

# CHAPITRE 2 : PRESENTATION D'UN FORMALISME ALGORITHMIQUE

## 2.1 - NECESSITE D'UN FORMALISME ALGORITHMIQUE



- **Dès qu'un problème est analysé le concepteur devra d'exprimer sa solution dans un formalisme universel, sous la forme d'un algorithme.**

**Le problème est donc d'utiliser un langage commun, afin de comprendre les algorithmes construits par d'autres et vice-versa.**

**D'ou l'intérêt d'un formalisme.**

**Un formalisme algorithmique est un ensemble de conventions (ou de règles) dans lequel on exprime toute solution d'un problème donné.**

langage commun

Communication

Précision - Non ambiguïté

# CHAPITRE 2 : PRESENTATION D'UN FORMALISME ALGORITHMIQUE

## 2.2 - PRESENTATION DU FORMALISE ADOPTE

### 2.2.1 – structure d'un algorithme

Entête →

**ALGORITHME** Nom\_de\_l'algorithme

Environnement →

*Déclarations des objets et Modules utilisés  
dans l'algorithme*

Corps →

**DEBUT**

*manipulation des objets et modules déclarés*

**FIN**

# CHAPITRE 2 : PRESENTATION D'UN FORMALISME ALGORITHMIQUE

## 2.2 - PRESENTATION DU FORMALISE ADOPTE

### 2.2.1 – structure d'un algorithme

### 2.2.2 – Le corps de l'algorithme

**On y définira les outils de base nécessaires pour exprimer tout algorithme. Ils servent à préciser comment doivent s'enchaîner chronologiquement les actions composant un algorithme.**



# CHAPITRE 2 : PRESENTATION D'UN FORMALISME ALGORITHMIQUE

## 2.2 - PRESENTATION DU FORMALISE ADOPTE

### 2.2.1 – structure d'un algorithme

### 2.2.2 – Le corps de l'algorithme

### 2.2.2.1 – Les structures de contrôle

# **1. L'enchaînement ( séquencement ) : Les actions primitives sont exécutées dans l'ordre d'apparition dans l'algorithme.**

Action 1

Action 2

.

.

.

.

Action n

—

—

—

## 2 . La conditionnelle

**SI** *expression logique* **ALORS**  
**DSI**  
*Bloc*  
**FSI**

—  
 —

Si la condition est vérifiée on exécute le bloc, dans le cas contraire on va en séquence.

### 3 . L 'alternative

**SI** *expression logique* **ALORS**

**DSI**

*Bloc1*

**FSI**

**SINON**

**DSIN**

*Bloc2*

**FSIN**

—

Si la condition est vérifiée alors le bloc1 est exécuté, si elle n'est pas vérifiée c'est le bloc2 qui est exécuté.

**REMARQUE :** La conditionnelle est un cas particulier de l'alternative

**NOTE IMPORTANTE** :un bloc est un ensemble cohérent composé de 1 ou plusieurs actions . Un bloc commence par un début de bloc (ici DSI) et se termine par une fin de bloc (ici FSI).

De plus, si un bloc est composé d'une seule action primitive, les début et fin de bloc sont facultatifs.

**BLOC**

**DEBUT**

**Action 1**

**Action 2**

•

•

•

•

**Action n**

**FIN**

## La conditionnelle : EXEMPLE

Lire ( $a, b$ )

SI  $a > b$  ALORS

DSI

$a \leftarrow a + 10$

FSI

$b \leftarrow b - 5$

$a \leftarrow a + b$

SI  $b = 0$  ALORS

DSI

$c \leftarrow b - 10$

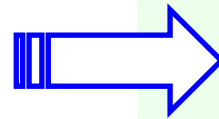
$a \leftarrow a - b$

$b \leftarrow c + a$

FSI

$d \leftarrow a + c$

Ecrire ( $a, b, c, d$ )



Lire ( $a, b$ )

SI  $a > b$  ALORS  $a \leftarrow a + 10$

$b \leftarrow b - 5$

$a \leftarrow a + b$

SI  $b = 0$  ALORS

DSI

$c \leftarrow b - 10$

$a \leftarrow a - b$

$b \leftarrow c + a$

FSI

$d \leftarrow a + c$

Ecrire ( $a, b, c, d$ )

## L'alternative : EXEMPLE

**LIRE** (a,b,c)

$\Delta \leftarrow b*b - 4*a*c$

**SI**  $\Delta > 0$  **ALORS**

**DSI**

$x1 \leftarrow (-b - \text{Sqrt}(\Delta) / (2*a))$

$x2 \leftarrow (-b + \text{Sqrt}(\Delta) / (2*a))$

**ECRIRE** (x1, x2)

**FSI**

**SINON**

**SI**  $\Delta = 0$  **ALORS**

**DSI**

$x1 \leftarrow -b / 2 * a$

**ECRIRE** (x1)

**FSI**

**SINON** **ECRIRE** ( 'pas de racines' )

## 4 . La répétitive

Il arrive souvent que l'on ait besoin de répéter une action ou un ensemble d'actions (bloc) plusieurs fois. Pour ce faire on utilise une structure répétitive

**Il existe 3 formes de structures répétitives :**

1. La forme **POUR**
2. La forme **TANT QUE**
3. La forme **REPETER**



## - La forme POUR

**POUR** *variable\_de\_contrôle* **ALLANT DE** *valeur\_initiale* **A** *valeur\_finale* **FAIRE**

**DPOUR**

bloc

**FPOUR**

**FONCTIONNEMENT** : Pour chaque valeur de la variable de contrôle qui varie de la valeur initiale à la valeur finale avec un pas égal à 1 le bloc sera exécuté. Chaque exécution du bloc est appelée **itération** (ou **boucle**).

**Dans la forme POUR il y a :**

- 1. Une initialisation de la variable de contrôle**
- 2. Une incrémentation à chaque itération**
- 3. Un test de fin**

**REMARQUE :** La forme POUR est bien adaptée aux cas où l'on peut prévoir le nombre d'exécutions du bloc, c'est à dire le nombre d'itérations, et où il est donc possible de déterminer les valeurs initiales et finales de la variable de contrôle.

## Exemple 1

```

a ← 2
POUR i ALLANT DE 1 A 5 FAIRE
    DPOUR
    | K ← a * i
    | A ← k
    FPOUR

```

—  
—

## Exemple 2 : Structures Pour imbriquées

```

x ← 10
POUR cpt ALLANT DE x A x + 2 FAIRE
    DPOUR
    | a ← 2
    | b ← a
    | POUR I ALLANT DE 1 A 3 FAIRE ECRIRE ( i * cpt )
    | FPOUR

```

—

## - La forme TANT QUE

**TANT QUE** *condition* **FAIRE**

**DTQ**

*Bloc*

**FTQ**

**FONCTIONNEMENT :** Dans la forme TANT QUE, si la condition est vérifiée le bloc est exécuté, puis on remonte en début de bloc, et si la condition est à nouveau vérifiée le bloc est à nouveau exécuté. La répétition s'arrête lorsque la condition n'est plus vérifiée.

Exemple :

**LIRE** (x,y)

**TANT QUE**  $x > y$  **FAIRE**

**DTQ**

$a \leftarrow x$

$b \leftarrow y$

$c \leftarrow a - b$

$x \leftarrow c + 1$

**FTQ**

**ECRIRE** (x,y,a,b)

## - La forme REPETER

### **REPETER**

*Bloc*

**JUSQU'A** *condition*

**FONCTIONNEMENT** : C'est une variante de la forme TANT QUE, il s'agit de répéter l'exécution d'un bloc jusqu'à ce que la condition soit vérifiée. Donc dès que la condition est vérifiée il y a arrêt des répétitions et sortie de la boucle.

**REMARQUE** : c'est l'un des rares cas où le bloc n'est pas délimité par un DEBUT et FIN, car il l'est en fait par les mots REPETER et JUSQU'A

## Exemple

$A \leftarrow 5$

**REPETER**

$X \leftarrow a * 2$

$Y \leftarrow 100 - x$

$A \leftarrow a - 1$

**JUSQU'A**  $a = 0$

**ECRIRE** (x, y)

Les formes TANT QUE et REPETER sont utilisées lorsque le nombre de boucles (itérations) est inconnu. Cependant il est au moins égal à **1** pour le REPETER alors qu'il peut être égal à **0** dans la forme TANT QUE. Cette différence permet souvent de choisir entre ces 2 formes.

Attention aux erreurs de conception qui peuvent conduire à un "bouclage infini", la condition restant toujours vérifiée.



# CHAPITRE 2 : PRESENTATION D'UN FORMALISME ALGORITHMIQUE

## 2.2 - PRESENTATION DU FORMALISE ADOPTE

### 2.2.2 – Le corps de l'algorithme

#### 2.2.2.1 – Les structures de contrôle

#### 2.2.2.2 – autres actions de base

##### 2.2.2.2.1 – l'affectation

## FORMAT :

**Variable ← expression**

L'affectation est symbolisée par ← . Le rôle de l'affectation consiste à affecter (donner, attribuer) une valeur à un objet. La valeur pouvant être une constante, la valeur d'un autre objet ou le résultat de l'évaluation d'une expression.

## Fonctionnement :

Dans l'affectation il faudra donc évaluer, éventuellement, l'entité se trouvant à droite du symbole d'affectation et ensuite mettre ce résultat dans l'entité se trouvant à gauche du symbole d'affectation.

## exemples

$X \leftarrow 0$

*mettre la valeur zéro dans X.*

$X \leftarrow Y$

*mettre la valeur de l'objet Y dans X .*

$X \leftarrow X + 1$

*ajouter 1 à X*

$X \leftarrow X - Y + Z$

*mettre dans X le résultat de l'évaluation  
de l'expression  $X - Y + Z$*

# CHAPITRE 2 : PRESENTATION D'UN FORMALISME ALGORITHMIQUE

## 2.2 - PRESENTATION DU FORMALISE ADOPTE

### 2.2.2 – Le corps de l'algorithme

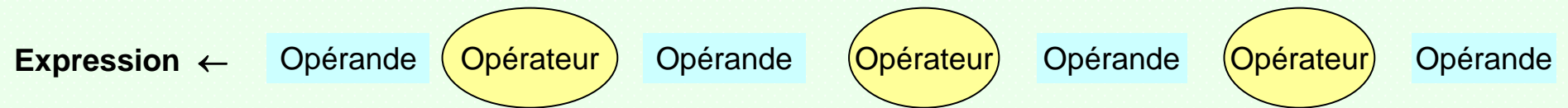
#### 2.2.2.1 – Les structures de contrôle

#### 2.2.2.2 – autres actions de base

##### 2.2.2.2.1 – l'affectation

##### 2.2.2.2.2 – les expressions (arithmétiques, logiques , relationnelles et mixtes)

Les expressions sont de la forme :



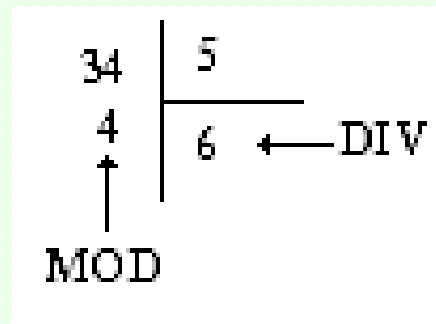


## LES EXPRESSIONS ARITHMÉTIQUES :

- Opérandes : entiers, réels
- Opérateurs : +, -, /, \*, DIV, MOD

/ : Slash , \* : astérisque

**DIV**: division entière. Division avec troncature de la partie décimale.



Exemple :

$X \leftarrow 34 \text{ DIV } 5$

consiste à mettre 6 dans X, en effet la division de 34 par 5 donne 6.8 et on ne prendra que la partie entière de ce résultat.

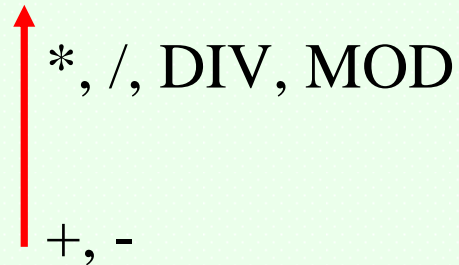
**MOD** : modulo. Donne le reste d'une division.

Exemple :

$X \leftarrow 34 \text{ MOD } 5$

X contiendra 4 car le reste de la division de 34 par 5 donne 6 et un reste égal à 4.

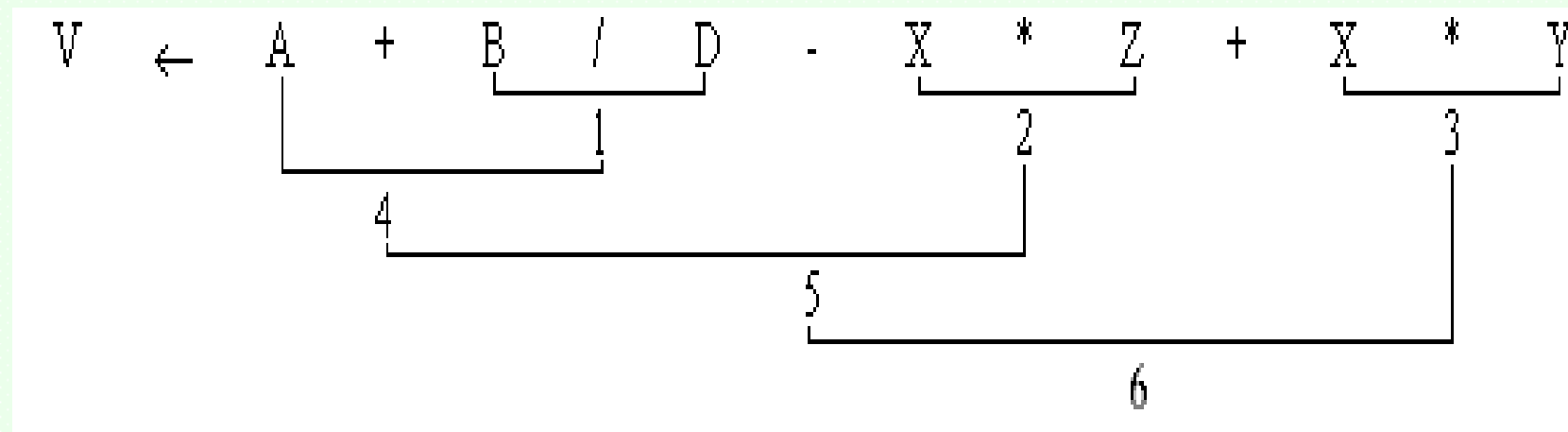
## Hiérarchie des opérateurs:



La hiérarchie des opérateurs nous permet de savoir comment sera évaluée une expression. En effet, on commencera par les opérateurs dont la hiérarchie est la plus supérieure et ensuite on passera à ceux dont la hiérarchie est immédiatement inférieure. Dans la cas d'une expression arithmétiques on commencera par faire, indistinctement, toutes les multiplications , les divisions, les divisions entières et les modulus et ensuite on passera aux additions et aux soustractions.

Quand la hiérarchie est la même l'expression est évaluée de la gauche vers la droite.

- Exemple :







- le tableau suivant donne le type qu'il faudra prendre pour le résultat d'une expression arithmétique en fonction des opérateurs et des types des opérandes. Il devra être respecté au risque d'avoir des erreurs lors de l'exécution des programmes.

Opérateurs	Types des opérandes	Type du résultat
+      -      *	E op E	<b>E (entier)</b>
	R op R	<b>R (réel)</b>
	R op E	<b>R</b>
	E op R	<b>R</b>
/	E op E	<b>R</b>
	R op R	<b>R</b>
	R op E	<b>R</b>
	E op R	<b>R</b>
DIV , MOD	E op E	<b>E</b>

## LES EXPRESSIONS LOGIQUES

- Opérandes : Vrai , Faux
- Opérateurs : ET , OU , NON

### Hiérarchie des opérateurs :

↑  
NON  
ET  
OU

Nota : pratiquement tous les opérateurs sont binaires , car ils exigent 2 opérandes, à l'exception de l'opérateur logique NON qui est unaire car il exige un seul opérande.

En effet, pour exprimer le complément de A, on écrira : NON A

Exemple : a OU c ET b ET NON a

## LES EXPRESSIONS RELATIONNELLES

- Opérandes : Numériques , Alphanumériques
- Opérateurs : = , > , >= , < , <= , <>


Les opérateurs ont la même hiérarchie

Exemple :      a <> b  
                     B <= x

## LES EXPRESSIONS MIXTES

- Opérandes : quelconques
- Opérateurs : quelconques

### Hiérarchie des opérateurs :


 NON  
 \* , / , DIV , MOD , ET  
 + , - , OU  
 = , > , >= , < , <= , <>

**Exemple :**

**a \* b OU a DIV c ET x OU NON a MOD b + c >= r**

## L'UTILISATION DES PARENTHÈSES

- Des expressions complexes nécessitent l'utilisation des parenthèses, car il faut les exprimer sous une forme linéaire (sur une même ligne). Les expressions entre parenthèses sont évaluées en premier et en commençant par les parenthèses les plus intérieures.

NOTA: tant que l'expression reste valable il est possible d'utiliser un nombre excessif de parenthèses.

**Exemple 1:**

$$Cr = \frac{L.B.F}{\frac{F.B + n}{d} + e}$$

**S'écrit :**

$$Cr = (L * B * F) / (((F * B) + n) / d + 2)$$

**ou :**

$$CR = L * B * F / ((F * B + n) / d + 2)$$

**Exemple 2:**

$$res = \frac{a + \frac{b}{c}}{x + \frac{v + w}{y}}$$

**S'écrit :**

$$res = (a + b / c) / (x + (v + w) / y)$$

**Exemple 3:**

$$v = \frac{a + b}{c - d}$$

**S'écrit :**

$$v = (a + b) / (c - d)$$





## LA LECTURE

### **LIRE ( f, p1 , p2 , ... , pn)**

Elle permet de fournir des valeurs venant de l'extérieur à des variables de l'algorithme, car il arrive souvent qu'un objet ne varie pas durant l'exécution de l'algorithme, cependant l'utilisateur peut lui changer de valeur entre 2 exécutions de ce même algorithme. Cet objet est appelé paramètre et l'utilisation des paramètres permet de généraliser les algorithmes.

Nota: f indique le nom logique du fichier d'entrée. Nous verrons plus tard cette notion. Par défaut, il s'agit du clavier.

### Fonctionnement :

Les valeurs lues au clavier sont affectées respectivement aux variables avec prise en compte de la compatibilité des types

#### Exemples:

LIRE (N)

LIRE ( a,b,c)

## L'ECRITURE

**ECRIRE** (  $f$ ,  $p_1$ ,  $p_2$  , ... ,  $p_n$ )

Elle permet de restituer les résultats d'un algorithme.

Nota:  $f$  indique le nom logique du fichier de sortie. Nous verrons plus tard cette notion. Par défaut, il s'agit de l'écran.

Ou  $p$  peut être :

- Une variable
- Un libellé : chaîne de caractères entre apostrophes
- Une expression

Fonctionnement : les expressions sont évaluées et les résultats sont écrits ou affichés)

# Exemple :

**ECRIRE (a)**

**ECRIRE ('les résultats sont :')**

**ECRIRE(  $b*b-4*a*c$ )**

**ECRIRE (' les résultats de l'équation sont ', X1, ' et ',  $(-b-\sqrt{\Delta})/2*a$ )**

# CHAPITRE 2 : PRESENTATION D'UN FORMALISME ALGORITHMIQUE

## 2.2 - PRESENTATION DU FORMALISE ADOPTE

### 2.2.2 – Le corps de l’algorithme

#### 2.2.2.1 – Les structures de contrôle

##### 2.2.2.2 – autres actions de base

##### 2.2.2.2.1 – l’affectation

##### 2.2.2.2.2 – les expressions (arithmétiques, logiques , relationnelles et mixtes)

### 2.2.3 – L’environnement – Objets élémentaires

Tous les constituants de l'univers d'un programme doivent être décrits dans l'environnement d'un algorithme.

# CHAPITRE 2 : PRESENTATION D'UN FORMALISME ALGORITHMIQUE

## 2.2 - PRESENTATION DU FORMALISE ADOPTE

### 2.2.2 – Le corps de l'algorithme

#### 2.2.2.1 – Les structures de contrôle

##### 2.2.2.2 – autres actions de base

###### 2.2.2.2.1 – l'affectation

###### 2.2.2.2.2 – les expressions (arithmétiques, logiques , relationnelles et mixtes)

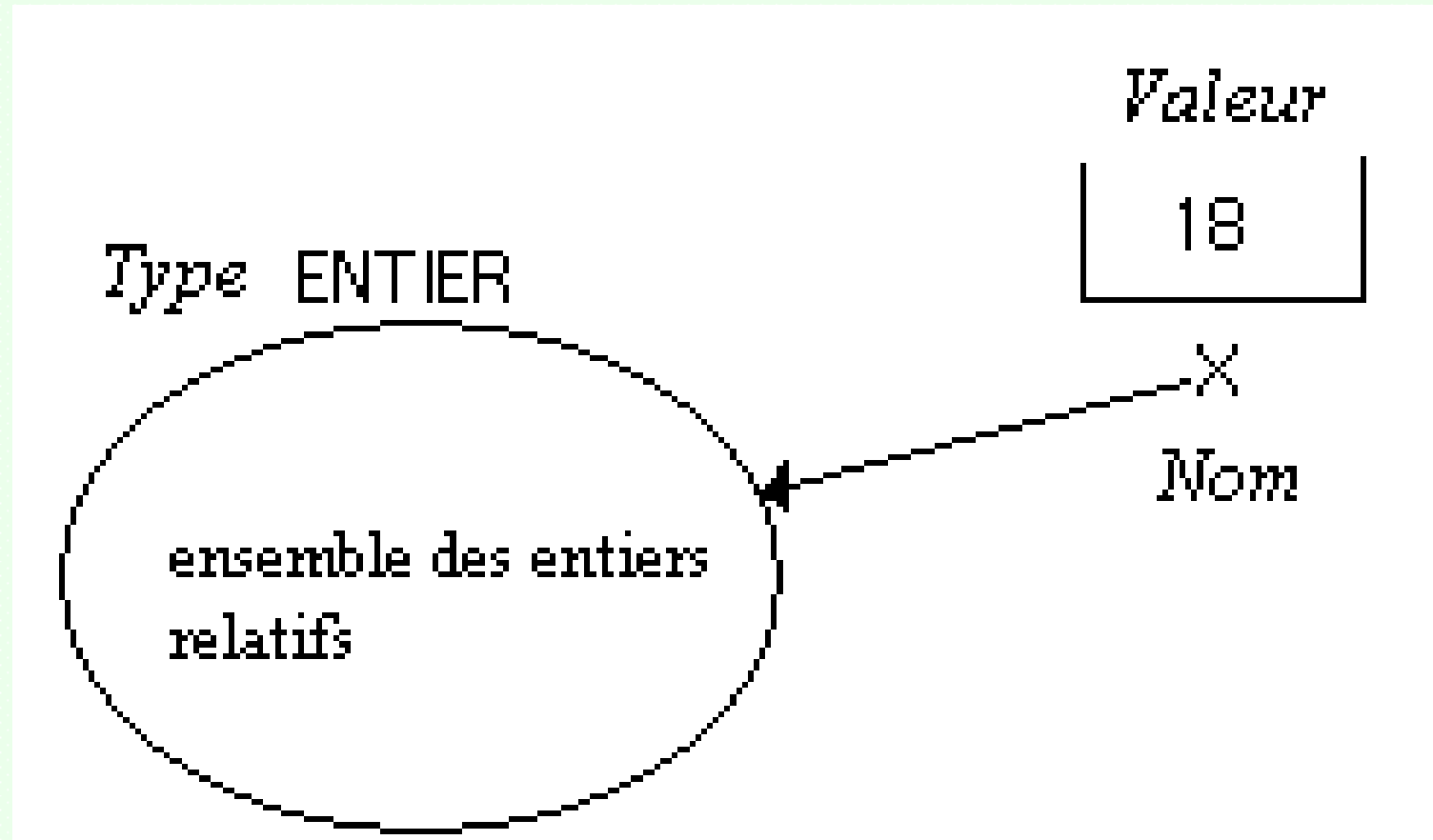
### 2.2.3 – L'environnement – Objets élémentaires

#### 2.2.3.1 – Objets dans un environnement

Tout algorithme utilise des objets qui seront déclarés dans son environnement.

**A chaque objet il faudra faire correspondre :**

- Un **NOM** qui permettra de le désigner et de le distinguer des autres éléments,
- Un **TYPE** qui indique la nature de l'ensemble dans lequel l'objet prend ses valeurs,
- Une **VALEUR** affectée à cet objet à un moment donné.





# CHAPITRE 2 : PRESENTATION D'UN FORMALISME ALGORITHMIQUE

## 2.2 - PRESENTATION DU FORMALISE ADOPTE

### 2.2.2 – Le corps de l'algorithme

#### 2.2.2.1 – Les structures de contrôle

##### 2.2.2.2 – autres actions de base

##### 2.2.2.2.1 – l'affectation

##### 2.2.2.2.2 – les expressions (arithmétiques, logiques , relationnelles et mixtes)

### 2.2.3 – L'environnement – Objets élémentaires

#### 2.2.3.1 – Objets dans un environnement

#### 2.2.3.2 – Les déclarations

**ALGORITHME** nom\_algorithme

Déclarations des étiquettes

Déclarations des constantes \*

Déclarations des types \*

Déclarations des variables \*

Déclarations des sous-programmes (Modules)

Nous ne traiterons  
que les  
déclarations

Un objet est dit  
élémentaire s'il n'est  
pas décomposable.

**DEBUT**

corps de l'algorithme

**FIN**

# Pourquoi il ne faut **JAMAIS** utiliser les étiquettes ?

Début

10    action

20    **GO TO Jsk**

Fx    action

40    **GO TO alilou**

Zozo   action

60    **GO TO allo**

Alilou   action

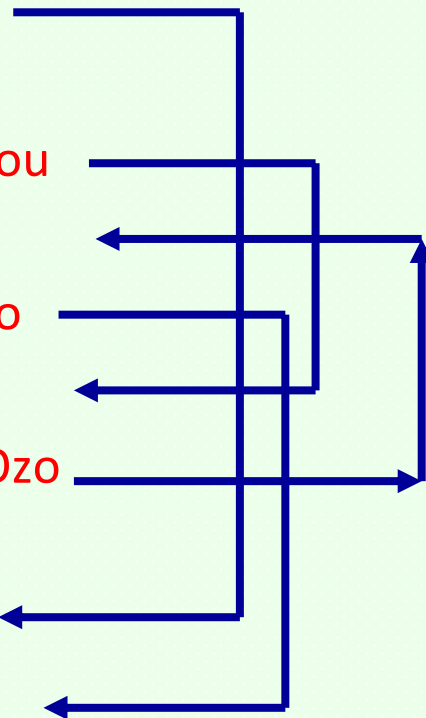
Aab1   **GO TO ZOzo**

70    action

Jsk    **GO TO fx**

Allo    action

Fin





Les étiquettes sont les obstacles N°1 à la programmation structurée.

Voila pourquoi il ne faut **JAMAIS** les utiliser !!

# CHAPITRE 2 : PRESENTATION D'UN FORMALISME ALGORITHMIQUE

## 2.2 - PRESENTATION DU FORMALISE ADOPTE

### 2.2.2 – Le corps de l'algorithme

#### 2.2.2.1 – Les structures de contrôle

##### 2.2.2.2 – autres actions de base

##### 2.2.2.2.1 – l'affectation

##### 2.2.2.2.2 – les expressions (arithmétiques, logiques , relationnelles et mixtes)

### 2.2.3 – L'environnement – Objets élémentaires

#### 2.2.3.1 – Objets dans un environnement

#### 2.2.3.2 – Les déclarations

#### 2.3.3.3 – Les constantes



## DÉCLARATION DE CONSTANTES

**CONSTANTE**    *Identificateur\_constante = Valeur*

Une constante est un objet élémentaire particulier dont la valeur est invariable durant l'exécution de l'algorithme.

**Exemples :**

**CONSTANTE**    **Pi = 3.14**

**CONSTANTE**    **Titre = 'Résultats'**

**CONSTANTE**    **cent = 100**

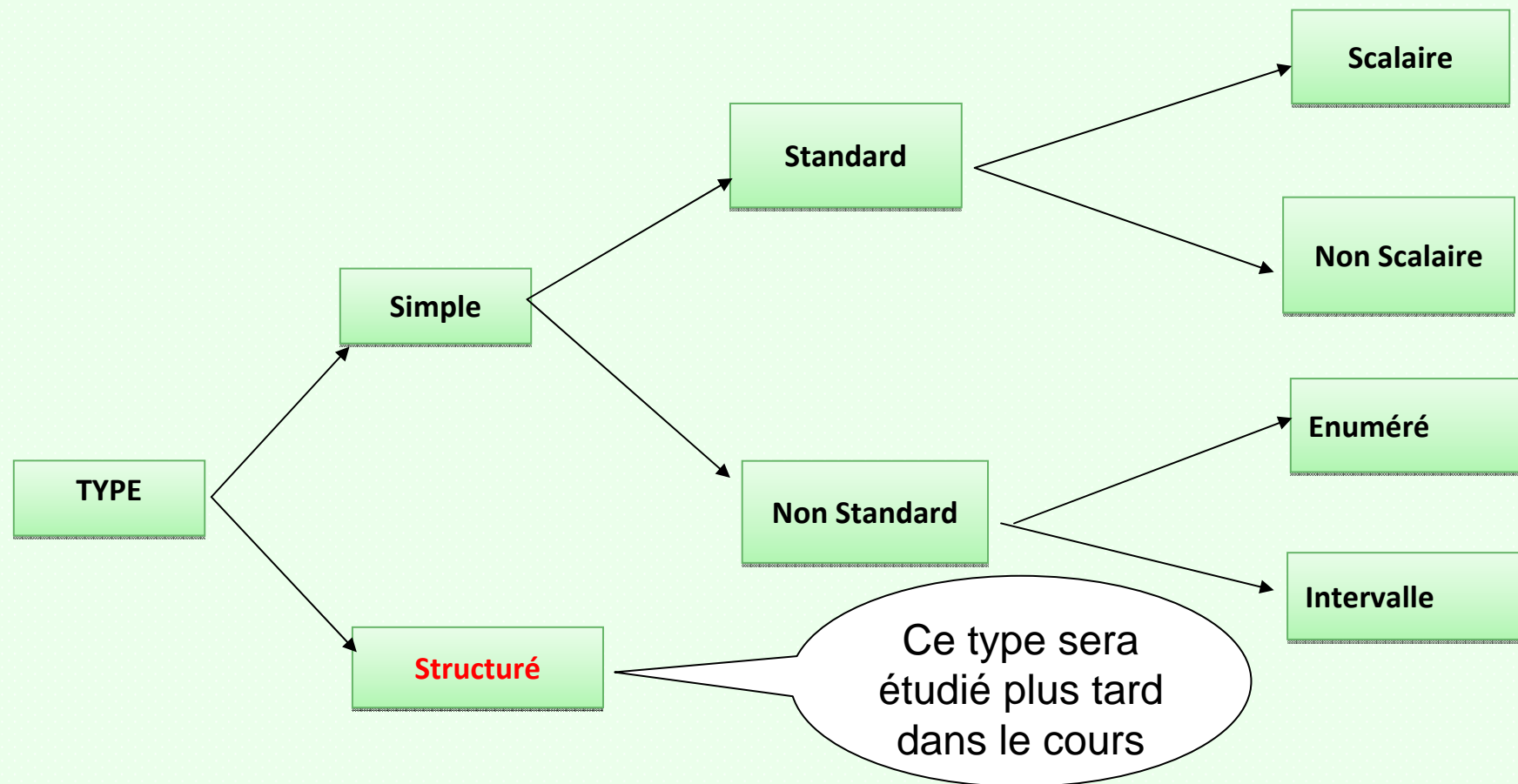
**CONSTANTE**    **virgule = ','**

**Nota** : dans le cas où nous avons plusieurs constantes à déclarer, le mot **Constante** pourra être écrit qu'une seule fois. Par exemple les déclarations précédentes peuvent s'écrire :

**CONSTANTE**    **Pi = 3.14**  
                          **titre = ' Résultats'**  
                          **cent = 100**  
                          **virgule = ' , '**

# DÉCLARATION DE TYPES SIMPLES

Un type décrit l'ensemble des **valeurs** que peut prendre un objet qui y est défini ainsi que les **opérations autorisées** sur cet objet



## Il existe 4 types standards:

### Scalaire

- Le type entier
- Le type booléen
- Le type caractère

### Non scalaire

- Le type réel



\* **Le type entier** : C'est l'ensemble des entiers relatifs, cependant il faudra signaler que si en mathématiques cet ensemble est infini, sur un ordinateur les valeurs sont limitées par la longueur des mots machines.

\* **Le type réel** : C'est l'ensemble des nombres ayant une partie fractionnelle. Cet ensemble est aussi limité, mais les limites sont plus larges et dépendent de la représentation interne ( Cf. cours de structure machines).

\* **Le type booléen (ou logique)** : C'est l'ensemble des valeurs { Vrai , Faux }

\* **Le type caractère** : Il correspond à un seul caractère . Suivant les systèmes le jeu de caractères peut varier, il comprend l'ensemble des caractères alphanumériques ( alphabétiques et numériques, signes spéciaux et caractère blanc ( ou espace)).

**NOTA** : pour connaître les contenus de ces différents ensembles utilisés par le langage Pascal , veuillez vous rapporter à la notice sur le langage Pascal, qui vous sera remise !

- NOTE IMPORTANTE : Les types standards ou prédéfinis n'ont pas besoin d'être définis par une déclaration.

Il arrive de définir de types spécifiques à notre problème par le biais de types non-standards, énumérés ou intervalles.

**Il existe deux Types non standards :**

- le type **énuméré**
- le type **intervalle**

**TYPE** *nom\_du\_type* = ( *élément1*, *élément2*, ....., *élémentn* )

Le type énuméré définit un ensemble ordonné de valeurs désignées par des identificateurs (de constantes) (256 au maximum)

Exemple :

**TYPE**

**jours = (dim, lun, mar, mer, jeu, ven, sam)**

**Couleur = (Violet, Indigo, bleu, vert, jaune, rouge)**

**taille = ( grand, moyen, petit)**

**reponse = ( oui, non)**

**fete = ( aïd , mouloud , achoura , moharram )**

**annee = (sept, oct, nov, dec, jan, fev, mar, avr, mai, jui, jul, aout)**

**TYPE** *nom\_du\_type* = *constante1* .. *constante2*

Ce type définit un intervalle d'un ensemble de valeurs ordonnées déjà défini ou prédéfinis par un type ordinal par l'indication de bornes inférieures et supérieures de l'intervalle

Le type des constantes qui sont les bornes de l'intervalle précise quel est le type ordinal de base dont est issu l'intervalle

Dans le type intervalle , les valeurs sont scalaires et Constant1 doit être supérieure à constante2 .

Exemples :

Type

indice = 1 .. 10

chiffre = '0' .. '9'

lettre\_Maj = 'A' .. 'Z'

lettre\_Min = 'a' .. 'z'

**Si on a les déclarations suivantes :**

**Type**

**fete = ( aid , mouloud , achoura , moharram)**

**mois = (sept,oct,nov,dec,jan,fev,mar,avr,mai,jui,jul,aout)**

**Variables**

**m : mois**

**f : fete**

**Nous pouvons très bien écrire dans notre algorithme :**

**Si (m >= mar ) ET (m <= jui) alors écrire ( ‘Printemps’)**

**Pour i allant de aid à moharram faire**

**congé ← congé + 1**

**Il est possible de déduire un type non à partir d'un autre type.**

**Exemple :**

**Type**

**Arc-en-ciel = (Violet , Indigo , bleu , vert , jaune , rouge)**

**annee = (sept,oct,nov,dec,jan,fev,mar,avr,mai,jui,jul,aout)**

**couleur = violet .. Vert**

**An\_sco = oct..jui**

**Attention , on ne peut pas lire ,ni écrire des objets de types non standards.**

# DÉCLARATION DE VARIABLES

**VARIABLE** *nom\_de\_la\_variable : type*

Variable	lettre : caractère
	l, m, n, o, p : reel
	x , y , z : entier
	Rep : booleen



## **ALGORITHME      exemple**

### **Constantes**

**Pi = 3.14**

**Cent = 100**

### **Types**

**reponse = ( oui, non)**

**fete = ( aïd , mouloud , achoura , moharram )**

**mois = (sept,oct,nov,dec,jan,fev,mar,avr,mai,jui,jul,aout)**

**numois = 1 .. 12**

**temp = -15 .. 60**

**lettre = 'a'..'f'**

### **Variables**

**a, b, h,i : entier**

**Res, tot : reel**

**Nm : numois**

**Conge : fete**

**R : reponse**

**Ecart : temp**

**Code1, code2 : lettre**

**M1,M2,M3 : mois**

DEBUT

FIN

**Soyez comme les canards:**

En surface

**AYEZ L'AIR CALME ET POSÉ**



**FIN DU CHAPITRE**



Sous la surface

**PÉDALEZ COMME UN FOU**