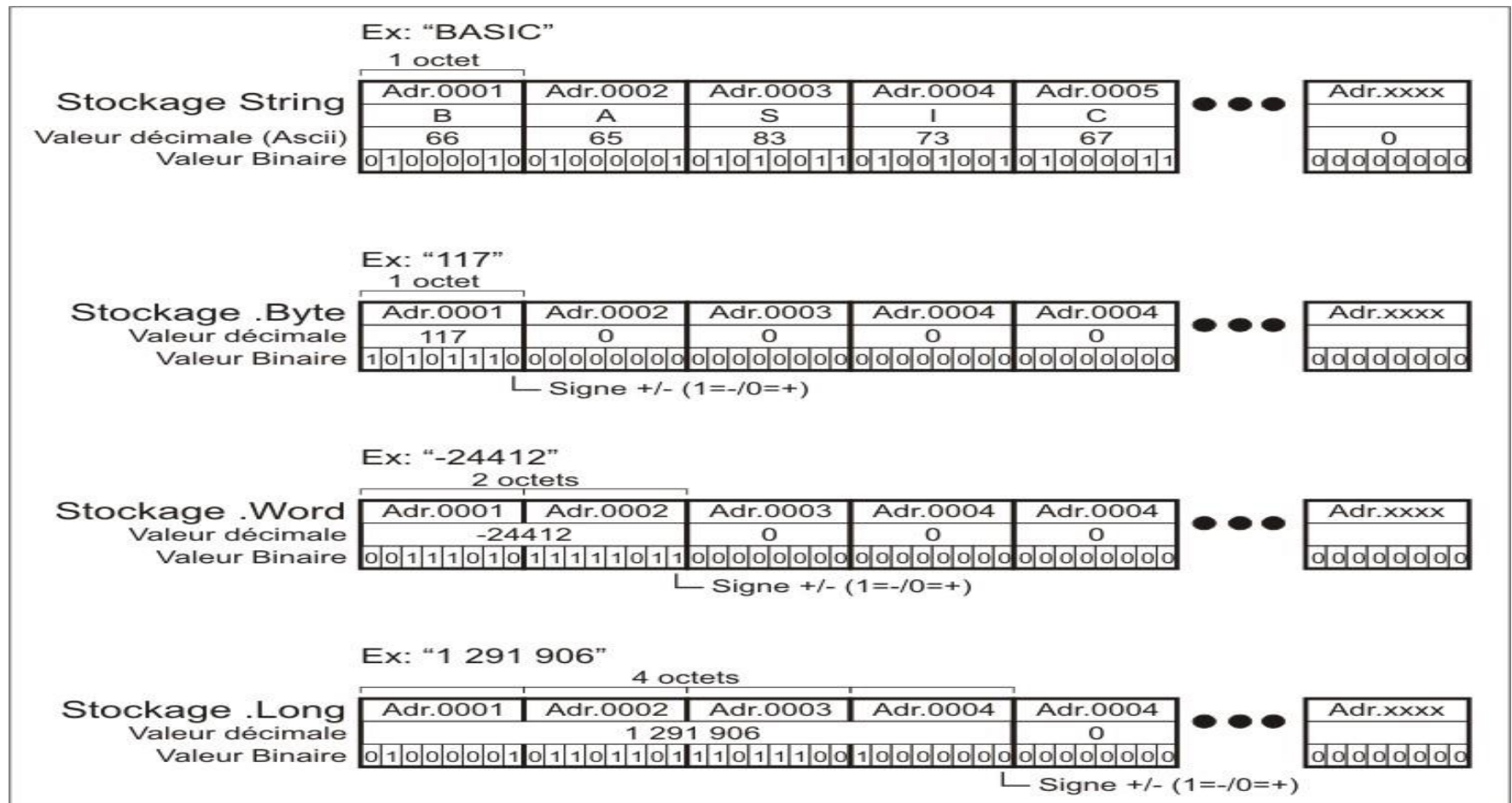
Ils permettent de manipuler des valeurs sans connaître leurs emplacements exacts



# Déclarations

- Variables : notion de variable

Une variable désigne un emplacement mémoire dont le contenu peut changer au cours d'un programme (d'où le nom variable)



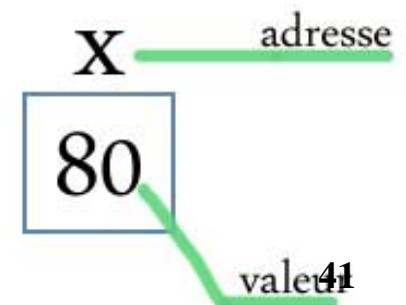


# Déclarations

- Variables : notion de variable

Une variable possède :

- une adresse, adresse binaire repérant l'espace mémoire (mémoire vive –RAM) occupé par la variable.
- un nom, on parle d'identifiant. Il est composé de lettres et chiffres
- une valeur qui peut être modifiée lors de l'exécution de l'algorithme
- un type qui caractérise
  - l'ensemble des valeurs que peut prendre la variable (entier, réel, booléen, chaîne de caractères, ...),
  - quel est l'espace mémoire occupé par une variable
  - les opérations autorisées sur la variable



# Déclarations

- Variables : notion de variable

On peut faire l'analogie avec une armoire d'archive qui contiendrait des tiroirs étiquetés :

- l'**armoire** serait la **mémoire** de l'ordinateur
- les **tiroirs** seraient les **variables** (l'étiquette correspondrait à l'identifiant)
- le contenu d'un tiroir serait la valeur de la variable correspondante
- la couleur du tiroir serait le type de la variable (bleu pour les factures, rouge pour les bons de commande, etc.)



# Déclarations

- Variables : syntaxe

Syntaxe :

```
variable <nom_variable> : <type_variable>
```

avec,

nom\_variable : l'identifiant de la variable,

type\_variable : le type de la variable

# Déclarations

- **Variables : noms**

Le choix des noms de variables est soumis à quelques règles qui varient selon le langage, mais en général:

- Les variables sont nommées par des identificateurs alphanumériques
- Un nom doit commencer par une lettre alphabétique
- Doit être constitué uniquement de lettres, de chiffres et du soulignement \_ (Eviter les caractères de ponctuation et les espaces)
- Doit être différent des mots réservés du langage (par exemple en Java: int, float, else, switch, case, default, for, main, return,)
- La longueur du nom doit être inférieure à la taille maximale spécifiée par le langage utilisé

# Déclarations

- Variables : noms

Brol  
bRol  
brol123



Ok

Brol\_2\_brol\_1  
Brol\_  
b1rol



Ok

1brol  
123  
ma variable



Non

123\_23  
Échange  
Ma-var



Non

\_brol



Eviter

# Déclarations

- Variables : noms

Conseil: pour la lisibilité du code, choisissez des noms significatifs qui décrivent les données manipulées

Exemples: TotalVentes2007, Prix\_TTC, Prix\_HT

Remarque: en pseudo-code algorithmique, on va respecter les règles citées, même si on est libre dans la syntaxe

# Déclarations

- Variables : types

Types (prédéfinis)	Valeurs possibles
Booléen	Vrai ou faux (0 ou 1)
Entier	-2 147 483 648 à 2 147 483 647 (dépend du langage utilisé)
Réel	$-1,79 \times 10^{308}$ à $-4,94 \times 10^{-324}$ pour les valeurs négatives $4,94 \times 10^{-324}$ à $1,79 \times 10^{308}$ pour les valeurs positives (dépend du langage utilisé)
Caractère	A, B, a, b, #, 1, 3, .... (lettres, signes de ponctuation, espaces, ou même de chiffres)
Chaîne de caractères	" Bonjour", .... (le maximum de caractères pouvant être stockés dans une seule chaîne dépend du langage utilisé)

# Déclarations

- Variables : types

Certain langage définissent les types supplémentaires :

Type numérique (entier ou réel)

- Byte (codé sur 1 octet): de 0 à 255
- Entier court (codé sur 2 octets) : -32 768 à 32 767
- Entier long (codé sur 4 ou 8 octets)
- Réel simple précision (codé sur 4 octets)
- Réel double précision (codé sur 8 octets)

Remarque: pour le type numérique on va se limiter aux entiers et réels sans considérer les sous types



# Déclarations

- Variables : types

## Exemple 1:

Dans mon algorithme de calcul de la somme de deux réels, je vais avoir besoin d'une variable pour stocker la somme de deux réels. Si je décide de nommer cette variable « sum », sa déclaration est donc :

variable sum : Réel

## Exemple 2:

variable test : Booléen

variable nb : Entier

variable string : Chaîne

variable c : Caractère

# Déclarations

- Variables : types

Comment choisir le type de vos variables : la réponse est simple, selon les données manipulées et la précision souhaitée

- Dans mon algo, j'ai des données qui correspondent à des caractères

Type = caractère

- Dans mon algo, je manipule des valeurs entières

Type = entier

- Dans mon algo, j'ai des réels avec peu de chiffres après la virgule

Type = réel simple précision


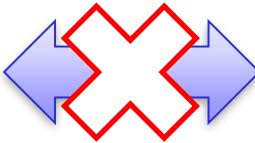
- Dans mon algo, je fais des calculs sur des réels correspondants à des données scientifiques sensibles

Type = réel double précision

# Déclarations

- Variables : types

Remarque 1:

variable somme : Réel	}		variables somme, moyenne : Réel
variable moyenne : Réel			
variable nb : Entier	}		variables nb, moyenne : Réel
variable moyenne : Réel			

Remarque 2

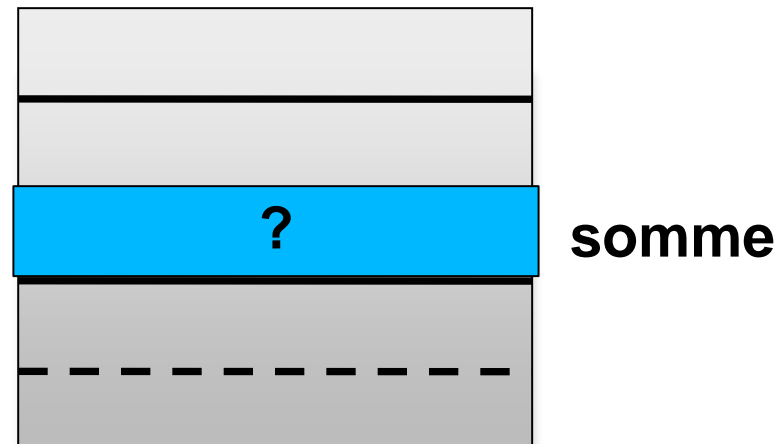
- Une chaîne de caractères est toujours notée entre guillemets " ....  
"
- Un caractère est toujours noté entre apostrophes ' '
- Eviter la confusion:
  - entre des nombres et des suites de caractères chiffres.
  - entre le nom d'une variable et son contenu

# Déclarations

- Variables : valeur d'une variable

Le fait de déclarer une variable ne veut pas dire que cette dernière a une valeur : sa valeur est indéfinie

Exemple : variable somme : Réel



**Mémoire  
Centrale**

# Déclarations

- **Constantes**

Une constante est une variable qui n'évolue pas (ne change pas de valeur).

Deux types de constantes :

- Constante implicite : 2, 2.3, "ABCD ", 'a'
- Constante explicite : définie explicitement.

Syntaxe

```
Constante nom_constant ← valeur_constant : Type
```

# Déclarations

- Constantes

## Exemple

Constante MAX  $\leftarrow$  10 : Entier

Constante True  $\leftarrow$  Vrai : Booléen

Constante False  $\leftarrow$  0 : Entier

Constante NestPasEntier  $\leftarrow$  10.0 : Réel

Constante Vrai  $\leftarrow$  1 : Booléen

## Remarque

Une constante explicite à une valeur et un type.

# Déclarations

- Constantes : exercices

Exercice :

Pourquoi on a besoin d'un type constante explicite?

Corrigé :

L'utilisation des constantes n'est pas obligatoire. Cependant, elles donnent plus de lisibilité à l'algorithme et facilite largement le maintien et la modification du code.

# Déclarations

- Déclarations : et le reste?

D'autres types de variables existent, comme les tableaux et les structures. Voir plus loin dans ce cours

La déclaration fonctions et des procédures est présentée plus loin dans ce cours



# Corps

- Corps



Entête

- Il permet l'identification de l'algorithme
- Mot-clé : Algorithme

Déclarations

- Liste exhaustive des variables, constantes, des structures, des fonctions et des procédures utilisées dans le corps de l'algorithme
- Mots-clés : variable, constante, structure, fonction,...

Corps

- C'est dans cette partie que les tâches (instructions, opérations,...) de l'algorithme sont placées
- Mots-clés : Début, Fin

# Corps

- Structure du corps d'un algorithme

**Rôle du corps** : définir les tâches (instructions, opérations,...) que l'algorithme doit effectuer pour résoudre l'énoncé d'un problème.

Syntaxe :

Début

instruction 1

instruction 2

.....

.....

Fin

Exemple :

Début

$X \leftarrow 4$

Ecrire("Donner la valeur de Y")

Lire(Y)

Ecrire(X+Y)

Fin

# Corps

- **Instructions**

Une instruction est la définition d'une action, qui lorsqu'elle exécutée, modifie l'état d'un programme

On peut identifier comme instructions de base en algorithmique :

- Les instruction d'affectation
- Les instruction d'entrée/sortie (dites aussi de communication)

# Corps

- Instructions d'affectation

**Rôle :** mettre une valeur dans un emplacement mémoire désigné par son nom.

Syntaxe:

1. `nom_variable ← valeur`
2. `nom_variable1 ← nom_variable2`
3. `nom_variable ← expression`

Exemple

`Note ← 15`

`Note1 ← Note2`

`Moyenne ← (Note2*2 + Note1)/3`

`Nom ← "Loulou"`

# Corps

- Instructions d'affectation : remarques

L'instruction  $c \leftarrow a + b$  se comprend de la façon suivante:

- On prend la valeur contenue dans la variable a
- On prend la valeur contenue dans la variable b
- On additionne ces deux valeurs
- On met ce résultat dans la variable c
- Si c avait auparavant une valeur, cette dernière est perdue

# Corps

- Instructions d'affectation : remarques

La valeur affecté doit être compatible avec le type de la variable utilisée

Variables  $i, j, k$  : entier

Variables  $x, y$  : réel

Variables  $OK$  : booléen

Variables  $ch1, ch2$  : chaîne de caractères

Exemples valides:

$i \leftarrow 1$

$j \leftarrow i$

$k \leftarrow i + j$

$x \leftarrow 10.3$

$OK \leftarrow \text{FAUX}$

$ch1 \leftarrow \text{"SMI"}$

$ch2 \leftarrow ch1$

$x \leftarrow 4$

$x \leftarrow j$

Exemples non valides:

$i \leftarrow 10.3$

$OK \leftarrow \text{"SMI"}$

$j \leftarrow x$

# Corps

- Instructions d'affectation : exercices

Exercice 1 :

A quoi seront égales les variables A et B après l'exécution de la suite d'instructions suivantes :

1.  $A \leftarrow 5$
2.  $B \leftarrow A+4$
3.  $A \leftarrow A+1$
4.  $B \leftarrow A-4$

# Corps

- Instructions d'affectation : exercices

Corrigé

Instruction	valeur de A	Valeur de B
0:	?	?
1: $A \leftarrow 5$	5	?
2: $B \leftarrow A+4$	5	9
3: $A \leftarrow A+1$	6	9
4: $B \leftarrow A-4$	6	2

A la fin,  $A=6$  et  $B=2$



# Corps

- Instructions d'affectation : exercices

## Exercice 2 :

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

Variables A, B : Entier

Début

$A \leftarrow 3$

$B \leftarrow 2$

$A \leftarrow B$

$B \leftarrow A$

Fin

Moralité : les deux dernières instructions permettent-elles d'échanger les deux valeurs de B et A ? Si l'on inverse les deux dernières instructions, cela change-t-il quelque chose ?

# Corps

- Instructions d'affectation : exercices

Corrigé :

Après            La valeur des variables est :

$A \leftarrow 3$	$A = 3$	$B = ?$
------------------	---------	---------

$B \leftarrow 2$	$A = 3$	$B = 2$
------------------	---------	---------

$A \leftarrow B$	$A = 2$	$B = 2$
------------------	---------	---------

$B \leftarrow A$	$A = 2$	$B = 2$
------------------	---------	---------

Les deux dernières instructions ne permettent donc pas d'échanger les deux valeurs de B et A, puisque l'une des deux valeurs (celle de A) est ici écrasée.

Si l'on inverse les deux dernières instructions, cela ne changera rien du tout, hormis le fait que cette fois c'est la valeur de B qui sera écrasée.

# Corps

- Instructions d'affectation : exercices

## Exercice :

Plus difficile, mais c'est un classique absolu, qu'il faut absolument le maîtriser : écrire un algorithme permettant d'échanger les valeurs de deux variables A et B, et ce, quelque soit leur contenu préalable.

## Corrigé :

Début

```
...  
C ← A  
A ← B  
B ← C
```

Fin

On est obligé de passer par une variable dite temporaire (la variable <sup>67</sup>

# Corps

- Instructions d'affectation : remarques

Beaucoup de langages de programmation (C/C++, Java, ...) utilisent le signe égal = pour l'affectation  $\leftarrow$ . Attention aux confusions :

- l'affectation n'est pas commutative :  $A=B$  est différente de  $B=A$
- l'affectation est différente d'une équation mathématique :
  - $A=A+1$  a un sens en langages de programmation
  - $A+1=2$  n'est pas possible en langages de programmation et n'est pas équivalente à  $A=1$

Certains langages donnent des valeurs par défaut aux variables déclarées. Pour éviter tout problème il est préférable d'initialiser les variables déclarées

# Corps

- **Opération sur les variables**

Un **opérateur** est un symbole d'opération qui permet d'agir sur des variables ou de faire des “calculs”

Une **opérande** est une entité (variable, constante ou expression) utilisée par un opérateur

Une **expression** est une combinaison d'opérateur(s) et d'opérande(s), elle est évaluée durant l'exécution de l'algorithme, et possède une valeur (son interprétation) et un type.

Par exemple dans  $a+b$  :

- $a$  est l'opérande gauche
- $+$  est l'opérateur
- $b$  est l'opérande droite
- $a+b$  est appelé une expression
- Si  $a$  et  $b$  sont des entiers, l'expression  $a+b$  est un entier

# Corps

- Opération sur les variables

Les opérateurs dépendent du type de l'opération, ils peuvent être :

- des opérateurs arithmétiques: +, -, \*, /, % (modulo), ^ (puissance)
- des opérateurs logiques: NON, OU, ET
- des opérateurs relationnels: =, <, >, <=, >=
- des opérateurs sur les chaînes: + (concaténation)

Une expression est évaluée de gauche à droite mais en tenant compte de priorités

# Corps

- Opération sur les variables

Types	Opérateur disponibles	Type du résultat	Exemple
Booléen	NON, ET, OU, =,	Booléen	Variable a,b : Booléen a=b, NON a, a ET b
Entier	+, -, *, DIV, mod	Entier	variable a, b : Entier I)a+b, a-b, a DIV b, a mod b II)a/b III)a=b, a <b, a>b
	/	Réel	
	=, , <, >, "	Booléen	
Réel	+, -, *, /	Réel	variable a, b : Réel I)a+b, a-b, a*b, a/b II)a=b, a<b, a>b
	=, , <, >, "	Booléen	
Chaîne de caractères	+	Chaîne de caractères	variable s1, s2 : Chaîne I)s1+s2 II)s1=s2
	=	Booléen	

# Corps

- Opération sur les variables : exercices

## Exercice 1 :

Si a vaut 2 et b 3, l'expression  $a+b$  vaut ?

"bonjour" + " tout le monde" vaut ?

$11 \div 2$  vaut ?

$11 \bmod 2$  vaut ?

a	NON a
Vrai	?
Faux	?

a	b	a OU b
Vrai	Vrai	?
Vrai	Faux	?
Faux	Vrai	?
Faux	Faux	?

a	b	a ET b
Vrai	Vrai	?
Vrai	Faux	?
Faux	Vrai	?
Faux	Faux	?



# Corps

- Opération sur les variables : exercices

Corrigé :

Si a vaut 2 et b 3, l'expression  $a+b$  vaut 5

"bonjour" + " tout le monde" vaut "bonjour tout le monde"

$11 \div 2$  vaut 5

$11 \bmod 2$  vaut 1

a	NON a
Vrai	Faux
Faux	Vrai

a	b	a OU b
Vrai	Vrai	Vrai
Vrai	Faux	Vrai
Faux	Vrai	Vrai
Faux	Faux	Faux

a	b	a ET b
Vrai	Vrai	Vrai
Vrai	Faux	Faux
Faux	Vrai	Faux
Faux	Faux	Faux

# Corps

- Opération sur les variables : priorité

Pour les opérateurs arithmétiques, l'ordre de priorité est le suivant (du plus prioritaire au moins prioritaire) :

- $\wedge$  : (élévation à la puissance)
- $*$  ,  $/$  (multiplication, division)
- $\%$  (modulo)
- $+$  ,  $-$  (addition, soustraction)

Exemple :  $2 + 3 * 7$  vaut 23

En cas de besoin (ou de doute), on utilise les parenthèses pour indiquer les opérations à effectuer en priorité

Exemple :  $(2 + 3) * 7$  vaut 35

# Corps

- Opération sur les variables : exercices

## Exercice 2 :

Que produit l'algorithme suivant ?

Variables A, B, C : Chaînes

Début

A ← "423"

B ← "12"

C ← A - B

Fin

## Corrigé :

Il ne peut produire qu'une erreur d'exécution, puisqu'on ne peut pas soustraire des caractères.

# Corps

- Opération sur les variables : exercices

Exercice 3 :

Que produit l'algorithme suivant ?

Variables A, B, C : Chaînes

Début

A ← "423"

B ← "12"

C ← A + B

Fin

Corrigé :

"42312"

# Corps

- Instructions d'entrée/sortie

Rôle d'une instruction de sortie : permet de restituer une valeur.  
Généralement, ça consiste à afficher sur l'écran

Syntaxe:

1. Ecrire (valeur)
2. Ecrire (variable)
3. Ecrire (expression)

Exemple:

Ecrire (4)

Ecrire(Note)

Ecrire ("La moyenne=", (Note1+Note2)/2)

Remarque:

Ecrire(Note) n'est pas la même chose que Ecrire("Note")

# Corps

- Instructions d'entrée/sortie

Rôle d'une instruction d'entrée : permet d'introduire une donnée au programme. Généralement, on tape la valeur

Syntaxe:

Lire(variable)

Exemple:

Lire(Note)

Effet:

- à la rencontre de cette instruction, l'ordinateur arrête l'exécution du programme et attend qu'on tape une valeur.
- On termine la saisie en appuyant sur la touche Entrée.
- La valeur qu'on a tapé est affectée à la variable

Remarque: Lire(valeur) et Lire(expression) n'ont pas de sens

# Corps

---

- Instructions d'entrée/sortie

Conseil :

Avant de lire une variable, il est fortement conseillé d'écrire des messages à l'écran, afin de prévenir l'utilisateur de ce qu'il doit frapper

# Corps

- Instructions d'entrée/sortie : exercice

## Exercice

Ecrire un algorithme qui demande un nombre entier à l'utilisateur, puis qui calcule et affiche le double de ce nombre



# Corps

- Instructions d'entrée/sortie : exercice

Corrigé

Algorithme Calcul\_double

variables A, B : entier

Début

Ecrire("entrer la valeur de A")

Lire(A)

$B \leftarrow 2 * A$

Ecrire("le double de ", A, "est :", B)

Fin

# Corps

- Instructions d'entrée/sortie : exercice

## Exercice

Ecrire un algorithme qui vous demande de saisir votre nom puis votre prénom et qui affiche ensuite votre nom complet

# Corps

- Instructions d'entrée/sortie : exercice

Algorithme AffichageNomComplet

Variables Nom, Prenom, Nom\_Complet : chaîne de caractères

Début

Ecrire("entrez votre nom")

Lire(Nom)

Ecrire("entrez votre prénom")

Lire(Prenom)

Nom\_Complet  $\leftarrow$  Nom + " " + Prenom

Ecrire("Votre nom complet est : ", Nom\_Complet)

Fin

# Corps

- Instructions d'entrée/sortie : exercice

Écrivez un algorithme qui calcule et affiche la circonférence et la surface d'un cercle ( $2 \pi r$  et  $2 \pi r^2$ ). L'algorithme demandera à l'utilisateur d'entrer la valeur du rayon.

# Corps

- Schémas algorithmiques

Les instructions de base d'un algorithme peuvent, en fonction de leur enchaînement, être organisées selon trois familles de schémas algorithmiques :

- Schéma linéaire
- Schéma alternatif ou conditionnel ou choix
- Schéma itératif ou répétitif

# Corps

- **Schémas algorithmiques**

Les instructions de base d'un algorithme peuvent, en fonction de leur enchaînement, être organisées selon trois familles de schémas algorithmiques :

- Schéma linéaire
- Schéma alternatif ou conditionnel ou choix
- Schéma itératif ou répétitif

# Corps

- Schéma linéaire

Le schéma linéaire consiste en une suite d'instructions à exécuter successivement dans l'ordre énoncé

Exemple :

Début

$X \leftarrow 4$

Ecrire("Donner la valeur de Y ")

Lire(Y)

Ecrire(X+Y)

Fin

# Corps

- **Schémas algorithmiques**

Les instructions de base d'un algorithme peuvent, en fonction de leur enchaînement, être organisées selon trois familles de schémas algorithmiques :

- Schéma linéaire
- Schéma alternatif ou conditionnel ou choix
- Schéma itératif ou répétitif



# Corps

- Schéma alternatif ou conditionnel

Le schéma conditionnel permet d'exécuter des instructions quand une condition est vérifiée et éventuellement dans le cas où la condition n'est pas vérifiée

Syntaxe :

```
Si condition Alors  
Début Si  
instructions  
Fin Si
```

```
Si condition Alors  
instructions  
Fin Si
```

```
Si condition Alors  
Début Si  
instructions  
Fin Si  
Sinon  
Début Sinon  
instructions  
Fin Sinon
```

```
Si condition Alors  
instructions  
Sinon  
instructions  
Fin Si
```

avec, « condition » une valeur booléen

# Corps

- Schéma alternatif ou conditionnel

Exemple 1 :

on veut afficher un message quand X est positive

Si  $X > 0$  Alors

Début Si

Ecrire("X est positive")

Fin Si

La condition peut être composée en utilisant des ‘ET’ et des ‘OU’ :

Si ( ((Y=3) OU (Z<4)) ET (X>0) ) Alors

.....

# Corps

- Schéma alternatif ou conditionnel

Exemple 2 :

Si  $X > 0$  Alors

Début Si

Ecrire("X est positive")

Fin si

Sinon

Début Sinon

Ecrire("X n'est pas positive")

Fin Sinon

# Corps

- **Schéma alternatif ou conditionnel**

On peut imbriquer les conditions

```
graph LR; A["Si X > 0 alors"] --- B["Début Si"]; B --- C["Ecrire('X supérieur à 0')"]; C --- D["Fin si"]; D --- E["Sinon"]; E --- F["Début Sinon"]; F --- G["Si X=0 alors"]; G --- H["Début Si"]; H --- I["Ecrire('X égal à 0')"]; I --- J["Fin Si"]; J --- K["Sinon"]; K --- L["Début Sinon"]; L --- M["Ecrire('X inférieur à 0')"]; M --- N["Fin Sinon"]; N --- O["FinSinon"];
```

Si X > 0 alors  
Début Si  
Ecrire("X supérieur à 0")  
Fin si  
Sinon  
Début Sinon  
Si X=0 alors  
Début Si  
Ecrire("X égal à 0")  
Fin Si  
Sinon  
Début Sinon  
Ecrire("X inférieur à 0")  
Fin Sinon  
FinSinon

# Corps

- Schéma alternatif ou conditionnel : exercices

## Exercice :

Ecrire un algorithme qui demande un nombre entier à l'utilisateur, puis qui teste et affiche s'il est divisible par 3

## Corrigé :

Algorithme Divisible\_par3

Variable n : entier

Début

    Ecrire (" Entrez un entier : ")

    Lire (n)

    Si  $(n \bmod 3 = 0)$  alors

        Ecrire (n, " est divisible par 3")

    Sinon

        Ecrire (n, " n'est pas divisible par 3")

    Finsi

Fin

# Corps

- Schéma alternatif ou conditionnel : exercices

Exercice :

Ecrire un algorithme qui calcul le maximum de deux réels a et b

Corrigé :

variable max : Réel

Début

Si  $a < b$  alors

Début Si

max  $\leftarrow$  b

Fin Si

Sinon

Début Sinon

max  $\leftarrow$  a

Fin Sinon

Fin

variable max : Réel

Début

max  $\leftarrow$  b

Si max  $<$  a alors

Début Si

max  $\leftarrow$  a

Fin Si

Fin

# Corps

- **Schéma alternatif ou conditionnel : exercices**

Exercice :

Le prix de photocopies dans une reprographie varie selon le nombre demandé: 0,5 DH la copie pour un nombre de copies inférieur à 10, 0,4DH pour un nombre compris entre 10 et 20 et 0,3DH au-delà.

Ecrivez un algorithme qui demande à l'utilisateur le nombre de photocopies effectuées, qui calcule et affiche le prix à payer

# Corps

- Schéma alternatif ou conditionnel : exercices

Variable copies : entier

Variable prix : réel

Début

Ecrire ("Nombre de photocopies : ")

Lire (copies)

Si copies < 10 Alors

    prix ← copies\*0.5

Sinon

    Si copies < 20

        prix ← copies\*0.4

    Sinon

        prix ← copies\*0.3

    Fin si

Fin si

Ecrire ("Le prix à payer est : ", prix)

Fin



# Corps

- **Schéma alternatif ou conditionnel : exercices**

Exercice :

Ecrivez un algorithme qui permet de discerner une mention à un étudiant selon la moyenne de ses notes :

- "Très bien" pour une moyenne comprise entre 16 et 20  
( $16 \leq \text{moyenne} \leq 20$ )
- "Bien" pour une moyenne comprise entre 14 et 16  
( $14 \leq \text{moyenne} < 16$ )
- "Assez bien" pour une moyenne comprise entre 12 et 14  
( $12 \leq \text{moyenne} < 14$ )
- "Passable" pour une moyenne comprise entre 10 et 12  
( $10 \leq \text{moyenne} < 12$ )

# Corps

- **Schéma alternatif ou conditionnel : cas où**

Lorsque l'on veut comparer la même variable (de type Entier, Naturel, Caractère) à des valeurs successives on peut remplacer la succession de si. . . alors. . . sinon par :

Algorithme menu

Début

Si condition\_1 alors

...

Si condition\_n alors

...

Fin Si

Fin Si

Fin



Algorithme menu

Début

Cas variable vaut

valeur\_1 : ...

valeur\_2 : ...

...

valeur\_n : ...

Fin Cas

Fin

# Corps

- Schéma alternatif ou conditionnel : cas où

Algorithme menu  
Variable a : Entier  
Début  
Lire (a)  
Si a=3 alors  
     $a \leftarrow a+1$   
Fin Si  
Sinon  
    Si a=7 alors  
         $a \leftarrow a+2$   
    Fin Si  
Fin Sinon  
Fin



Algorithme menu  
Variable a : Entier  
Début  
Lire(a)  
Cas a vaut  
    3 :  $a \leftarrow a+1$   
    7 :  $a \leftarrow a+2$   
Fin Cas  
Fin

# Corps

- Schéma alternatif ou conditionnel : cas où

Attention :

Seule une partie des schémas conditionnels peut être traduite en schéma conditionnel cas où

Par exemple Si  $a > 3$  Faire ne peut pas être traduit en schéma cas où

# Corps

- **Schémas algorithmiques**

Les instructions de base d'un algorithme peuvent, en fonction de leur enchaînement, être organisées selon trois familles de schémas algorithmiques :

- Schéma linéaire
- Schéma alternatif ou conditionnel ou choix
- Schéma itératif ou répétitif

# Corps

- Schéma itératif ou répétitif

Un schéma itératif permet de répéter une même action un certain nombre de fois. On parle aussi de boucle.

Deux cas possibles :

- On sait à l'écriture de l'algorithme combien on doit faire d'itérations :
  - Boucle Pour
- On ne sait pas à l'écriture de l'algorithme combien on doit faire d'itérations. La prochaine itération est conditionnée (expression booléenne)
  - Boucle Tant que et
  - Boucle Répéter. . . Tant que

# Corps

- **Schéma itératif ou répétitif : Boucle Pour**

Dans cette boucle, on sait à l'avance le nombre d'itérations à faire. La sortie de la boucle s'effectue lorsque le nombre souhaité de répétitions est atteint.

Pour réaliser un boucle Pour, nous avons besoin d'une variable qui va contrôler le nombre d'itérations. Cette variable est caractérisée par :

- Une valeur initiale
- Une valeur finale
- Un pas de variation

# Corps

- Schéma itératif ou répétitif : Boucle Pour

Syntaxe :

```
Pour variable ← «valeur initiale» à « valeur finale» [de pas p] Faire  
    instructions  
Fin Pour
```

par défaut p vaut 1

```
(autre syntaxe)  
Pour variable allant de « valeur initiale » à « valeur finale » répéter  
    instructions  
Fin pour
```



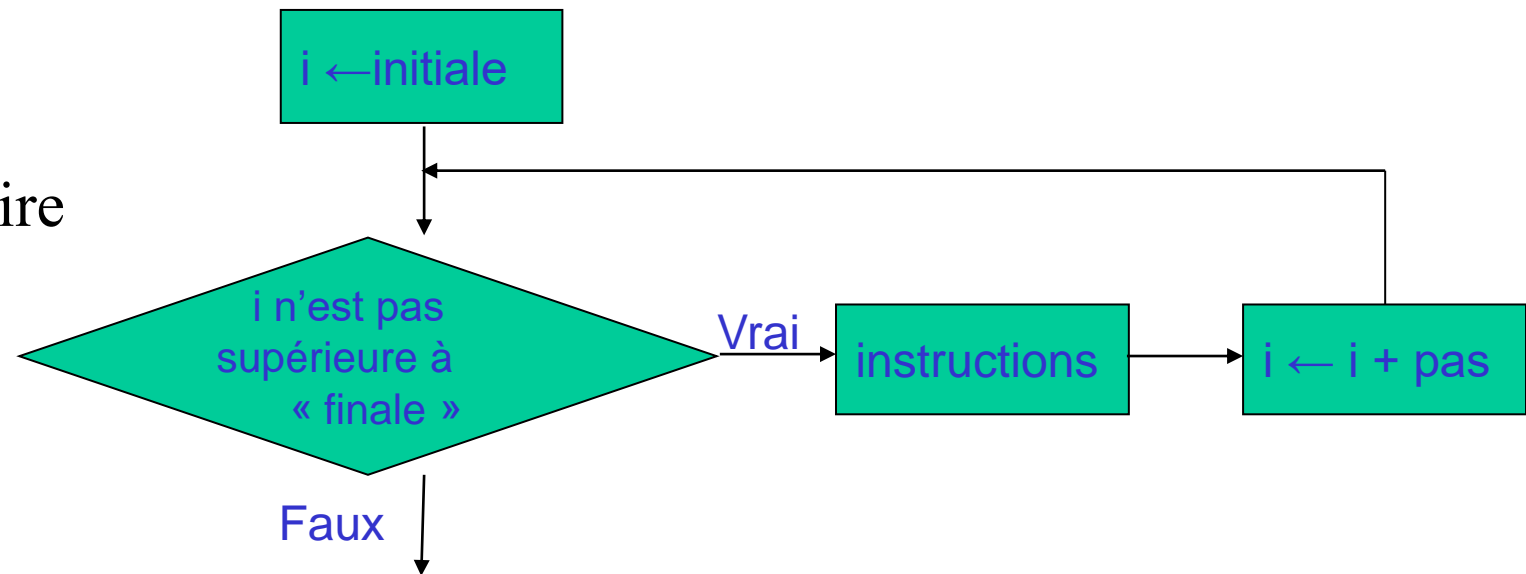
# Corps

- Schéma itératif ou répétitif : Boucle Pour

Déroulement de la boucle Pour :

Pour  $i \leftarrow 1$  à 10 Faire  
Ecrire("Bonjour")  
Fin Pour

Ecrire("Bonjour")



# Corps

- Schéma itératif ou répétitif : Boucle Pour

## Déroulement de la boucle Pour :

- La valeur initiale est affectée à la variable compteur
- On compare la valeur du compteur et la valeur de finale :
  - Si la valeur du compteur est  $>$  à la valeur finale dans le cas d'un pas positif (ou si compteur est  $<$  à finale pour un pas négatif), on sort de la boucle et on continue avec l'instruction qui suit FinPour
  - Si compteur est  $\leq$  à finale dans le cas d'un pas positif (ou si compteur est  $\geq$  à finale pour un pas négatif), instructions seront exécutées
- Ensuite, la valeur de compteur est incrémentée de la valeur du pas si pas est positif (ou décrémenté si pas est négatif)
- On recommence l'étape 2 : La comparaison entre compteur et finale est de nouveau effectuée, et ainsi de suite ...

# Corps

- Schéma itératif ou répétitif : Boucle Pour

Remarque :

- Le nombre d'itérations dans une boucle Pour est connu avant le début de la boucle
- Compteur est une variable de type entier (ou caractère). Elle doit être déclarée
- Pas est un entier qui peut être positif ou négatif. Pas peut ne pas être mentionné, car par défaut sa valeur est égal à 1. Dans ce cas, le nombre d'itérations est égal à finale - initiale + 1
- Initiale et finale peuvent être des valeurs, des variables définies avant le début de la boucle ou des expressions de même type que compteur

# Corps

- Schéma itératif ou répétitif : Boucle Pour

Exemple :

```
Pour i ← 1 à 100 Faire
    Ecrire("donner une note")
    Lire (X)
Fin Pour
Ecrire(i)
```

- La première valeur de i est 1
- A chaque itération, on ajoute 1 à i

```
Pour i ← 1 à 100 de pas 1 Faire
    Ecrire("donner une note")
    Lire (X)
Fin Pour
```

# Corps

- Schéma itératif ou répétitif : Boucle Pour

Exemple :

```
Pour i ← 6 à 8 Faire  
  Ecrire(i)  
Fin Pour
```

Cette boucle sera exécutée 3 fois

```
Pour i ← 10 à 8 Faire  
  Ecrire(i)  
Fin Pour
```

Cette boucle ne sera exécutée  
aucune fois car  $Val1 > Val2$

# Corps

- Schéma itératif ou répétitif : Boucle Pour : exercices

Exercice :

Donner le résultat de l'exécution de l'algorithme suivant

```
Algorithme Affichage
Variable i : Entier
Début
  Pour i ← 6 à 8 Faire
    Ecrire("la valeur de : ", i)
  Fin Pour
  Ecrire("la valeur de : ", i)
Fin
```

# Corps

- Schéma itératif ou répétitif : Boucle Pour : exercices

Corrigé:

la valeur de : 6

la valeur de : 7

la valeur de : 8

la valeur de : 9

# Corps

- Schéma itératif ou répétitif : Boucle Pour : exercices

Exercice :

Donner le résultat de l'exécution de l'algorithme suivant

Algorithme Affichage

Variable i : Entier

Début

Pour i  $\leftarrow$  1 à 8 de pas 2 Faire

    Ecrire("la valeur de : ", i)

Fin Pour

    Ecrire("la valeur de : ", i+10)

Fin



# Corps

- Schéma itératif ou répétitif : Boucle Pour : exercices

Corrigé:

la valeur de : 1

la valeur de : 3

la valeur de : 5

la valeur de : 7

la valeur de : 19

# Corps

- Schéma itératif ou répétitif : Boucle Pour : exercices

Exercice :

Ecrivez un algorithme qui affiche 100 fois la phrase : "je dois absolument passer la colle qui compte 25% de la note finale".

Algorithme Colle

Variable i : Entier

Début

Pour i  $\leftarrow$  1 à 100 Faire

Ecrire ( ".....")

Fin Pour

Fin

# Corps

- Schéma itératif ou répétitif : Boucle Pour : exercices

Exercice :

Écrire un algorithme qui affiche les entiers impairs de 1 à 100.

Algorithme Impair

Variable i : Entier

Début

Pour i  $\leftarrow$  1 à 100 de pas 2 Faire

Ecrire (i)

Fin Pour

Fin

# Corps

- Schéma itératif ou répétitif : Boucle Pour : exercices

Exercice :

Ecrire un algorithme qui fait la somme de 1 à  $n$ , tel que  $n$  est un entier saisi par l'utilisateur

Indée : utiliser

- Une variable « index » pour contrôler la boucle
- Une variable « n » pour saisir l'entier de l'utilisateur
- Une variable « som » pour stocker la somme de 1 à « index » : à l'itération « index », « som » prendra la somme de la valeur ancienne de « som » + la valeur de l'« index »

# Corps

- Schéma itératif ou répétitif : Boucle Pour : exercices

Corrigé :

Algorithme sum

Variables index, n, som: Entier

Début

$som \leftarrow 0$

    Ecrire ("Donner le nombre n")

    Lire (n)

    Pour index  $\leftarrow$  1 à n Faire

$som \leftarrow som + index$

    Fin pour

    Écrire ("La somme de 1 à " , n, " :", som)

Fin

# Corps

- Schéma itératif ou répétitif : Boucle Pour : exercices

Exercice :

Calculer  $x$  à la puissance  $n$  où  $x$  est un réel non nul et  $n$  un entier positif ou nul

# Corps

- Schéma itératif ou répétitif : Boucle Pour : exercices

Corrigé :

Variables x, puiss : réel

Variables n, i : entier

Debut

Ecrire (" Entrez respectivement les valeurs de x et n ")

Lire (x, n)

puiss  $\leftarrow$  1

Pour i allant de 1 à n

    puiss  $\leftarrow$  puiss \* x

FinPour

Ecrire (x, " à la puissance ", n, " est égal à ", puiss)

Fin

# Corps

- Schéma itératif ou répétitif : Boucle Pour : exercices

Exercice :

Ecrire un algorithme qui affiche le produit de tous les nombres compris entre 1 et 10

Idée 1:

- Utiliser deux variables entières  $i$  et  $j$
- $i$  et  $j$  prennent leurs valeurs dans l'intervalle  $[1..10]$
- A chaque nouvelle valeur, on affiche  $i*j$



# Corps

- Schéma itératif ou répétitif : Boucle Pour : exercices

Corrigé :

Algorithme affiche\_produit

Variable i, j : Entier

Début

Pour i  $\leftarrow$  1 à 10 Faire

Pour j  $\leftarrow$  1 à 10 Faire

Ecrire(i, "\*", j, "=", i \* j)

Fin Pour

Fin Pour

Fin

# Corps

- Schéma itératif ou répétitif : Boucle Pour : exercices

Exercice (suite) :

Ecrire un algorithme qui affiche le produit de tous les nombres compris entre 1 et 10

Idée 2:

- Utiliser deux variables entières  $i$  et  $j$
- $i$  prend ses valeurs dans l'intervalle  $[1..10]$
- $j$  prend ses valeurs dans l'intervalle  $[i..10]$
- Ceci nous permettra d'éviter de calculer deux fois le même produit

# Corps

- Schéma itératif ou répétitif : Boucle Pour : exercices

Corrigé :

Algorithme affiche\_produit

Variable i, j : Entier

Début

Pour i  $\leftarrow$  1 à 10 Faire

Pour j  $\leftarrow$  i à 10 Faire

Ecrire(i, " \* ", j, " = ", i \* j)

Fin Pour

Fin Pour

Fin

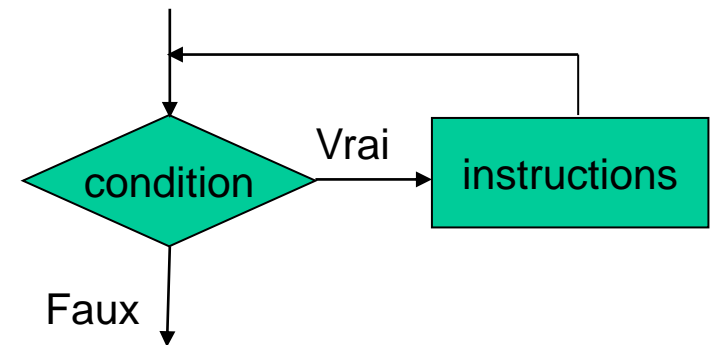
# Corps

- **Schéma itératif ou répétitif : Boucle Tant que**

Dans cette boucle, on ne sait pas à l'avance le nombre d'itérations à faire. La sortie de la boucle s'effectue lorsque la condition n'est plus vérifiée.

Syntaxe :

```
Tant que condition Faire  
    instructions  
Fin Tant que
```



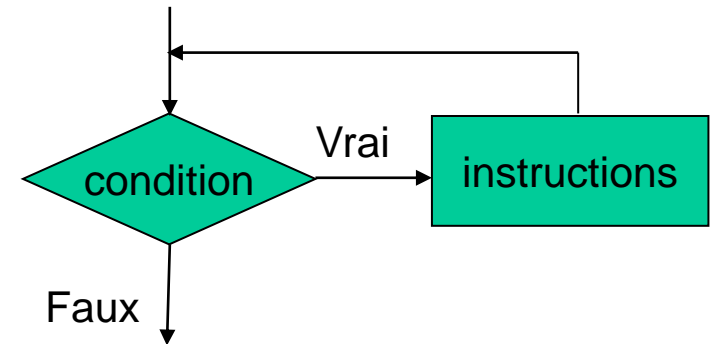
Les instructions à l'intérieur d'une boucle Tant que peuvent ne jamais être exécutées.

# Corps

- **Schéma itératif ou répétitif : Boucle Tant que**

Syntaxe :

Tant que condition Faire  
instructions  
Fin Tant que



La condition (dite condition de contrôle de la boucle) est évaluée avant chaque itération

Si la condition est vraie, on exécute instructions (corps de la boucle), puis, on retourne tester la condition. Si elle est encore vraie, on répète l'exécution, ...

Si la condition est fausse, on sort de la boucle et on exécute l'instruction qui vient après Fin Tant que

# Corps

- Schéma itératif ou répétitif : Boucle Tant que

## Exemple

Algorithme Saisie

Variable n : Entier

Début

    Écrire ("Saisissez un nombre")

    Lire (n)

    Tant que n > 0 Faire

        Écrire ("Saisissez un nombre")

        Lire(n)

    Fin Tant que

Fin

# Corps

- Schéma itératif ou répétitif : Boucle Tant que

Exemple

Supposer maintenant qu'on supprime l'instruction Lire (n)

Algorithme Saisie

Variable n : Entier

Début

~~Lire (n)~~

Tant que  $n > 0$  Faire

    Écrire ("Saisissez un nombre")

    Lire(n)

Fin Tant que

Fin

# Corps

- **Schéma itératif ou répétitif : Boucle Tant que**

Le nombre d'itérations dans une boucle Tant Que n'est pas connu au moment d'entrée dans la boucle. Il dépend de l'évolution de la valeur de condition

Une des instructions du corps de la boucle doit absolument changer la valeur de condition de vrai à faux (après un certain nombre d'itérations), sinon le programme tourne indéfiniment

Attention aux boucles infinies

Exemple de boucle infinie :

$i \leftarrow 2$

Tant Que  $i > 0$

$i \leftarrow i+1$  (attention aux erreurs de frappe : + au lieu de -)

Fin Tant Que



# Corps

- **Schéma itératif ou répétitif : Boucle Tant que**

Contrôle de saisie d'une lettre majuscule jusqu'à ce que le caractère entré soit valable

Variable C : caractère

Debut

Ecrire (" Entrez une lettre majuscule ")

Lire (C)

Tant Que ( $C < 'A'$  ou  $C > 'Z'$ )

Ecrire ("Saisie erronée. Recommencez")

Lire (C)

Fin Tant Que

Ecrire ("Saisie valable")

Fin

# Corps

- **Schéma itératif ou répétitif : Boucle Tant que**

Un algorithme qui détermine le premier nombre entier  $N$  tel que la somme de 1 à  $N$  dépasse strictement 100

version 1

Variables  $sum, i$  : entier

Debut

$i \leftarrow 0$

$sum \leftarrow 0$

Tant Que ( $sum \leq 100$ )

$i \leftarrow i + 1$

$sum \leftarrow sum + i$

Fin Tant Que

Ecrire (" La valeur cherchée est  $N =$ ",  $i$ )

Fin

# Corps

- **Schéma itératif ou répétitif : Boucle Tant que**

Un algorithme qui détermine le premier nombre entier  $N$  tel que la somme de 1 à  $N$  dépasse strictement 100

version 2 : attention à l'ordre des instructions et aux valeurs initiales

Variables  $sum, i$  : entier

Debut

$sum \leftarrow 0$

$i \leftarrow 1$

TantQue ( $sum \leq 100$ )

$sum \leftarrow sum + i$

$i \leftarrow i + 1$

FinTantQue

Ecrire (" La valeur cherchée est  $N =$ ",  $i - 1$ )

Fin

# Corps

- Schéma itératif ou répétitif : lien entre Tant que et Pour

La boucle Pour est un cas particulier de Tant Que (cas où le nombre d'itérations est connu et fixé) . Tout ce qu'on peut écrire avec Pour peut être écrit avec Tant Que (la réciproque est fausse)

Pour compteur allant de « initiale » à « finale » par « pas »  
instructions

Fin Pour

peut être remplacé par :  
(cas d'un pas positif)

compteur  $\leftarrow$  initiale  
Tant Que compteur  $\leq$  finale  
instructions  
compteur  $\leftarrow$  compteur + pas  
Fin Tant Que

# Corps

- **Schéma itératif ou répétitif : lien entre Tant que et Pour**

Calcul de  $x$  à la puissance  $n$  où  $x$  est un réel non nul et  $n$  un entier positif ou nul (version avec Tant Que)

Variables  $x$ ,  $\text{puiss}$  : réel

Variables  $n$ ,  $i$  : entier

Debut

Ecrire (" Entrez respectivement les valeurs de  $x$  et  $n$  ")

Lire ( $x$ ,  $n$ )

$\text{puiss} \leftarrow 1$ ,  $i \leftarrow 1$

TantQue ( $i \leq n$ )

$\text{puiss} \leftarrow \text{puiss} * x$

$i \leftarrow i + 1$

FinTantQue

Ecrire ( $x$ , " à la puissance ",  $n$ , " est égal à ",  $\text{puiss}$ )

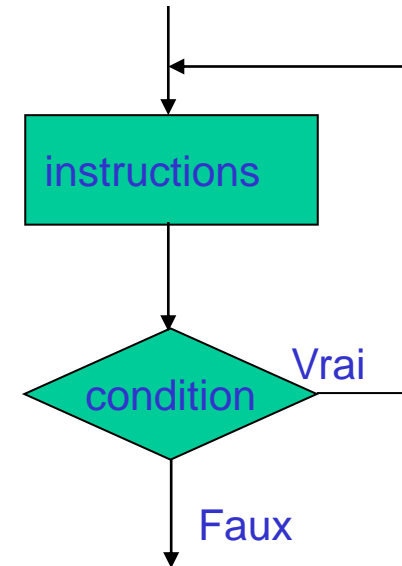
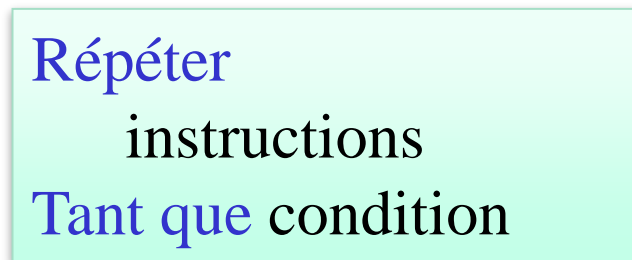
Fin

# Corps

- **Schéma itératif ou répétitif : Boucle Répéter ... Tant que**

Dans cette boucle, on ne sait pas à l'avance le nombre d'itérations à faire. La sortie de la boucle s'effectue lorsque la condition n'est plus vérifiée.

Syntaxe :



Les instructions à l'intérieur d'une boucle Tant sont exécutées au moins une fois.

# Corps

- Schéma itératif ou répétitif : Boucle Répéter Tant que

## Exemple

Algorithme Saisie

Variable n : Entier

Début

    Répéter

        Écrire ("Saisissez un nombre")

        Lire(n)

    Tant que  $n \leq 0$

Fin

# Corps

- **Schéma itératif ou répétitif : Boucle Répéter Tant que**

Exercice : Si l'utilisateur a saisi le couple (2, 3), quel est le message affiché par l'algo. Même question pour la saisie de (4, 3) et de (4, 4).

Algorithme Classement

variable n, m : entier

Début

    Répéter

        Ecrire("Saisissez deux nombres")

        Lire(n,m)

    Tant que  $n=m$

        Si  $n < m$  alors

            Ecrire ("Saisie dans l'ordre croissant")

        Fin Si

    Sinon

        Ecrire ("Saisie dans l'ordre décroissant")

    Fin Sinon



# Corps

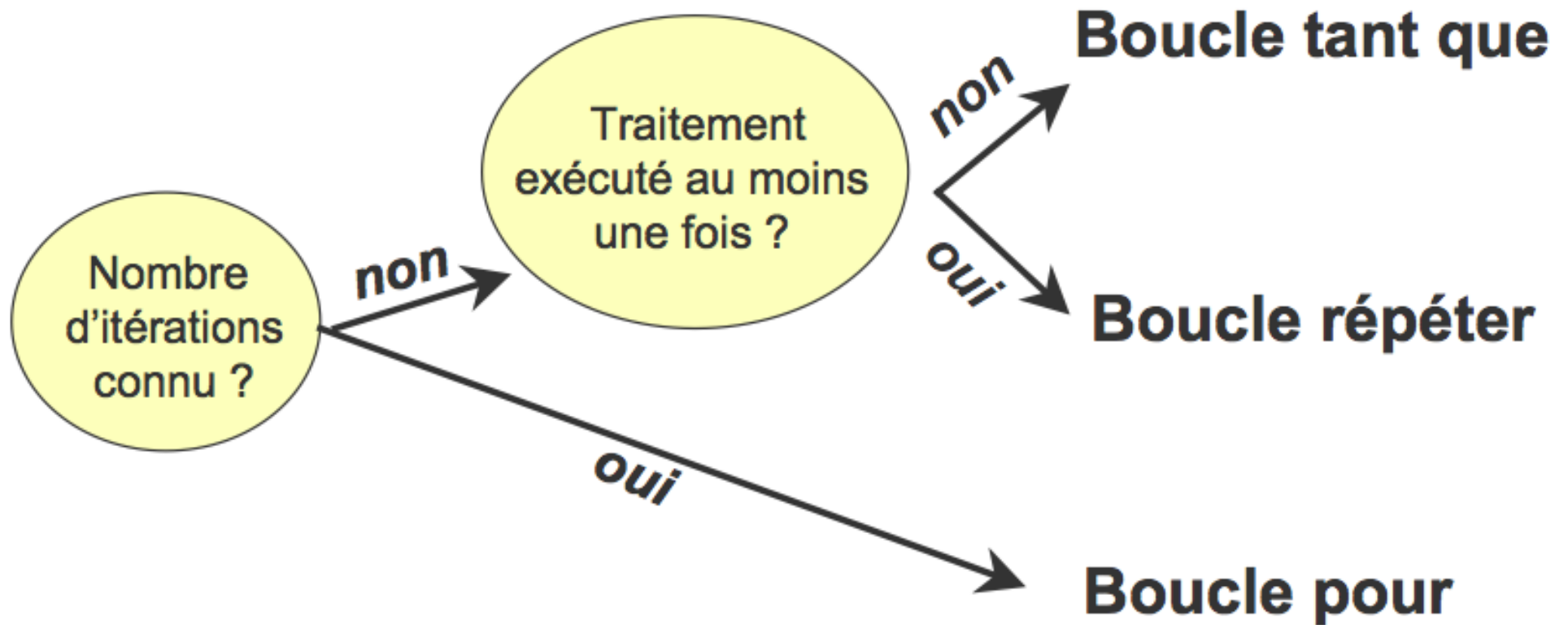
- Schéma itératif ou répétitif : Boucle Tant que : exercices

Corrigé:

à vous de voir

# Corps

- Choix des boucles



# Corps

- Conclusion sur les algorithmes

Le moule d'un algorithme

Algorithme AuNomEvocateur

{déclaration des variables, des constantes, .... }

Début

{préparation du traitement : saisies,....}

{traitements, si itération, la décrire }

{présentation des résultats: affichages,... }

Fin

# Corps

- Conclusion sur les algorithmes
  - Il faut avoir une écriture rigoureuse
  - Il faut avoir une écriture soignée : respecter l'indentation
  - Il existe plusieurs solutions algorithmiques à un problème posé : Il faut rechercher l'efficacité de ce que l'on écrit

# Travaux Pratiques

- Exercice 1

Ecrire un algorithme qui

- Lit 3 notes puis
- Affiche la moyenne

# Travaux Pratiques

- Corrigé

Algorithme moyenne3

Variable a,b,c : Réel

Variable moy : Réel

Début

Ecrire("Entrer trois notes")

Lire (a,b,c)

$\text{moy} \leftarrow (a+b+c)/3$

Ecrire ("La moyenne est",moy)

Fin

Algorithme moyenne3

Variable a,b,c : Réel

Variable moy : Réel

Début

Ecrire("Entrer trois note")

Lire (a)

Lire(b)

Lire(c)

$\text{moy} \leftarrow (a+b+c)/3$

Ecrire ("La moyenne est",moy)

Fin



# Travaux Pratiques

---

- Exercice 2

Ecrire un algorithme qui affiche la valeur absolue d'un réel

# Travaux Pratiques

- Corrigé

```
Algorithme valeurAbsolue
Variable val, résultat : Réel
Début
Ecrire("Entrer un réel")
Lire (val)
Si  $val \geq 0$  alors
    Début Si
        résultat  $\leftarrow$  val
    Fin Si
Sinon
    Début Sinon
        résultat  $\leftarrow$  - val
    Fin Sinon
Ecrire("La valeur absolue de", val, "est", résultat)
Fin
```



# Travaux Pratiques

---

- Exercice 3

Ecrire un algorithme qui à partir d'un nombre entier strictement positif affiche VRAI si ce nombre est pair et FAUX sinon

# Travaux Pratiques

- Corrigé

```
Algorithme EstPair
Variable val : Entier
Début
Ecrire("Entrer un entier strictement positif")
Lire (val)
Si val mod 2 = 0 alors
Début Si
    Ecrire("Vrai")
Fin Si
Sinon
Début Sinon
    Ecrire("Faux")
Fin Sinon
Fin
```

# Travaux Pratiques

- Exercice 4

Ecrire un algorithme qui

- Lit 3 notes puis
- Calcule la moyenne, et
- Si la moyenne est
  - supérieure à 10, affiche le message « Module validé »
  - Inférieur à 5, affiche le message « Note éliminatoire »
  - Dans le reste des cas, affiche le message « Rattrapage »

# Travaux Pratiques

- Corrigé

Algorithme moyenne3bis

Variable a,b,c : Réel

Variable moy : Réel

Début

Ecrire("Entrer trois notes")

Lire (a,b,c)

$moy \leftarrow (a+b+c)/3$

.....

Fin

Si  $moy \geq 10$  alors

Début Si

Ecrire ("Module validé")

Fin Si

Sinon

Début Sinon

Si  $moy < 5$  alors

Début Si

Ecrire ("Note éliminatoire")

Fin Si

Sinon

Début Sinon

Ecrire ("Rattrapage")

Fin Sinon

Fin Sinon

# Travaux Pratiques

---

- Exercice 5

Ecrire un algorithme qui calcule factoriel de 100

# Travaux Pratiques

- Corrigé

Algorithme Factorielle

Variables n, fact : Entier

Début

fact  $\leftarrow$  1

Pour n allant de 1 à 100 répéter

    fact  $\leftarrow$  (n \* fact)

Fin pour

    Ecrire ("La factorielle de 100:", fact)

Fin

Algorithme Factorielle

Variables n, fact : Entier

Début

fact  $\leftarrow$  1

Pour n  $\leftarrow$  1 à 100 Faire

    fact  $\leftarrow$  (n \* fact)

Fin pour

    Ecrire ("La factorielle de 100 est ", fact)

Fin

# Travaux Pratiques

- Exercice 6

Ecrire un algorithme qui affiche la partie entière inférieure de la racine carrée d'un entier positif donné  $n$

# Travaux Pratiques

- Corrigé

Algorithme racineEntière

Variable racine : Entier

Variable n : Entier

Début

racine  $\leftarrow$  1

Ecrire("Entrer un réel positif")

Lire (n)

.....

Fin

Tant que  $\text{racine} * \text{racine} \leq n$  Faire  
racine  $\leftarrow$  racine + 1

Fin Tant que

Ecrire("La racine entière de", n, "est",  
racine-1)



# Travaux Pratiques

---

- Exercice 7

Ecrire une fonction, factorielle, qui calcule pour un entier positif donné  $n$  la valeur de  $n !$

# Travaux Pratiques

- Corrigé

```
Algorithme Factorielle
Variables i, n, fact : Entier
Début
    fact ← 1
    Ecrire ("Donner n")
    Lire (n)
    Pour i ← 1 à n Faire
        fact ← (i * fact)
    Fin Pour
    Ecrire ("La factorielle de ", n "est ", fact)
Fin
```

# Travaux Pratiques

- Exercice 8

Ecrire un algorithme, Puissance, qui à partir d'un réel  $x$  et d'une valeur entière positive  $n$ , retourne  $x$  à la puissance  $n$ .

# Travaux Pratiques

- Corrigé

Algorithme Puissance

Variables i, n: Entier

Variable x, p : Réel

Début

$p \leftarrow 1$

    Ecrire ("Donner x et n")

    Lire (x,n)

    Pour  $i \leftarrow 1$  à n Faire

$p \leftarrow (p * x)$

    Fin Pour

        Ecrire ("x puissance n: ", p)

Fin

# Travaux Pratiques

- Exercice 9

Ecrire un algorithme qui

- Demande le nombre de note que l'utilisateur veut saisir, puis
- Saisit les notes de l'utilisateur
- Affiche la moyenne

Idée :

Vu qu'on sait pas exactement le nombre de notes que l'utilisateur va saisir, on ne peut pas savoir le nombre de variables qu'on besoin pour stocker les notes. Par conséquence, au lieu de stocker chaque note séparément, on va utiliser une variable « cumul » qui va nous servir pour stocker la somme des notes au fur et à mesure que l'utilisateur les saisit.

# Travaux Pratiques

Nombre de notes que  
l'utilisateur veut saisir

- **Corrigé**

Algorithme moyenneN  
Variable nombreNote : Entier  
Variable compteur : Entier  
Variable cumul, moy, noteCourante: Réel

Début

cumul  $\leftarrow$  0

Ecrire("Entrer le nombre de notes à saisir")

Lire (nombreNote)

Pour compteur  $\leftarrow$  1 à nombreNote Faire

    Lire (noteCourante)

    cumul  $\leftarrow$  cumul + noteCourante

Fin Pour

moy  $\leftarrow$  (cumul / nombreNote)

Ecrire ("La moyenne est", moy)

Fin

Variable qui  
contiendra la  
somme des  
notes saisies  
jusqu'à  
maintenant

Variable qui  
stocke la  
valeur de la  
note que  
l'utilisateur  
vient de saisir

# Travaux Pratiques

- Exercice 10

Ecrire un algorithme qui

- Demande le nombre de note que l'utilisateur veut saisir, puis
- Saisit les notes de l'utilisateur et
- Vérifie que chaque note est entre 0 et 20. Si ce n'est pas le cas, l'utilisateur doit ressaisir la note
- Affiche la moyenne

# Travaux Pratiques

- Corrigé

Répéter  
Lire (noteCourante)  
Tant que (noteCourante < 0  
          OU  
          notCourante > 20)

```
Algorithme moyenneN
Variable cumul, nombreNote : Entier
Variable compteur, noteCourante : Entier
Variable moy : Réel
Début
    cumul ← 0
    Ecrire("Entrer le nombre de notes à saisir")
    Lire (nombreNote)
    Pour compteur ← 1 à nombreNote Faire
        -- Lire (noteCourante) --
        cumul ← cumul + noteCourante
    Fin Pour
    moy ← (cumul / nombreNote)
    Ecrire ("La moyenne est", moy)
Fin
```



# Déclarations (suite)

- Déclarations



Entête

- Il permet l'identification de l'algorithme
- Mot-clé : Algorithme

Déclarations

- Liste exhaustive des variables, constantes, des structures, des fonctions et des procédures utilisées dans le corps de l'algorithme
- Mots-clés : variable, constante, structure, fonction, ...

Corps

- C'est dans cette partie que les tâches (instructions, opérations, ...) de l'algorithme sont placées
- Mots-clés : Début, Fin

# Déclarations (suite)

- Déclarations

La partie déclaration dans un algorithme contient une liste exhaustive des déclarations:

- des variables,
- des constantes;
- des structures,
- des fonctions,
- des procédures,

utilisées dans le corps de l'algorithme

Les variables, constantes, structures, fonctions et procédures utilisées doivent avoir fait l'objet d'une déclaration préalable.

# Déclarations (suite)

- **Types composites**

On a vu qu'une variable peut être de type :

- Entier,
- Réel,
- Caractère,
- ...

Ce sont des types simples (élémentaires)

Il existe d'autres types dits composites ou structurés :

- Tableaux
- Structures

# Déclarations (suite)

- Tableaux

Imaginons que dans un programme, nous ayons besoin simultanément de 12 valeurs (par exemple, des notes pour calculer une moyenne). Evidemment, la seule solution dont nous disposons à l'heure actuelle consiste à déclarer douze variables, appelées par exemple N1, N2, N3, etc. et faire

$$\text{Moy} = (N1 + N2 + N3 + N4 + N5 + N6 + N7 + N8 + N9 + N10 + N11 + N12) / 12$$

Mais si on a un programme de gestion avec quelques centaines ou quelques milliers de valeurs à traiter, alors là comment faire?

# Déclarations (suite)

- Tableaux : notions

Heureusement, les langages de programmation offrent la possibilité de rassembler toutes ces variables dans une seule structure de donnée appelée tableau. Donc, un tableau est une collection ordonnée et homogène de valeurs :

- ordonnée car les cases mémoires composant un tableau se suivent
- homogène car toutes les valeurs d'un tableau ont le même type

Il est composé d'un certain nombre de cases

- Chaque case peut prendre une valeur
- Une case est repérée par son indice

8	3	3	5	0	9	4	7
↑ 0	↑ 1	↑ 2	↑ 3	↑ 4	↑ 5	↑ 6	↑ 7

Contenu

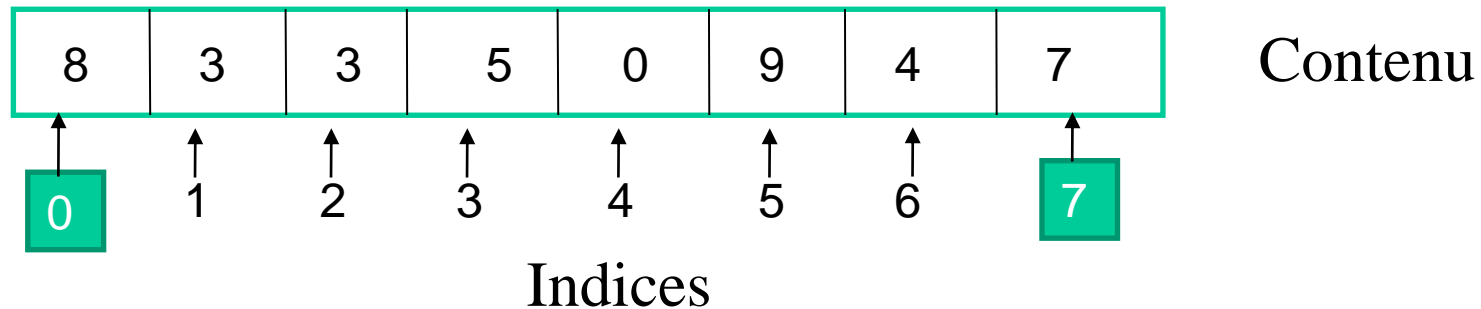
Indices

# Déclarations (suite)

- Tableaux : notions

Un tableau a **une taille fixe : cardinal**, correspondant au nombre des valeurs pouvant être stockées dans la tableau.

Si **N** est la taille du tableau alors **les indices des cases vont de 0 à N-1**



# Déclarations (suite)

- Tableaux : syntaxe

## Syntaxe

```
variable nom_var : Tableau[Taille] de Type
```

## Exemple:

```
variable notes : Tableau[8] d'entiers
```

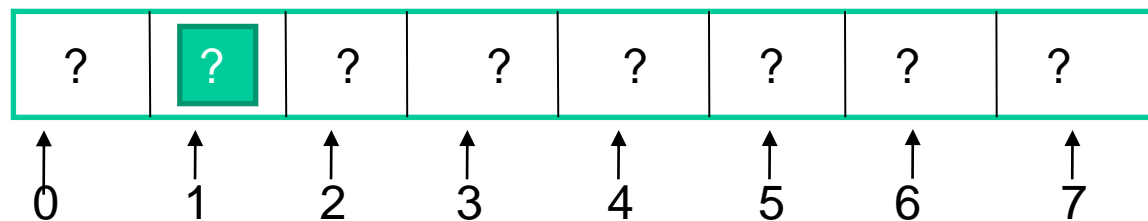
Ceci va créer une variable notes qui est un tableau contenant 8 cases.  
Chaque case peut contenir une valeur entière

# Déclarations (suite)

- Tableaux : syntaxe

**Attention :** Lorsqu'on déclare un tableau, il n'est pas initialiser, c'est-à-dire, les valeurs des cases ne sont pas définies

variable notes : Tableau[8] d'entiers



On peut définir des tableaux de tous types : tableaux d'entiers, de réels, de caractères, de booléens, de chaînes de caractères, ...



# Déclarations (suite)

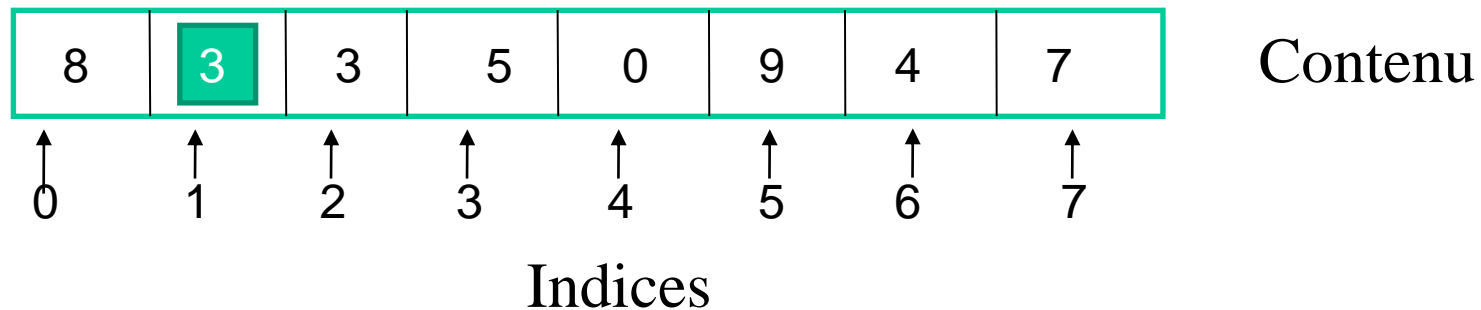
- Tableaux : accès aux tableaux

Accès aux cases des tableaux : se fait à travers l'indice de la case dans la tableau

Exemple

variable notes : Tableau[8] d'entiers

Pour accéder à deuxième case du tableau en écrit notes[1]



Attention : si vous essayez d'accéder à une case dont l'indice est supérieure à la taille du tableau, il se produit une erreur d'exécution

# Corps (suite)

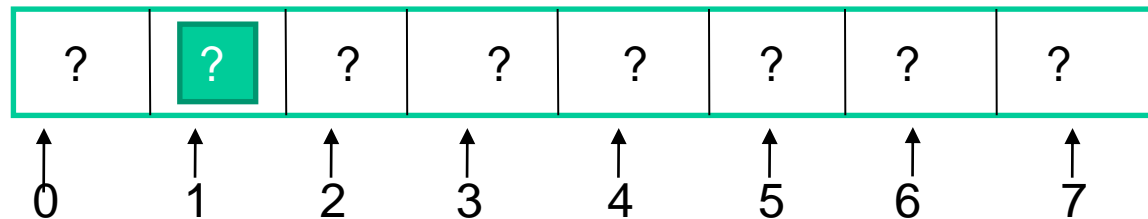
- Tableaux : accès aux tableaux

## Opérations sur les tableaux : l'affectations

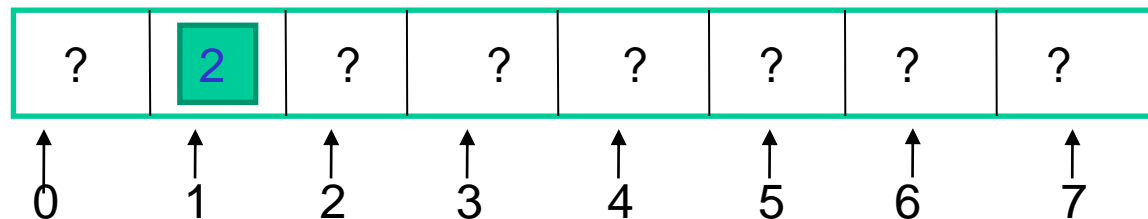
Pour changer la valeur d'une case, on utilise  $\leftarrow$

Exemple

variable notes : Tableau[8] d'entiers



Maintenant si :  $\text{notes}[1] \leftarrow 2$ , on obtient



# Corps (suite)

- Tableaux

Exemple : initialisation d'un tableau à zéro

Algorithme initialisation

Constante  $MAX \leftarrow 20$  : Entier

Variable tab : Tableau [ MAX ] d'entiers

Variable i : entier

Début

Pour i  $\leftarrow 0$  à MAX -1 Faire

    tab[ i ]  $\leftarrow 0$

Fin pour

Fin

# Corps (suite)

- Tableaux

Exemple : initialisation d'un tableau par l'utilisateur

Algorithme initialisation

Constante  $MAX \leftarrow 20$  : Entier

Variable tab : Tableau [ MAX ] d'entiers

Variable i : entier

Début

Pour  $i \leftarrow 0$  à  $MAX - 1$  Faire

Ecrire ("Saisie de l'élément ",  $i + 1$ )

Lire (tab[ i ])

Fin pour

Fin

# Corps (suite)

- Tableaux

Exemple : affichage d'un tableau

Algorithme Affichage

Constante MAX  $\leftarrow$  20 : Entier

Variable tab : Tableau [ MAX ] d'entiers

Variable i : entier

Début

Pour i  $\leftarrow$  0 à MAX -1 Faire

Ecrire (tab[ i ])

Fin pour

Fin

# Corps (suite)

- Tableaux

Exemple : opérations sur les tableaux:

Algorithme Notes

Variables  $i$  : entier

Variable SOM : réel

Variables notes : Tableau [20] de réels

Début

Pour  $i \leftarrow 0$  à 19 Faire

    Répéter

        Ecrire ("Saisie de l'élément ",  $i + 1$ )

        Lire (notes[  $i$  ])

        Tant que ( notes[  $i$  ] < 0 OU notes[  $i$  ] > 20)

    Fin pour

# Corps (suite)

- Tableaux

## Remarques :

- Variables  $n$  : entier
  - Variables notes : Tableau  $[n]$  de réels :  $n$ 'a pas de sens car «  $n$  »  $n$ 'a pas de valeurs.
- Lors de la déclaration d'un tableau, il faut préciser le nombre maximal de cases du tableau :
  - Tableau  $[30]$  de réels : déclaration d'un tableau contenant 30 cases.
- Si le nombre éventuel d'éléments manipulés dans un algorithmique n'est pas connu à l'avance, il faut prévoir une taille fixe maximal du tableau

# Corps (suite)

- Tableaux

Exercice :

- Ecrire un algorithme qui demande à l'utilisateur de saisir 10 notes (utilisation d'un tableau) et qui affiche la moyenne des notes saisies



# Déclaration (suite)

- Tableaux à deux dimensions

Les langages de programmation permettent de déclarer des tableaux dans lesquels les valeurs sont repérées par deux indices. Ceci est utile par exemple pour représenter des matrices

Un tableau à deux dimension peut être vu comme une matrice

Syntaxe

variable *nom\_var* : Tableau[Taille1][Taille2] de Type

Exemple:

variable T : Tableau[10][5] d'entiers

Ceci va créer une variable T qui est un tableau contenant  $10 \times 5 = 50$  cases.

# Déclaration (suite)

- Tableaux à deux dimensions

Exemple : saisie des valeurs d'une matrice 10\*20

Algorithme saisieMatrice

Variable matrice : Tableau[10][20] d'entiers

Variable i, j : Entier

Début

    Pour i ← 0 à 9 Faire

        Pour j ← 0 à 19 Faire

            Lire(matrice[ i ][ j ])

        Fin Pour

    Fin Pour

Fin

# Déclaration (suite)

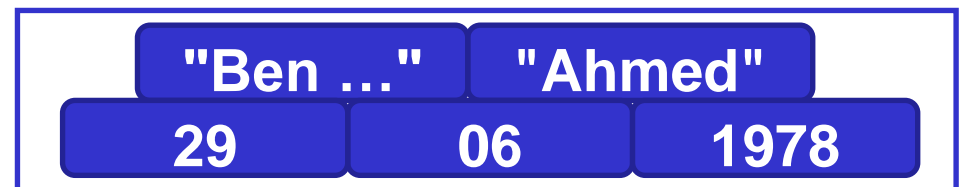
- **Structures**

On a vu que les tableaux permettent de regrouper des informations du même type (Entier, ....).

Mais, comment regrouper des informations de types différents.

Par exemple, une personne à

- Nom (Chaîne)
- Prénom (Chaîne)
- Date de Naissance (elle-même composée du jour + mois + année)



# Déclaration (suite)

- **Structures : syntaxe**

**Rôle : Représenter un "objet" composé d'un ensemble fini d'éléments (champs ou attributs) de types différents.**

## Syntaxe

```
variable nom_variable : structure {  
    champ1 : type1  
    champ2 : type 2  
    .....  
}
```

# Déclaration (suite)

- **Structures : exemple**

```
variable personne : structure {  
    Nom : Chaîne  
    Prénom : Chaîne  
    Date : Tableau[3] d'entiers }
```

**L'accès aux éléments se fait par la notation pointée**

```
personne.Nom ← "Ben ..."
```

```
personne.Prénom ← "Ahmed"
```

```
personne.Date[0] ← 29
```

```
personne.Date[1] ← 06
```

```
personne.Date[2] ← 1978
```

# Déclaration (suite)

- **Type**

**Maintenant, si je veux remplir les informations relatives à deux personnes, je dois faire :**

**variable personnel : structure {**

**Nom : Chaîne**

**Prénom : Chaîne**

**Date : Tableau[3] d'entiers }**

**variable personne2 : structure {**

**Nom : Chaîne**

**Prénom : Chaîne**

**Date : Tableau[3] d'entiers }**

**Puis remplir personnel et personne2!!! Ce n'est pas pratique du**

# Déclaration (suite)

- **Type : syntaxe**

Pour éviter ça, je peux créer mon propre type.

**Type** **Personne** : structure {

**Nom** : Chaîne

**Prénom** : Chaîne

**Date** : Tableau[3] d'entiers }

Le nouveau type **Personne** est manipulé de la même façon que les types prédéfinis :

**Variable moi** : **Personne**

**Variable toi** : **Personne**

# Déclaration (suite)

- **Type : exemple**

Algorithme formulaire

Type Personne : structure {

    Nom : Chaîne

    Prénom : Chaîne

    Date : Tableau[3] d'entiers }

Variable toi : Personne

Début

    toi.Nom ← "Ton nom"

    toi.Prénom ← "Ton prénom"

    toi.Date[0] ← 29

    toi.Date[1] ← 06

    toi.Date[2] ← 1978

Fin



# Déclaration (suite)

- **Type : exemple**

**Algorithme init**

**Constante MAX  $\leftarrow$  10 : Entier**

**Type Vecteur : Tableau[MAX] d'entiers**

**Variable V1 : Vecteur**

**Variable V2 : Vecteur**

**Variable index : Entier**

**Début**

**Pour index de 0 à MAX -1 Faire**

**V1[index]  $\leftarrow$  0**

**V2[index]  $\leftarrow$  0**

**Fin Pour**

**Fin**

# Déclaration (suite)

- **Exercice**

**Supposez que vous avez un point P dans un plan, il est défini par le couple x et y. Un point peut être déplacé dans le plan.**

**Ecrire un algorithme qui demande à l'utilisateur de saisir x et y et une valeur d et déplace le point P.**

# Déclaration (suite)

- **Corrigé**

**Algorithme déplacerPoint**

**Variable d : réel**

**Type Point : structure {**

**x : réel**

**y : réel }**

**Variable P, P1 : Point**

**Début**

**Ecrire("Donner x, y et d")**

**Lire(P.x, P.y, d)**

**P1.x  $\leftarrow$  P.x + d**

**P1.y  $\leftarrow$  P.y + d**

**Fin**

**Algorithme déplacerPoint**

**Variable d : réel**

**Variable P : structure {**

**x : réel**

**y : réel }**

**Début**

**Ecrire("Donner x, y et d")**

**Lire(P.x, P.y, d)**

**P.x  $\leftarrow$  P.x + d**

**P.y  $\leftarrow$  P.y + d**

**Fin**

# Déclarations (suite)

- **Déclarations**



Entête

- Il permet l'identification de l'algorithme
- Mot-clé : Algorithme

Déclarations

- Liste exhaustive des variables, constantes, des structures, des fonctions et des procédures utilisées dans le corps de l'algorithme
- Mots-clés : variable, constante, structure, fonction, ...

Corps

- C'est dans cette partie que les tâches (instructions, opérations, ...) de l'algorithme sont placées
- Mots-clés : Début, Fin

# Déclarations (suite)

- **Déclarations**

La partie déclaration dans un algorithme contient une liste exhaustive des déclarations:

- des variables,
- des constantes;
- des structures,
- des fonctions,
- des procédures,

utilisées dans le corps de l'algorithme

Les variables, constantes, structures, fonctions et procédures utilisées doivent avoir fait l'objet d'une déclaration préalable.

# Fonctions et Procédures

- Analyse descendante

Problème : Comment traiter les problèmes trop complexes pour être appréhendés en un seul bloc ?

Solution : On peut appliquer la méthode "Diviser pour régner", et décomposer le problème en sous-problèmes plus simples : c'est l'analyse descendante

# Fonctions et Procédures

- **Analyse descendante**

Rôle : Méthode pour écrire un algorithme de qualité : lecture plus facile d'un algorithme, modularité du code

Principe :

- Abstraire

Repousser la plus loin possible l'écriture de l'algorithme (codage)

- Décomposer

Décomposer la résolution du problème initial en une suite de “sous problèmes” que l'on considère comme résolus

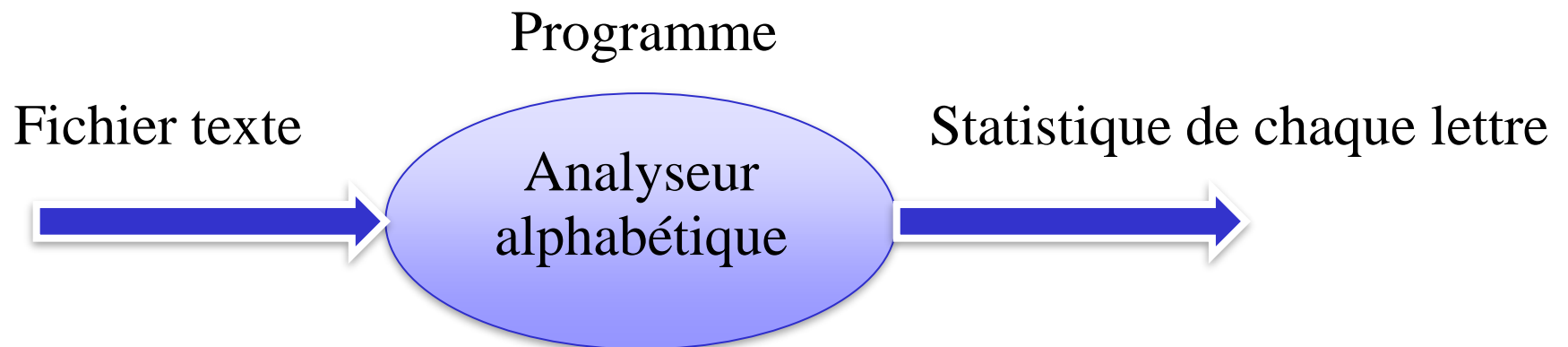
- Combiner

Résoudre le problème initial par combinaison des abstractions des “sous-problèmes”

# Fonctions et Procédures

- Analyse descendante : exemple

Problème d'analyseur alphabétique : Écrire un programme analysant un fichier texte et indiquant le nombre d'occurrence de chaque lettre de l'alphabet dans le fichier



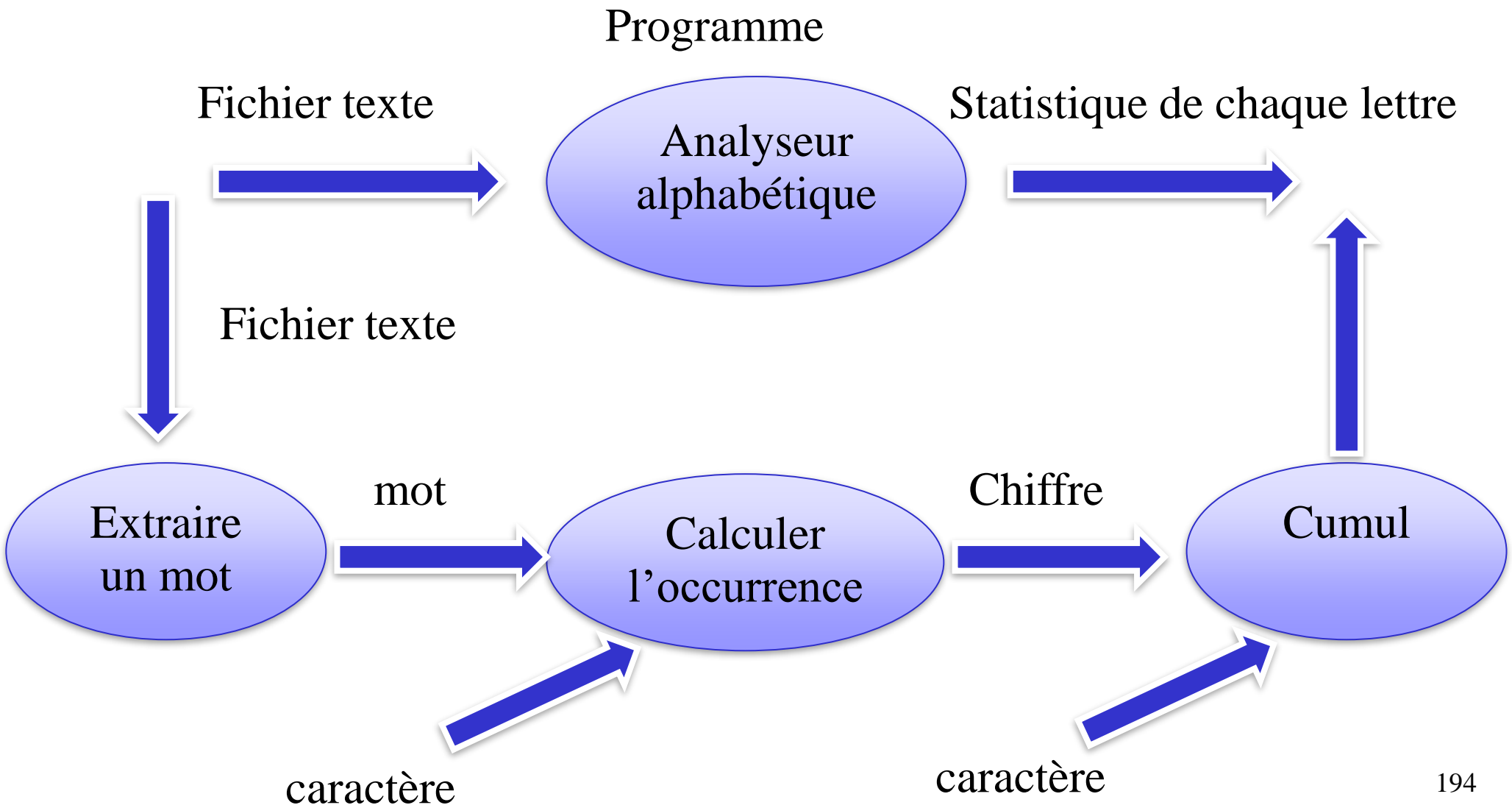


# Fonctions et Procédures

- **Analyse descendante : exemple**
  - Résoudre le problème revient à :
    - Extraire un mot du fichier
    - Pour chaque, lettre de l'alphabet, compter le nombre d'occurrence
    - Calculer le cumul d'occurrence de chaque lettre
    - Répéter la traitement jusqu'au dernier mots du fichier.
  - Chacun de ces sous-problèmes devient un nouveau problème à résoudre
  - Si on considère que l'on sait résoudre ces sous-problèmes, alors on sait “quasiment” résoudre le problème initial

# Fonctions et Procédures

- Analyse descendante : exemple



# Fonctions et Procédures

- Fonctions et procédures

Donc écrire un programme qui résout un problème revient toujours à écrire des sous-programmes qui résolvent des sous parties du problème initial. Il faut comprendre les mots “programme” et de “sous-programme” comme “programme algorithmique” indépendant de toute implantation

En algorithmique il existe deux types de sous-programmes :

- Les fonctions : réalisent des traitements en se servant des valeurs de ce certaines variables et renvoient un résultat. Elles se comportent comme des fonctions mathématiques :  $y=f(x, y, \dots)$
- Les procédures : réalisent seulement des traitements mais ne renvoies aucun résultat

# Fonctions et Procédures

- Analyse descendante : exemple

Algorithme analyseurAlphabetique

Variable fichier, mot : Chaîne

Variable tabOccurrences : Tableau[26] d'entiers

Début

Ecrire ("Donner le nom du fichier à analyser")

Lire (fichier)

Répéter

    mot ← extraireMotSuivant(fichier)

    CompterOccurrence(mot, tabOccurrences)

Tant que mot ≠ " "

Fin

← Appel de Fonction

← Appel de Procédure

# Fonctions et Procédures

- Fonctions et procédures

Les fonctions et les procédures sont des groupes d'instructions indépendants désignés par un nom. Elles ont plusieurs intérêts :

- permettent de "factoriser" les programmes, c-à-d de mettre en commun les parties qui se répètent
- permettent une structuration et une meilleure lisibilité des programmes
- facilitent la maintenance du code (il suffit de modifier une seule fois)

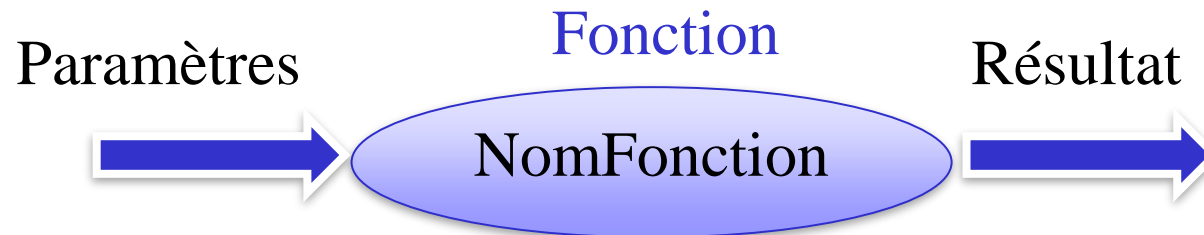
Ces procédures et fonctions peuvent éventuellement être réutilisées dans d'autres programmes

# Fonctions et Procédures

- Fonctions : syntaxe

Une fonction est un bloc d'instructions ayant :

- Un type pour les valeurs qu'elle retourne (type du résultat).  
L'instruction retourne sert à retourner la valeur du résultat
- Un nom (nom de la fonction) : le choix du nom doit respecter les mêmes règles que celles pour les noms de variables
- Une liste de paramètres typés, entre parenthèses (paramètres)



# Fonctions et Procédures

- Fonctions : syntaxe

Syntaxe :

```
Fonction nomFonction (Paramètre1 : type1, ... ) : typeRetour
```

```
    Variable variableLocale : type
```

```
    ....
```

```
Début
```

```
    instructions
```

```
    retourne ...
```

```
Fin Fonction
```

# Fonctions et Procédures

- Fonctions

Fonction ValeurAbsolue (X : Entier) : Entier

Début

Si  $X \geq 0$  alors

retourner X

Sinon

retourner -X

Fin Si

Fin Fonction



# Fonctions et Procédures

- Fonctions

Les paramètres des fonctions sont vus comme des variables locales au bloc de code de la fonction. On peut définir d'autres variables dans le bloc.

Si la fonction n'a besoin d'aucun paramètre on écrit simplement ( )

Fonction ValeurAbsolue (X : Entier) : Entier

Variable res : Entier

Début

Si  $X \geq 0$  alors

res  $\leftarrow$  X

Sinon

res  $\leftarrow$  - X

Fin Si

retourne res

Fin Fonction

# Fonctions et Procédures

- Fonctions

On spécifie la valeur que renvoie une fonction au moyen de l'instruction retourne

Valeur de même type que le type de retour déclaré de la fonction

Fonction ValeurAbsolue (X : Entier) : Entier

Variable res : Entier

Début

Si  $X \geq 0$  alors

res  $\leftarrow$  X

Sinon

res  $\leftarrow$  - X

Fin Si

retourne res

Fin Fonction

# Fonctions et Procédures

- Fonctions

L'instruction retourne  
permet la terminaison  
anticipée de la fonction

Peut exister en plusieurs  
exemplaires

```
Fonction ValeurAbsolue (X : Entier) : Entier
```

```
Début
```

```
    Si  $X \geq 0$  alors
```

```
        retourner X
```

```
    Sinon
```

```
        retourner -X
```

```
    Fin Si
```

```
    Ecrire ("Ce message ne sera jamais  
    afficher »)
```

```
Fin Fonction
```

# Fonctions et Procédures

- **Fonctions**

La fonction  
SommeCarre calcule la  
somme des carrés de  
deux réels x et y

La fonction Pair  
détermine si un nombre  
est pair

Fonction SommeCarre (x : réel, y: réel ) : réel

variable z : réel

Début

$z \leftarrow x^2 + y^2$

retourne z

Fin Fonction

Fonction Pair (n : entier ) : booléen

Début

retourne (n%2=0)

Fin Fonction

# Fonctions et Procédures

- Fonctions

Ecrire une fonction  
permettant de retourner  
le max de deux réels

Ecrire une fonction  
permettant de retourner  
le factoriel d'un entier  
donné



# Fonctions et Procédures

- Appel de Fonctions

On appelle une fonction en donnant son nom, suivi de la valeur des paramètres de l'appel, dans l'ordre de définition. Le nom de fonction avec ses arguments est une expression typée utilisable normalement.

```
Fonction Pair (n : entier) : booléen
Début
    retourne (n%2=0)
Fin Fonction
```

```
Fonction SommeCarre (x : réel, y: réel) : réel
variable z : réel
Début
    z ← x^2+y^2
    retourne z
Fin Fonction
```

Algorithme AppelFonction

variables      z : réel,  
                  b : booléen

Début

```
b ← Pair(3)
z ← 5*SommeCarre(7,2)+1
Ecrire("SommeCarre(3,5):",
SommeCarre(3,5))
```

Fin

paramètre effectif 3

re formel n est remplacé par le

# Fonctions et Procédures

- Appel de Fonctions

Algorithme abs

Variable a : Entier

Début

a ← valeurAbsolue(-3)

Ecrire ("la valeur  
absolue de

-3 est ",a)

Fin

Fonction valeurAbsolue (X : Entier) : Entier

Variable res : Entier

Début

Si  $X \geq 0$  alors

res ← X

Sinon

res ← -X

Fin Sinon

retourne res

Fin Fonction

# Fonctions et Procédures

- Appel de Fonctions

Algorithme abs

Variable a, b : Entier

Début

Ecrire ("Entrez un entier :")

Lire(b)

a  $\leftarrow$  valeurAbsolue(b)

Ecrire ("la valeur absolue  
de", b, "est ",a)

Fin

Fonction valeurAbsolue (X : Entier) : Entier

Variable res : Entier

Début

Si  $X \geq 0$  alors

res  $\leftarrow$  X

Sinon

res  $\leftarrow$  -X

Fin Sinon

retourne res

Fin Fonction



# Fonctions et Procédures

- Appel de Fonctions

Fonction **min2** (a , b : Entier) : Entier

Début

Si  $a \geq b$  alors

retourne b

Sinon

retourne a

Fin Sinon

Fin

Fonction **min3** (a , b, c : Entier) : Entier

Début

retourne **min2(c, min2(a,b))**

Fin

Algorithme min

variable mini : Entier

Début

mini  $\leftarrow$  min3(3,5,8)

Ecrire ("Le minimum est",mini)

Fin

# Fonctions et Procédures

- Appel de Fonctions

Fonction **min2** (a , b : Entier) : Entier

Début

Si  $a \geq b$  alors

retourne b

Sinon

retourne a

Fin Sinon

Fin

Fonction **min3** (a , b, c : Entier) : Entier

Début

retourne min2(c, min2(a,b))

Fin

Algorithme min

variable mini,A,B,C : Entier

Début

Ecrire ("Saisir trois entier")

Lire(A,B,C)

mini  $\leftarrow$  min3(A,B,C)

Ecrire ("Le minimum est",mini)

Fin

# Fonctions et Procédures

- **Procédures**

Dans certains cas, on peut avoir besoin de répéter une tâche dans plusieurs endroits du programme, mais que dans cette tâche on ne calcule pas de résultats ou qu'on calcule plusieurs résultats à la fois

Dans ces cas on ne peut pas utiliser une fonction, on utilise une procédure

Une procédure est un sous-programme semblable à une fonction mais qui ne retourne rien

Une procédure s'écrit en dehors du programme principal sous la forme :

Procédure nom\_procédure (paramètres et leurs types)

Instructions constituant le corps de la procédure

Fin Procédure

Remarque : une procédure peut ne pas avoir de paramètres

# Fonctions et Procédures

- **Procédures**

L'appel d'une procédure, se fait dans le programme principale ou dans une autre procédure par une instruction indiquant le nom de la procédure :

Procédure `exemple_proc (...)`

...

Fin Procédure

Algorithme `exepmleAppelProcédure`

Début

`exemple_proc (...)`

...

Fin

Contrairement à l'appel d'une fonction, on ne peut pas affecter la procédure appelée ou l'utiliser dans une expression. L'appel d'une procédure est une instruction autonome

# Fonctions et Procédures

- Procédures : exemple

Procédure Affichage\_chaine (Y : Chaîne)

Début

Ecrire (Y)

Fin Procédure

Procédure Affichage\_entier (X : Entier)

Début

Ecrire ("La valeur est ",X)

Fin Procédure

Procédure Affichage\_tab (tab : Tableau d'Entiers, taille : Entier)

Variable i : Entier

Début

Pour i allant de 0 à taille -1 Faire

Ecrire ("La ième valeur du tableau est ",tab[i] )

Fin Procédure

# Paramètres des procédures et fonctions

- Paramètres formels et effectifs

Les paramètres servent à échanger des données entre la procédure/fonction appelante et la procédure/fonction appelée

- Vocabulaire

- Un paramètre formel est aussi appelé paramètre
- Un paramètre effectif est aussi appelé argument

- Signification

- Les paramètres placés dans la déclaration d'une fonction/procedure sont des paramètres formels. Ils peuvent prendre toutes les valeurs possibles dans le type déclaré mais ils sont abstraits (n'existent pas réellement)
- Les paramètres placés dans l'appel d'une fonction/procedure sont des paramètres effectifs. Ils contiennent les valeurs pour effectuer le traitement

# Paramètres des procédures et fonctions

- Paramètres formels et effectifs

Un paramètre effectif est une variable ou constante (numérique ou définie par le programmeur)

Le paramètre formel et le paramètre effectif sont associés lors de l'appel de la fonctions. Donc,

- Le nombre de paramètres effectifs doit être égal au nombre de paramètres formels.
- L'ordre et le type des paramètres doivent correspondre

# Paramètres des procédures et fonctions

- Paramètres formels et effectifs

```
Fonction valeurAbsolue (X : Entier) : Entier
Variable res : Entier
Début
    Si  $X \geq 0$  alors
        res  $\leftarrow$  X
    Sinon
        res  $\leftarrow$  -X
    Fin Sinon
    retourne res
Fin Fonction
```

Paramètres formels

Paramètres effectifs

```
Algorithme abs1
Variable a, b : Entier
Début
    Ecrire ("Entrez un entier :")
    Lire(b)
    a  $\leftarrow$  valeurAbsolue(b)
    Ecrire ("la valeur absolue de", b, "est ",a)
Fin
```

```
Algorithme abs2
Variable a : Entier
Début
    a  $\leftarrow$  valeurAbsolue(-3)
    Ecrire ("la valeur absolue de
-3 est ",a)
Fin
```



# Transmission des paramètres

- **Transmission des paramètres dans les procédures**

Il existe deux modes de transmission de paramètres :

- La transmission par valeur : les valeurs des paramètres effectifs sont affectées aux paramètres formels correspondants au moment de l'appel de la procédure. Dans ce mode le paramètre effectif ne subit aucune modification.
- La transmission par adresse (ou par référence) : les adresses des paramètres effectifs sont transmises à la fonction appelante. Dans ce mode, le paramètre effectif subit les mêmes modifications que le paramètre formel lors de l'exécution de la procédure
  - Remarque : le paramètre effectif doit être une variable (et non une valeur) lorsqu'il s'agit d'une transmission par adresse.
  - Lorsque le type des paramètres est un type tableau ou une référence (pointeur), la transmission est par référence

# Transmission des paramètres

- Transmission des paramètres dans les procédures

Procédure incrementer1 (x : entier par valeur, y : entier par adresse)

$x \leftarrow x+1$

$y \leftarrow y+1$

Fin Procédure

Algorithme Test\_incrementer1

variables    n, m : entier

Début

$n \leftarrow 3$

$m \leftarrow 3$

incrementer1(n, m)

écrire (" n= ", n, " et m= ", m)

Fin

résultat :

n=3 et m=4

l'instruction  $x \leftarrow x+1$  n'a pas de sens avec un passage par valeur

# Transmission des paramètres

- **Transmission des paramètres dans les procédures**

- Procédure qui calcule la somme et le produit de deux entiers :

Procédure SomProd (x,y: entier par valeur, som, prod : entier par adresse)

$\text{som} \leftarrow x + y$

$\text{prod} \leftarrow x * y$

Fin Procédure

- Procédure qui échange le contenu de deux variables :

Procédure Echange (x : réel par adresse, y : réel par adresse)

variables    z : réel

$z \leftarrow x$

$x \leftarrow y$

$y \leftarrow z$

# Variables globales et locales

- **Transmission des paramètres dans les procédures**
  - On peut manipuler 2 types de variables dans un module (procédure ou fonction) : des variables locales et des variables globales. Elles se distinguent par ce qu'on appelle leur portée (leur "champ de définition", leur "durée de vie")
  - Une variable locale n'est connue qu'à l'intérieur du module ou elle a été définie. Elle est créée à l'appel du module et détruite à la fin de son exécution
  - Une variable globale est connue par l'ensemble des modules et le programme principale. Elle est définie durant toute l'application et peut être utilisée et modifiée par les différents modules du programme

# Variables globales et locales

- **Transmission des paramètres dans les procédures**
  - La manière de distinguer la déclaration des variables locales et globales diffère selon le langage
  - En général, les variables déclarées à l'intérieur d'une fonction ou procédure sont considérées comme variables locales
  - En pseudo-code, on va adopter cette règle pour les variables locales et on déclarera les variables globales dans le programme principale
  - Conseil : Il faut utiliser autant que possible des variables locales plutôt que des variables globales. Ceci permet d'économiser la mémoire et d'assurer l'indépendance de la procédure ou de la fonction