# Cours
# Introduction à l'algorithmique

# INTRODUCTION ALGORITHME

## I. Notion d'Algorithme:

Du mathématicien persan Al-Khawa-Rizm (Bagdad, 780 – 850)
Pour les notions de Al-Jabr (Algèbre) théorie du calcul

**Selon le LAROUSSE, la définition d'algorithme est « un ensemble de règles opératoires dont l'enchaînement permet de résoudre un problème au moyen d'un nombre fini d'opérations. »**

***Quelques points importants :***

- Un algorithme décrit un traitement sur un ensemble fini de données de nature simple (nombres ou caractères), ou plus complexes (données structurées)

- Un algorithme est constitué d'un ensemble fini d'actions composées d'opérations ou actions élémentaires. Ces actions élémentaires doivent être effectives (réalisable par la machine), non ambiguës.

- Un algorithme doit toujours se terminer après un nombre fini d'opérations.

- L'expression d'un algorithme nécessite un langage clair (compréhension) structuré (enchaînements d'opérations) non ambiguë, universel (indépendants du langage de programmation choisi)

*Problème : un tel langage n'existe pas, on définit son propre langage.*

# INTRODUCTION ALGORITHME

## II. Méthodologie de conception d'un algorithme :

**Analyse descendante** : (ou programmation structurées) : on décompose un problème complexe en sous problèmes et ces sous problèmes en d'autres sous problèmes jusqu'à obtenir des problèmes faciles a résoudre.

On résout les sous problèmes simples sous forme d'algorithme puis on recompose les algorithmes pour obtenir l'algorithme global du problème de départ.

**Garder à l'esprit :**

- **La modularité** : un module résout un petit problème donné. Un module doit être réutilisable.

- Lisibilité de l'algorithme (mise en page, commentaires, spécification : dire quoi mais pas comment)

- Attention à la complexité de l'algorithme :

  - *Complexité en temps* : mesure du temps d'exécution en fonction de la taille des données
  - *Complexité en espace* : espace mémoire nécessaire pour effectuer les traitements.

- Ne pas réinventer la roue (c'est-à-dire ne pas refaire les programmes standard dont les solutions sont connues) ce qui implique avoir une certaine culture et un outil technique standard.

# Structure d'un algorithme

ALGORITHME <*NOM*>
<*Déclaration des variables*>
DEBUT
<*Actions*>
FIN

Un algorithme est constitué:

- d'un ***entête*** composé du MOT Réservé ALGORITHME et d'un nom de l'algorithme à réaliser
- d'une ***zone de déclaration*** des identificateurs (variables) utilisés dans l'algorithme
- et d'un ***corps délimité*** par deux mots réservés DEBUT et FIN. C'est ici qu'on écrit les actions de l'algorithme

```
ALGORITHME addition
VAR a,b,c :ENTIER
DEBUT
LIRE(a,b)
c←a+b
ECRIRE(c)
FIN
```

# ELEMENT DE BASE D'ALGORITHME

## I. Notion d'objet :

Un algorithme ou une action manipule des données pour obtenir un résultat.
Pour cela on manipule des objets simples ou structurés.

Un objet va être caractérisé par :

- un identificateur (son nom) : pour le désigner cet identificateur doit être parlant : **q**=quotient,**R**=Reste,**Moy**=Moyenne,**ADR**=Adresse….

- Un type (nature de l'objet : entier, caractère…) simple ou structuré. Un type détermine en particulier les valeurs possibles de l'objet et les opérations primitives applicable à l'objet.
R: *ENTIER*
CAR: *CARACTERE ; ADR: CHAINE*

- Une valeur (contenu de l'objet) unique. Cette valeur peut varier au cours de l'algorithme ou d'une exécution à l'autre : ces objets sont des variables.

Dans les cas contraires (valeur fixe) ce sont des constantes. Tous les objets manipulés par un algorithme doivent être clairement définis :
Mots clefs

```
CONST      : PI=3.14
VAR  a, b  : ENTIER
x, y       : CARACTERE
```

a, b, x, y sont des identificateurs

# Notion d'objet

*1. Constantes et variables*

*2. Type logique*

*3. Type CARACTERE*

*4. Type CHAINE DE CARACTERE*

*5. Type énumérés*

*6. Type intervalles*

*7.Type structurés*

*8. Type pointeur*

***Une constante*** est un objet dont l'état reste inchangé durant toute l'exécution d'un programme

```
CONST   PI=3.14
  NOM="PASCAL"
```

***Une variable*** est un objet dont le contenu peut être modifié par une action

**ENTIER:** Pour représenter les nombres entiers

**REEL :** Pour représenter les nombres réels.

**Exemples:**

```
VAR  Classement : ENTIER
       Moyenne    : REEL
```

# Notion d'objet

1. *Constantes et variables*
2. ***Type logique***
3. *Type CARACTERE*
4. *Type CHAINE DE CARACTERE*
5. *Type énumérés*
6. *Type intervalles*
7. *Type structurés*
8. *Type pointeur*

Une variable de ***type logique*** (booléen) peut prendre deux valeurs VRAIE ou FAUSSE.

**VAR** EXISTE :BOOLEEN
EXISTE← VRAIE

Les opérations principales les plus utilisées sont :

- ***Les opérateurs logiques***: NON, ET , OU
- ***Opérateurs de comparaison*** : = , ≤ , ≥ , ≠

VAR A , B , TROUVE :BOOLEEN
**TROUVE** ← (A **ET** B)

VAR X,Y: REEL
SUP :BOOLEEN
SUP ← ( X > Y)

# Notion d'objet

1. *Constantes et variables*
2. *Type logique*
3. ***Type CARACTERE***
4. *Type CHAINE DE CARACTERE*
5. *Type énumérés*
6. *Type intervalles*
7. *Type structurés*
8. *Type pointeur*

Il s'agit du domaine constitué des c***aractères alphabétiques*** et ***numériques***

**VAR** C : CARACTERE
C ← 'A'

# Notion d'objet

1. *Constantes et variables*
2. *Type logique*
3. *Type CARACTERE*
4. ***Type CHAINE DE CARACTERE***
5. *Type énumérés*
6. *Type intervalles*
7. *Type structurés*
8. *Type pointeur*

Une chaine de caractère est un objet qui peut contenir ***plusieurs caractères*** de manière ordonnée

**VAR** NOM : **CHAINE[30]** → Chaine de 30 caractères ('maximum')
ADRESSE : **CHAINE** → Chaine de 128 caractères maximum

# Notion d'objet

1. *Constantes et variables*
2. *Type logique*
3. *Type CARACTERE*
4. *Type CHAINE DE CARACTERE*
5. ***Type énumérés***
6. *Type intervalles*
7. *Type structurés*
8. *Type pointeur*

Un ***type énuméré*** est un type permettant de représenter des objets pouvant prendre leur valeur dans une liste finie et ordonnée de noms.

TYPE SEMAINE=(lundi, mardi, mercredi, jeudi, vendredi, samedi, dimanche)
TYPE COULEUR=(rouge, vert, bleu)

# Notion d'objet

1. *Constantes et variables*
2. *Type logique*
3. *Type CARACTERE*
4. *Type CHAINE DE CARACTERE*
5. *Type énumérés*
6. ***Type intervalles***
7. *Type structurés*
8. *Type pointeur*

Un ***type intervalle*** est un type dont les objets prennent leur valeur dans une portion de l'intervalle des valeurs d'un autre type (entier, énuméré ou caractère).

Exemple : NBRE=0..99
OUVRABLE=lundi..vendredi

# Notion d'objet

1. *Constantes et variables*
2. *Type logique*
3. *Type CARACTERE*
4. *Type CHAINE DE CARACTERE*
5. *Type énumérés*
6. *Type intervalles*
7. ***Type structurés***
8. *Type pointeur*

Une structure est un objet contenant un ensemble d'objets de types différents, appelés champs. Un type doit donc décrire l'ensemble des champs contenus dans ses objets.

Exemple :

```
STRUCTURE ETUDIANT
{  NOM : CHAINE
   NOTE : REEL
   Classement :ENTIER
}
```

# Notion d'objet

*1. Constantes et variables*

*2. Type logique*

*3. Type CARACTERE*

*4. Type CHAINE DE CARACTERE*

*5. Type énumérés*

*6. Type intervalles*

*7.Type structurés*

PLUSTARD !!

***8. Type pointeur***

# ELEMENT DE BASE D'ALGORITHME

## II. Actions élémentaires :

**Actions élémentaires : opérations simple, directement utilisable.**

### A. Affectation

**Permet de donner une valeur à une variable :**

**A ⇦ 28** « reçoit » Si A avait une valeur auparavant, cette valeur disparaît : elle est écrasée par 28

**Format général** :
<id_variable> ⇦ <expression> A⇦28+13
Les types <id_variable> et <expression> doivent être compatibles.

Attention : A⇦B+2
B doit avoir une valeur. Or au début d'une action, les variables ont une valeur indéterminée; B doit avoir été initialisé.

# ELEMENT DE BASE D'ALGORITHME

## II. Actions élémentaires :

Actions élémentaires : opérations simple, directement utilisable.

### A. Affectation

Permet de donner une valeur à une variable :

A ⇦ 28 « reçoit » Si A avait une valeur auparavant, cette valeur disparaît : elle est écrasée par 28

Format général :
<id_variable> ⇦ <expression> A⇦28+13
Les types <id_variable> et <expression> doivent être compatibles.

Attention : A⇦B+2
B doit avoir une valeur. Or au début d'une action, les variables ont une valeur indéterminée; B doit avoir été initialisé.

### B. Les opérations en entrée/ sortie

**Elles permettent de récupérer une valeur venant de l'extérieur (lecture) ou de transmettre une valeur à l'extérieur (écriture) .**

```
VAR A, B, C : ENTIER
LIRE (A, B)
C⇦A+B
ECRIRE (C)
```

plus rapide

```
VAR A, B  : ENTIER
LIRE (A, B)
ECRIRE (A+B)
```

# ELEMENT DE BASE D'ALGORITHME

## III.Les structures de contrôle conditionnelles :

**Une action décrit un enchaînement d'actions élémentaires. L'enchaînement est décrit par les structures de contrôle.**

Une structure de contrôle déjà vue : l'enchaînement séquentiel

```
LIRE (A, B)
C ⇦ 2*A + B
```

La plupart des autres structures de contrôle utilisent la notion de condition (expression booléenne) :

Une condition a une valeur qui est, soit vraie, soit fausse.

Pour déterminer la réalité de cette valeur on utilise :

les opérateurs de comparaisons : =, ≤, ≥, ≠
les opérateurs booléens (logique) : ET, OU, NON

Exemple:

**SUP ⇦ (A > B)** SUP prend la valeur vraie si A est supérieur à B

## III.Les structures de contrôle conditionnelles :

### A. L'alternative SI-ALORS-SINON

Elle permet d'effectuer tel ou tel traitement en fonction de la valeur d'une condition.

**Syntaxe** :

```
SI  <condition>
ALORS < action _alors>
FINSI
```

```
SI <condition>
ALORS < action _alors>
SINON < action _sinon>
FINSI
```

Exemple :

```
LIRE (note)
SI note ≥ 10
ALORS ECRIRE("Admis")
SINON ECRIRE( "Ajourné")
FINSI
```

Remarque, la ligne *SINON* <*action_sinon*> est facultative.

**Principe de fonctionnement :**

1 : la condition est évaluée

2 : Si la condition a la valeur vrai on exécute <action_alors>

Si la condition a la valeur fausse on exécute <action_sinon>

**Remarque :**

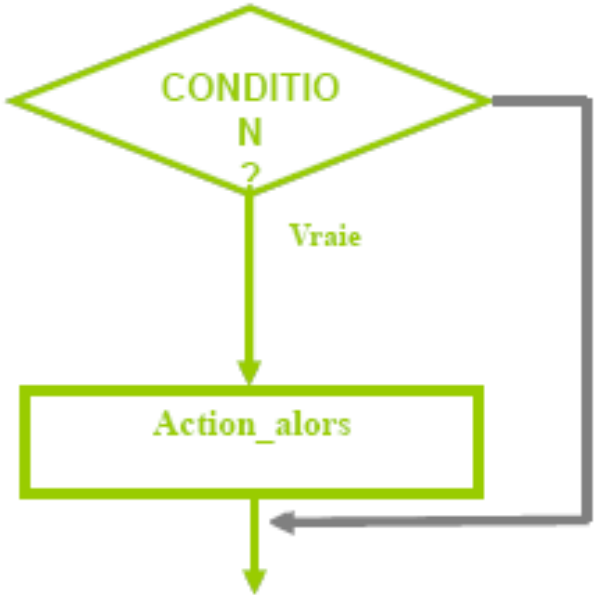
Les <action_alors> ou <action_sinon> peuvent être soit :

- des actions élémentaires
- des composées (bloc)

# ELEMENT DE BASE D'ALGORITHME

## Alternative Simple

SI :



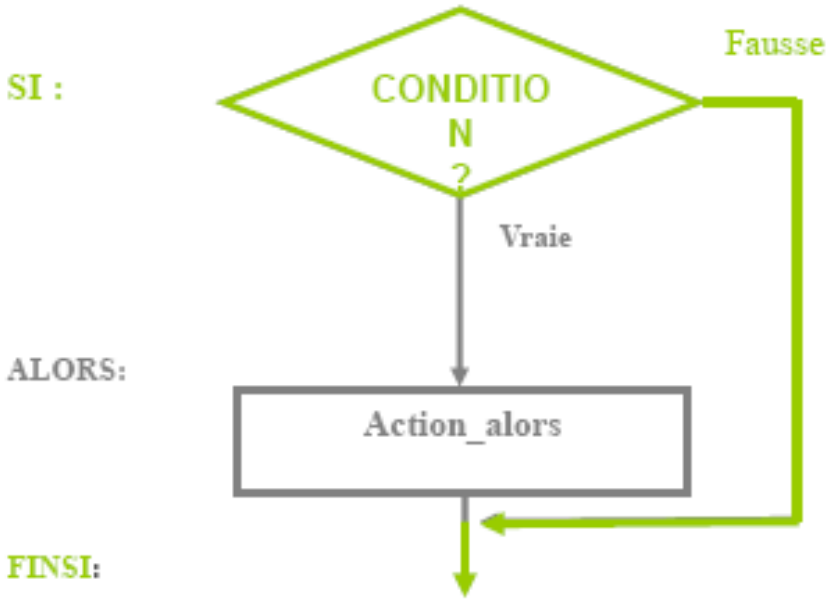

ALORS:

FINSI:

ORGANIGRAMME

# ELEMENT DE BASE D'ALGORITHME

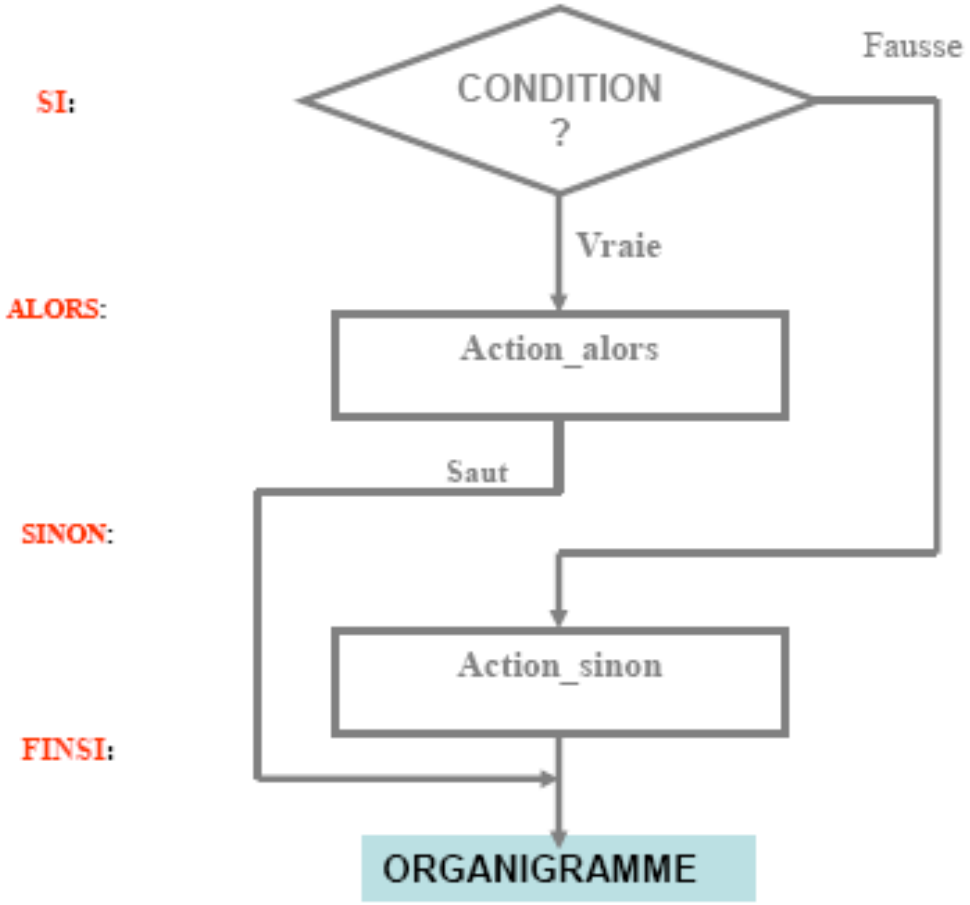
## Alternative Simple




ORGANIGRAMME

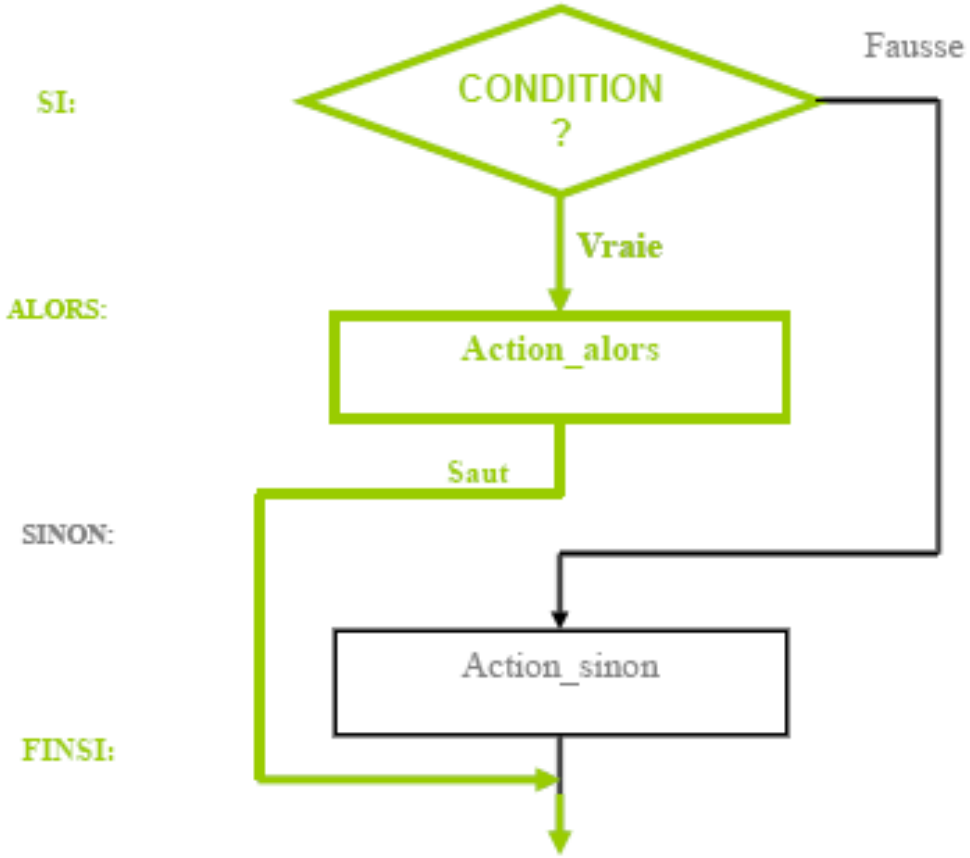
# ELEMENT DE BASE D'ALGORITHME

## Alternative Complète



# ELEMENT DE BASE D'ALGORITHME

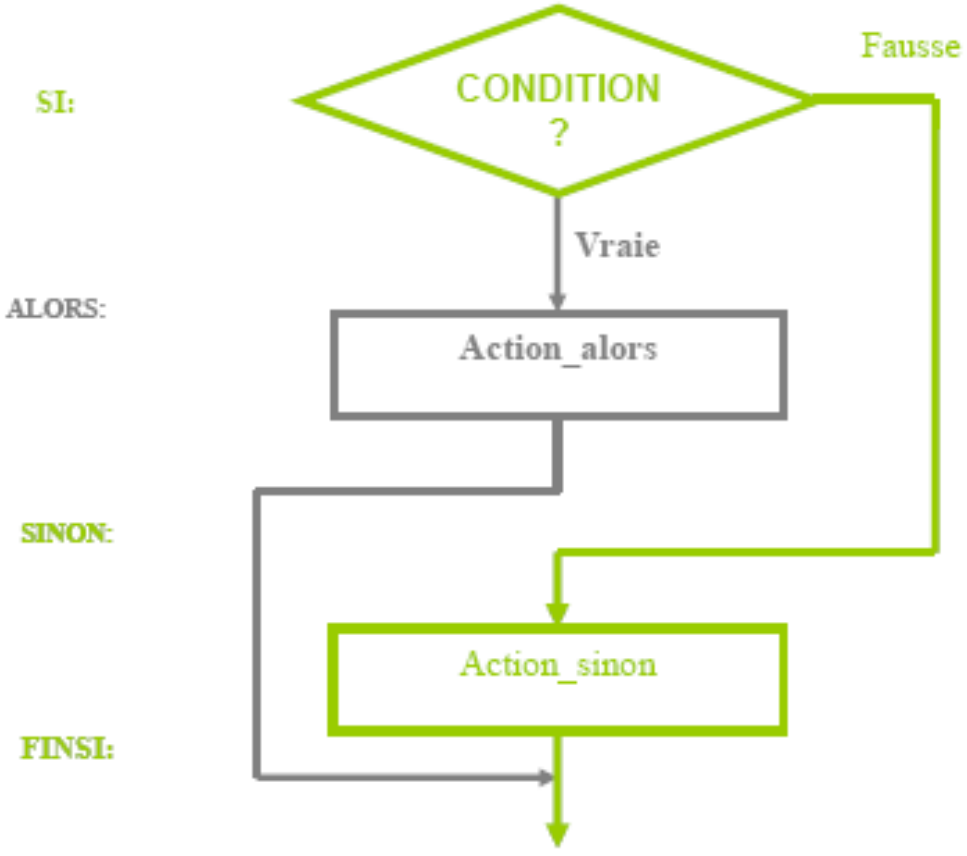
## Alternative Complète




ORGANIGRAMME

# ELEMENT DE BASE D'ALGORITHME

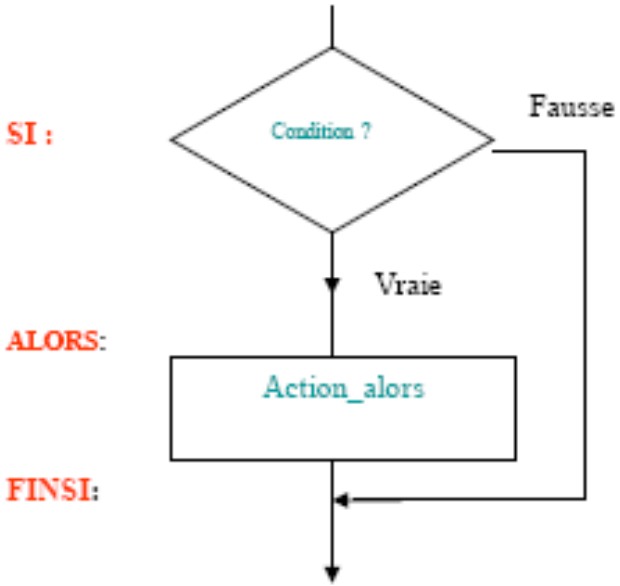
## Alternative Complète



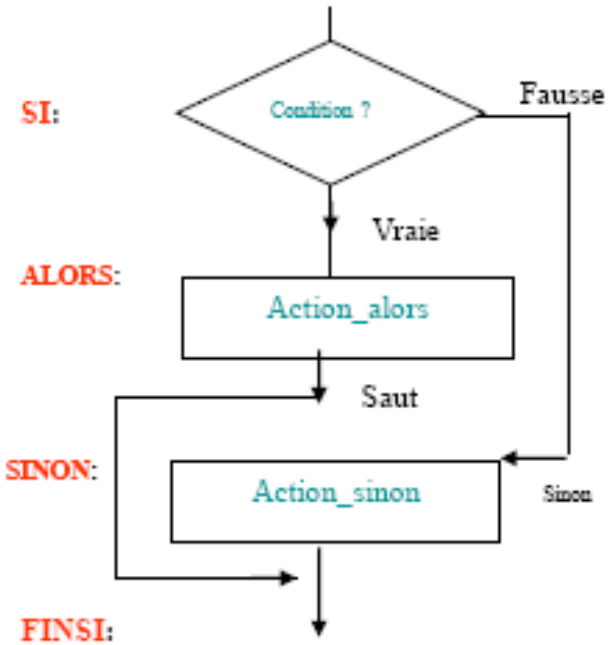

ORGANIGRAMME

# ELEMENT DE BASE D'ALGORITHME

## Alternative Simple




## Alternative Complète




## ORGANIGRAMME

## B. Structure à choix multiples SELON-QUE

**Syntaxe** :

SELONQUE
<condition 1> : <action 1>
<condition 2> : <action 2>
…
<condition n> : <action n>
SINON : <action_sinon>
FINSELONQUE

**Exemple** :

SELONQUE
Note ≥ 16 : ECRIRE("TB")
Note ≥ 14 : ECRIRE("B")
Note ≥ 12 : ECRIRE("AB")
Note ≥ 10 : ECRIRE("Passable")
SINON : ECRIRE("ajourné")
FINSELONQUE

**Fonctionnement** :

la condition 1 est évaluée :
- Si la condition 1 est vraie, alors on exécute l'action correspondante et on quitte la structure selon-que
- Si la condition 1 est fausse , on évalue la condition 2…et ainsi de suite.
- Si aucune n'est vraie on effectue l'action sinon.

Remarque : en programmation, cette structure peut exister mais avec une forme ou un fonctionnement éventuellement différent. Si elle n'existe pas, il faut se souvenir que, en fait, SELON QUE est un raccourci d'écriture pour des SI imbriqués

## IV. Structures répétitives :

**Idée : répéter un ensemble d'opérations, arrêter la répétition en fonction d'une condition**

### A. La structure TANT QUE :

**Syntaxe** :
TANTQUE <condition>
FAIRE <actions>
FINTANTQUE

L'action peut être simple ou composée

**Fonctionnement** :

1 : la condition est évaluée
2 : si la condition est fausse : c'est fini, on quitte le tant que
3 : si la condition est vraie, on exécute le contenu du tant que
puis on remonte à l'étape 1 tester de nouveau la condition

**Exemple** :

```
i⇦1
TANTQUE i≤10
FAIRE
        ECRIRE (i*i)
        i⇦i+1
FINTANTQUE
```

Afficher les dix premiers carrés

## IV. Structures répétitives :

### B. Structure REPETER JUSQU'A :

**Syntaxe** :
REPETER
<actions>
JUSQU'A <condition>

Fonctionnement :

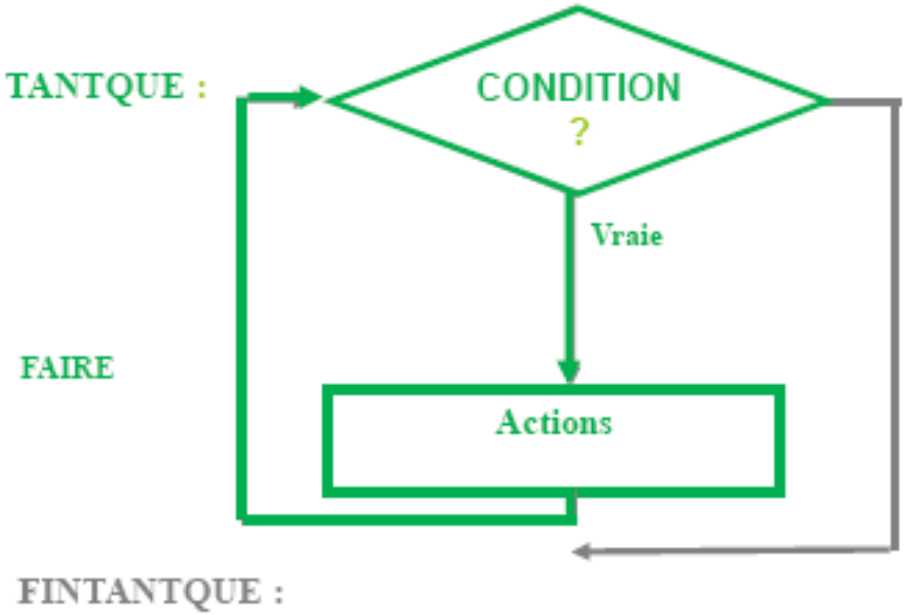
1 : on exécute le corps(actions)
2 : on évalue la condition
3 : si Vraie : on quitte le répéter
4 : si Fausse on recommence

Remarques :

Il y a toujours au moins une exécution du corps. La structure répéter permet de répéter un traitement 1 ou plusieurs fois.

# ELEMENT DE BASE D'ALGORITHME

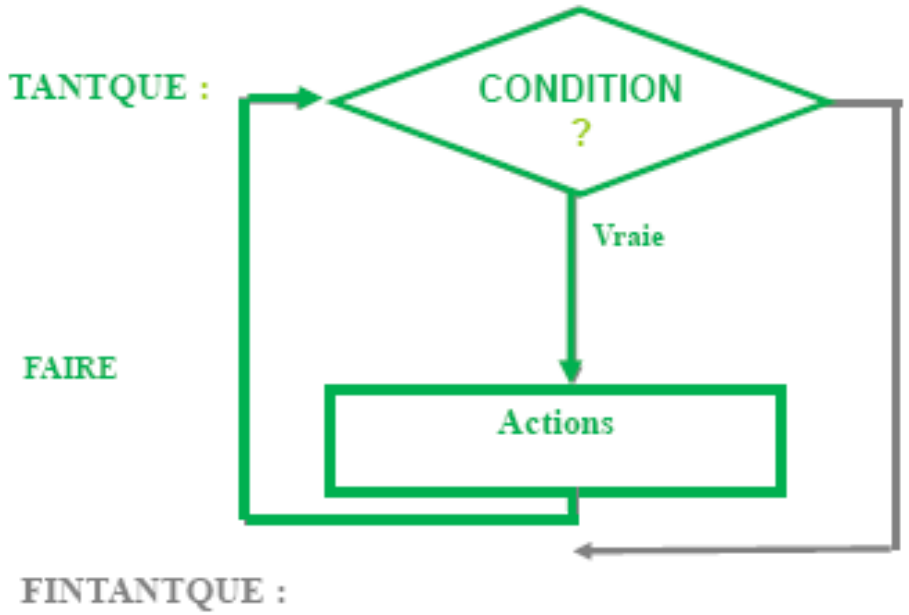
## BOUCLE TANTQUE




## ORGANIGRAMME

# ELEMENT DE BASE D'ALGORITHME

## BOUCLE TANTQUE

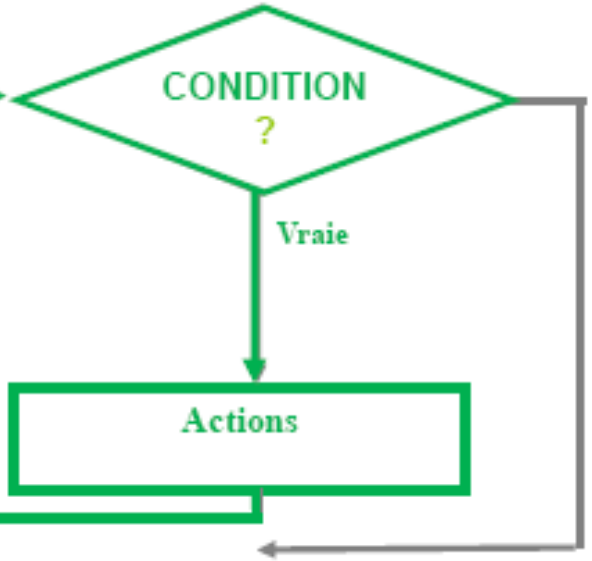



## ORGANIGRAMME

# ELEMENT DE BASE D'ALGORITHME

## BOUCLE TANTQUE

TANTQUE :



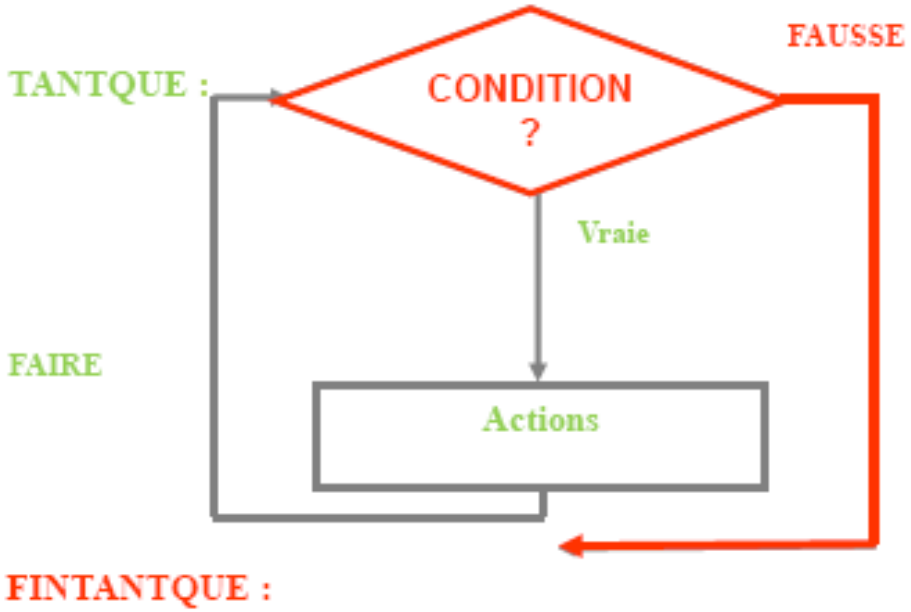

FAIRE

FINTANTQUE :

## ORGANIGRAMME

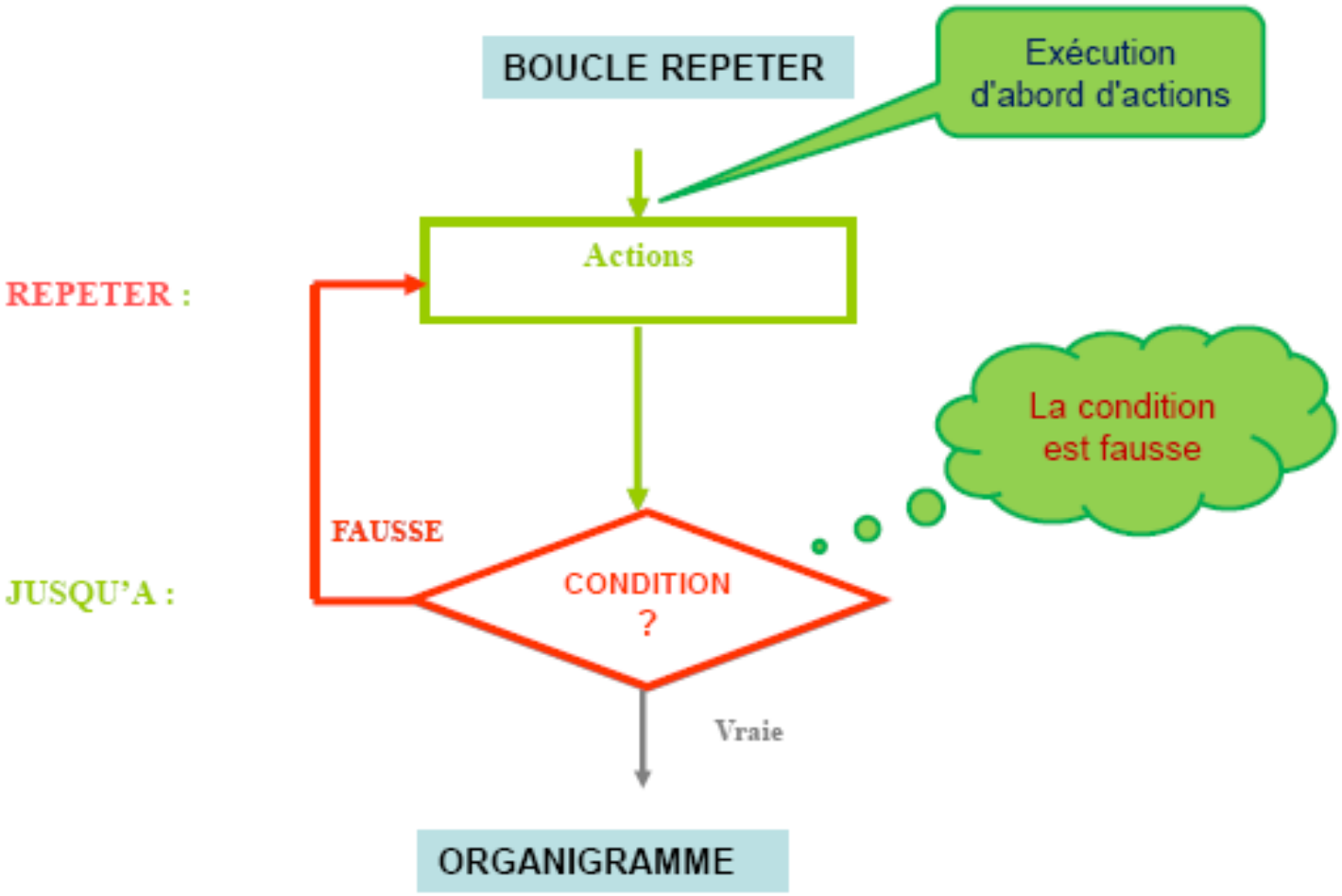
# ELEMENT DE BASE D'ALGORITHME

## BOUCLE TANTQUE




## ORGANIGRAMME

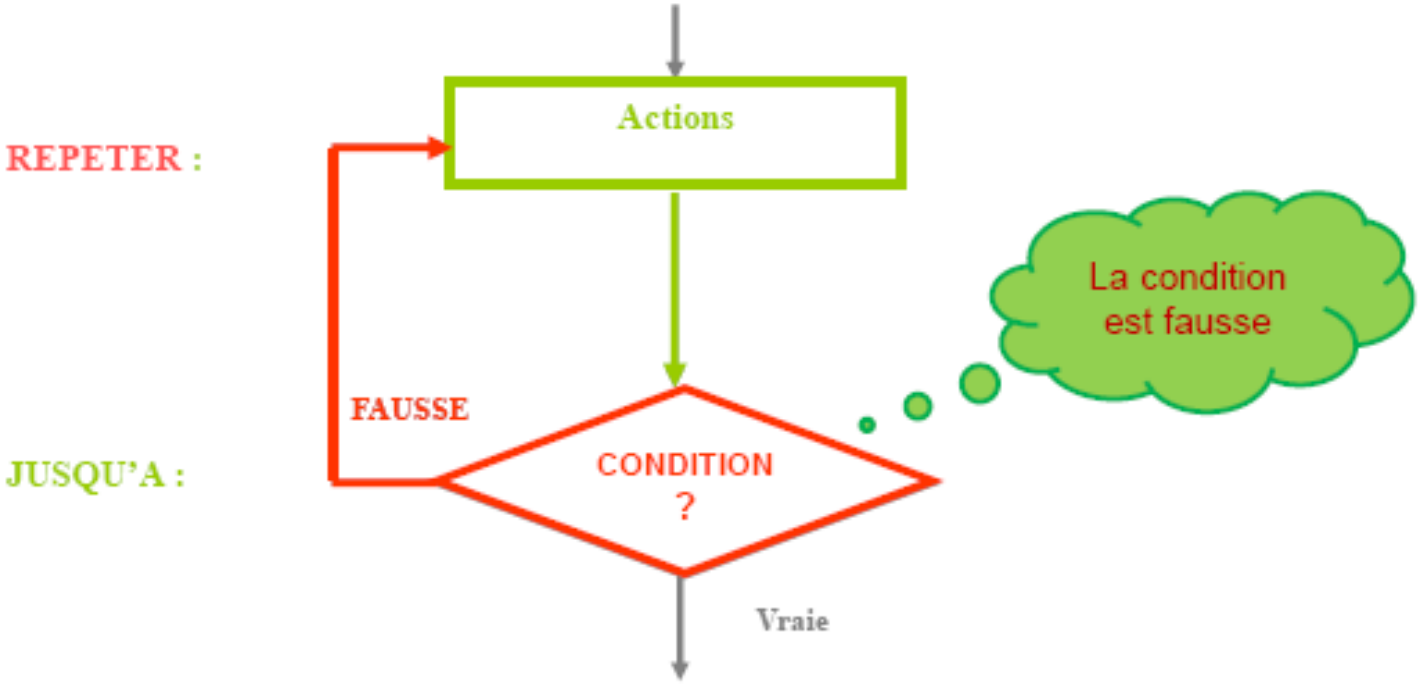
# ELEMENT DE BASE D'ALGORITHME




ORGANIGRAMME

# ELEMENT DE BASE D'ALGORITHME

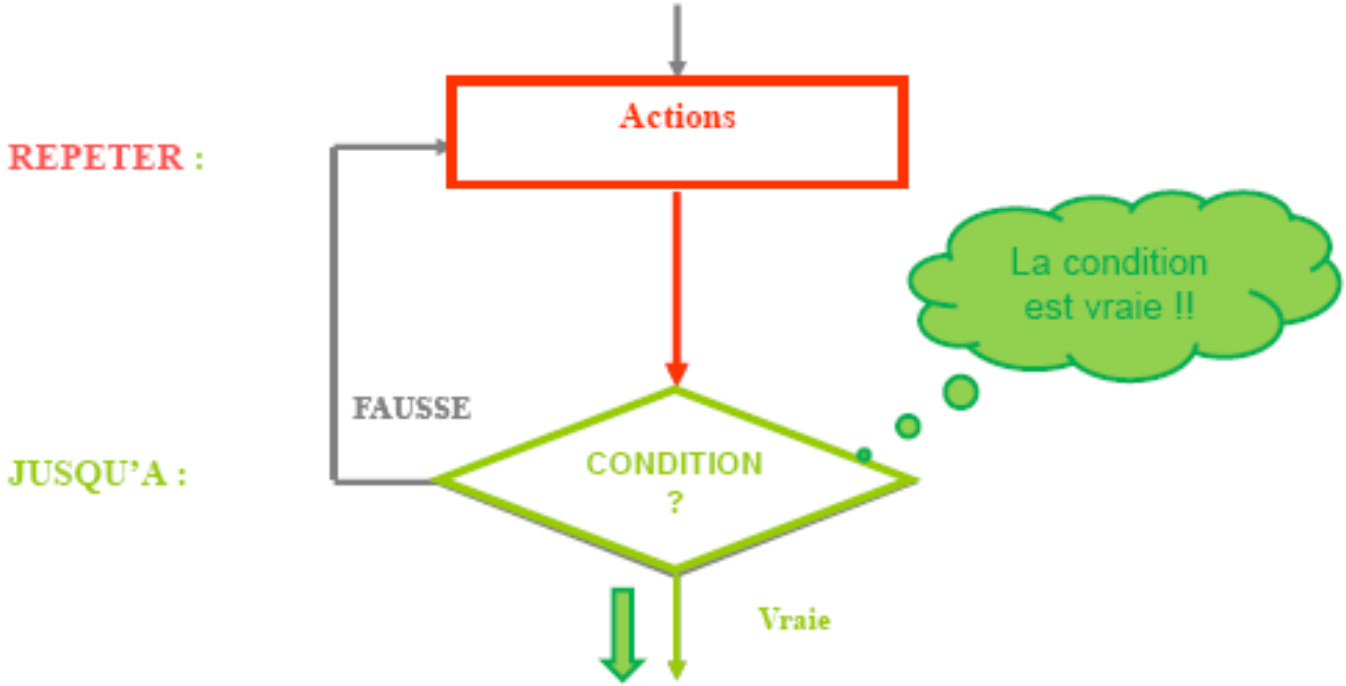
## BOUCLE REPETER




## ORGANIGRAMME

# ELEMENT DE BASE D'ALGORITHME

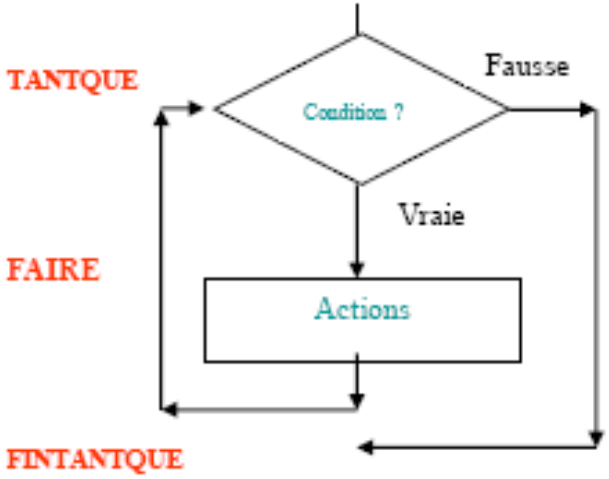
## BOUCLE REPETER



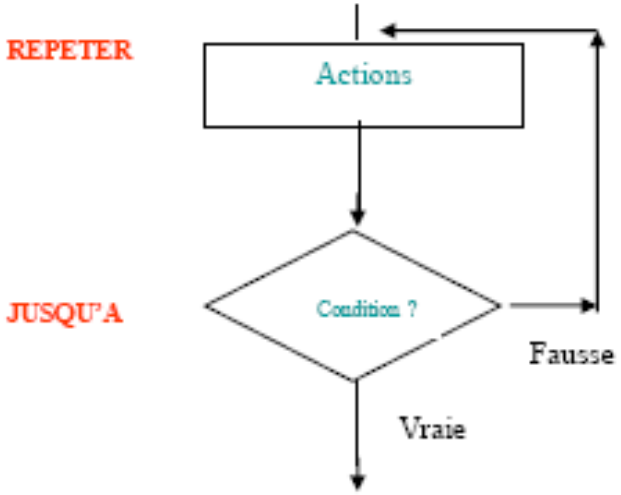

ORGANIGRAMME

# ELEMENT DE BASE D'ALGORITHME

## Boucle TANTQUE




## Boucle REPETER




## ORGANIGRAMME

## IV.Structures répétitives :

### C. Structure POUR

**Elle permet de parcourir un intervalle en répétant un traitement pour chacune des valeurs de cet intervalle.**

**Syntaxe** :

POUR <id_variable> DE <val_inférieure> A <val_supérieure>
[ par pas de <val_pas>] ⇦ facultatif
FAIRE <actions>
FINPOUR

**Fonctionnement** :

1 : Automatiquement Au départ , on a id_variable ⇦ val_inférieure
Donc, on n'a pas besoin d'initialiser, la structure se charge de la faire
2 : id_variable > val_supérieure ?
Si oui, alors STOP, on quitte la structure
Sinon :
– on exécute le programme
– automatiquement, l'incrémentation se fait (+1 ou +pas si l'on a définit un pas particulier, par défaut, le pas est 1)
– on remonte au début du 2 tester la condition id_variable > val_supérieure ?

# ELEMENT DE BASE D'ALGORITHME

## IV.Structures répétitives :

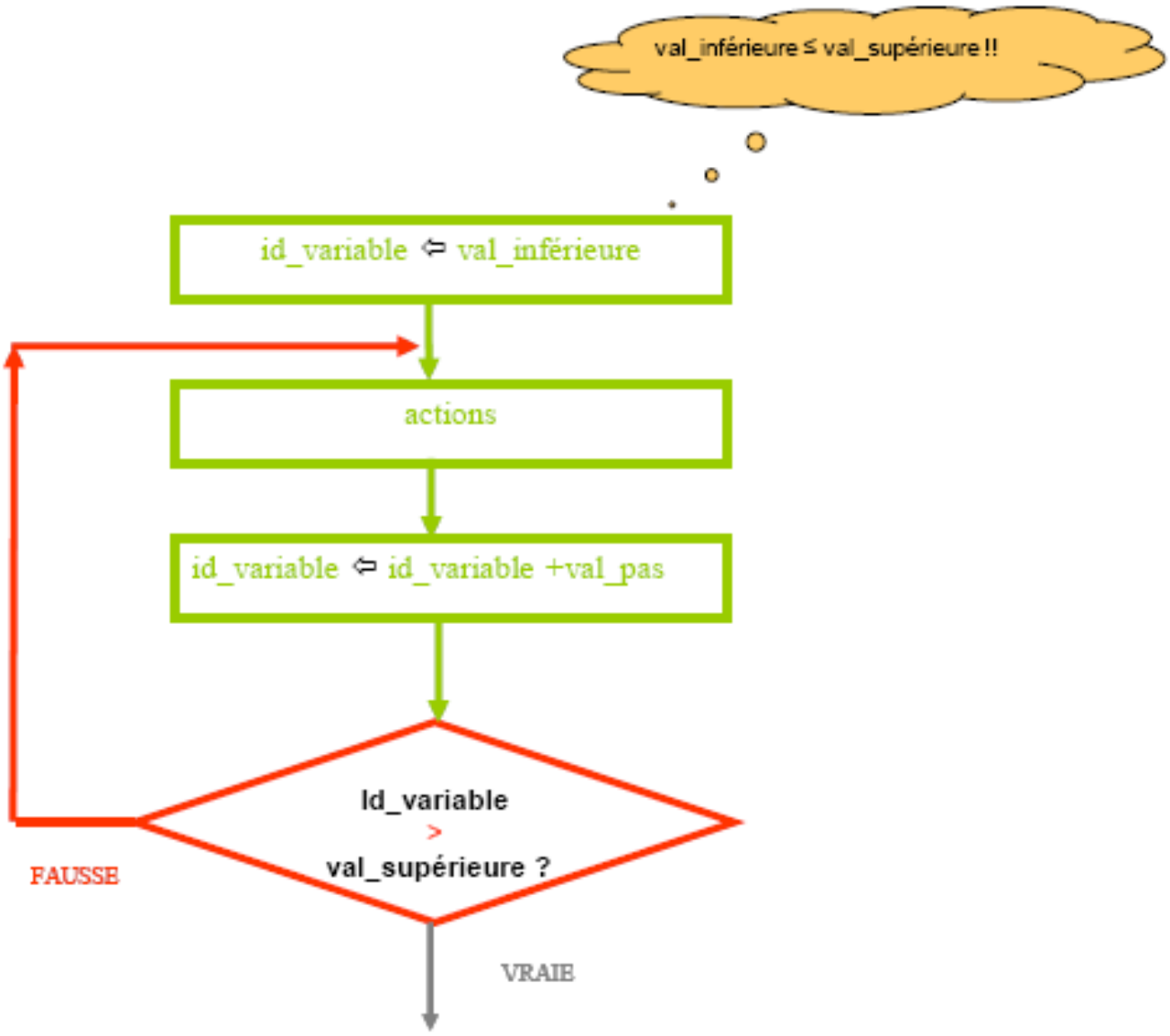
### STRUCTURE POUR

POUR <id_variable>
DE <val_inférieure>
A <val_supérieure>
[ par pas de <val_pas>
FAIRE <actions>
FINPOUR

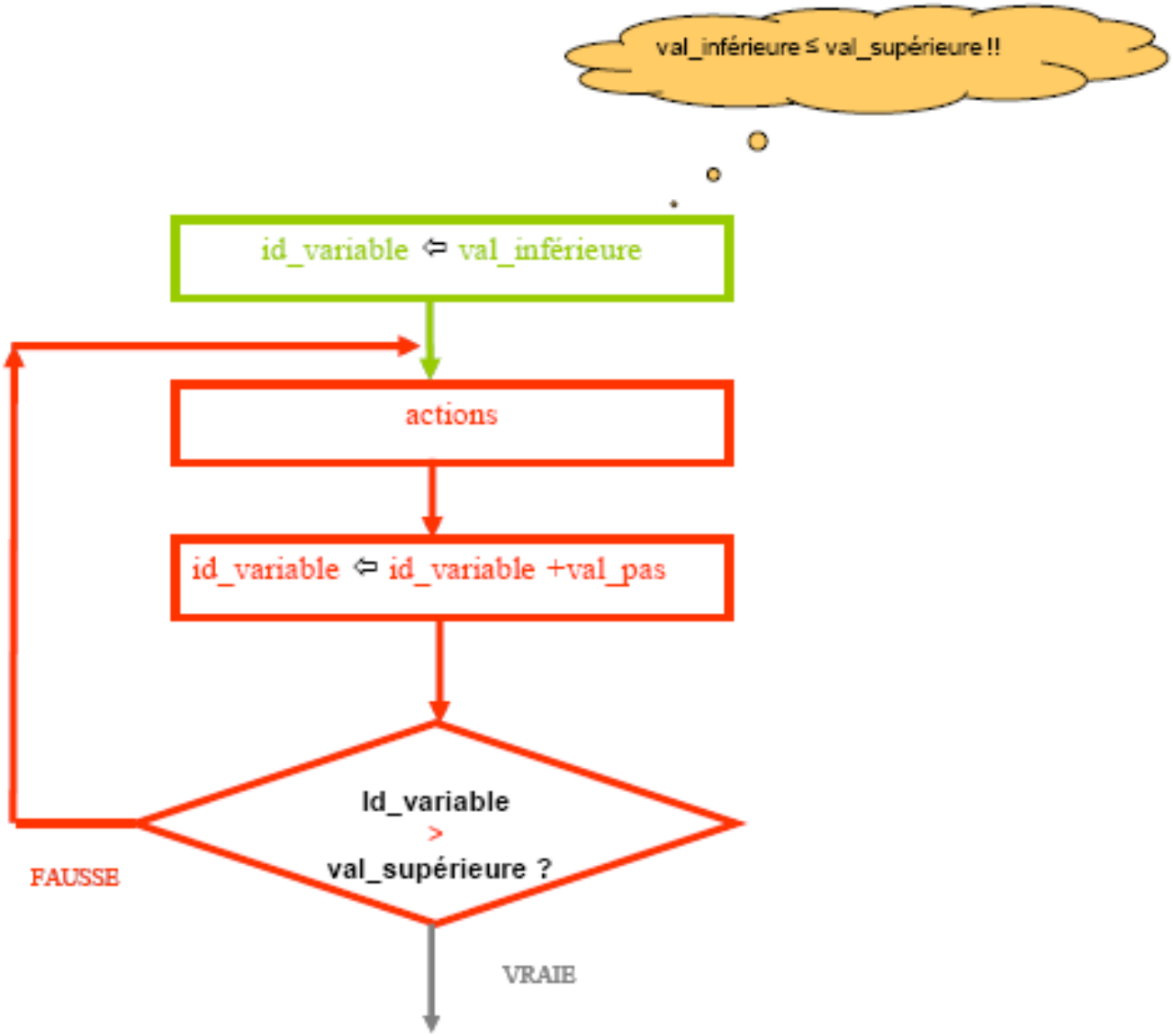
**Exemple** :Factorielle de n : n!=1*2*3..*(n-1)*n

Fact ⇦ 1
POUR i DE 1 A n
FAIRE fact ⇦ fact*i
FINPOUR

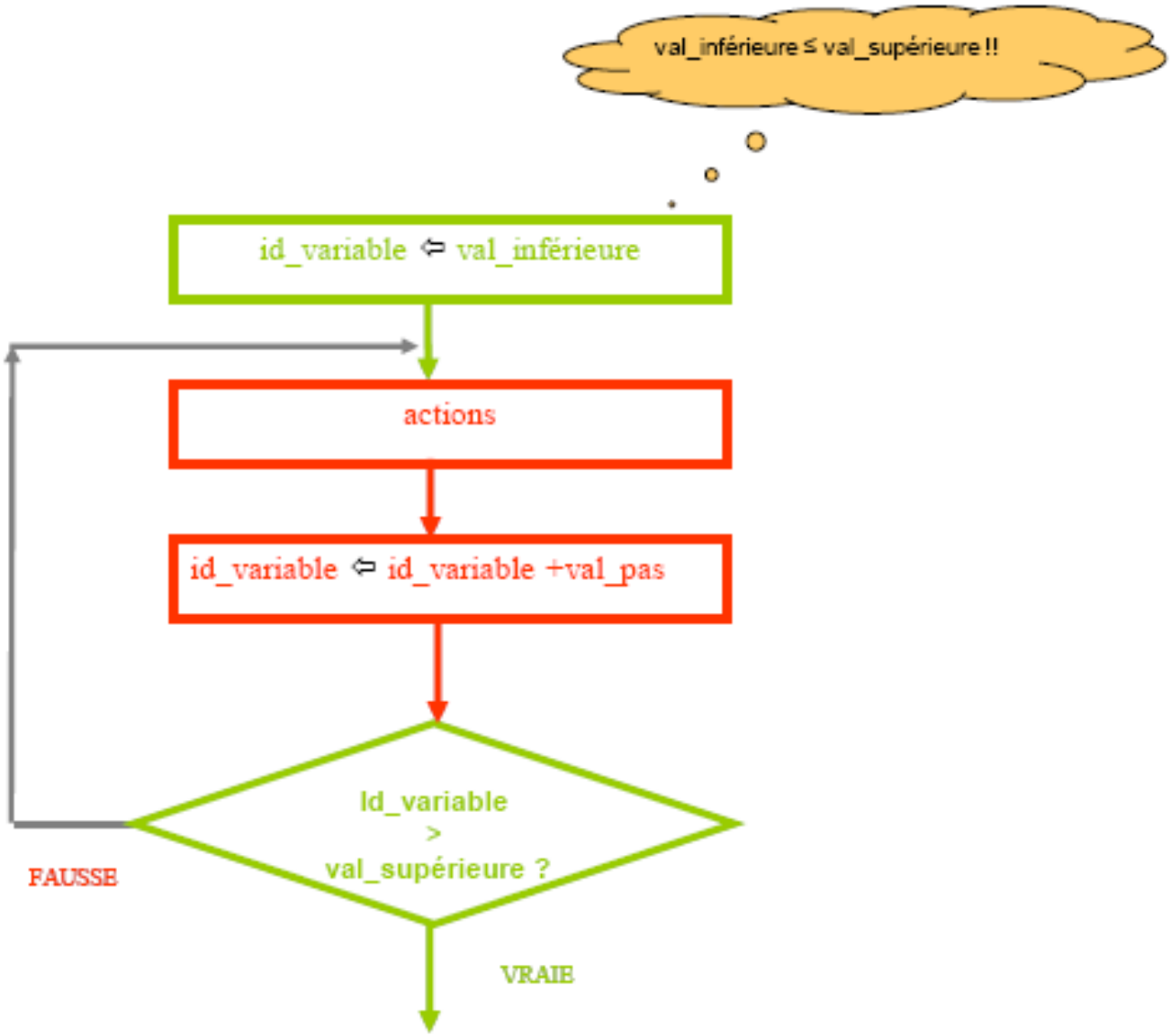
# BOUCLE POUR


val_inférieure ≤ val_supérieure !!
id_variable ⇦ val_inférieure
actions
id_variable ⇦ id_variable +val_pas
Id_variable
>
val_supérieure ?
FAUSSE
VRAIE

# BOUCLE POUR


val_inférieure ≤ val_supérieure !!
id_variable ⇦ val_inférieure
actions
id_variable ⇦ id_variable +val_pas
Id_variable
>
val_supérieure ?
FAUSSE
VRAIE

# BOUCLE POUR


val_inférieure ≤ val_supérieure !!
id_variable ⇦ val_inférieure
actions
id_variable ⇦ id_variable +val_pas
Id_variable
>
val_supérieure ?
FAUSSE
VRAIE

# ELEMENT DE BASE D'ALGORITHME

## V.Procedures et Fonctions :

Un algorithme est composé d'un ensemble fini d'actions. En fait, on distingue :

- les **Procedures** (ou ACTIONS) qui réalisent un traitement ( tri du fichier étudiant…)
- les **fonctions** qui effectuent un calcul et retournent un résultat

### A. PROCEDURE

**Syntaxe**

PROCEDURE <nom_procedure> <( Liste des paramètres)>
< déclaration des objets locaux de la procedure>
DEBUT
{corps de la procedure}
FIN

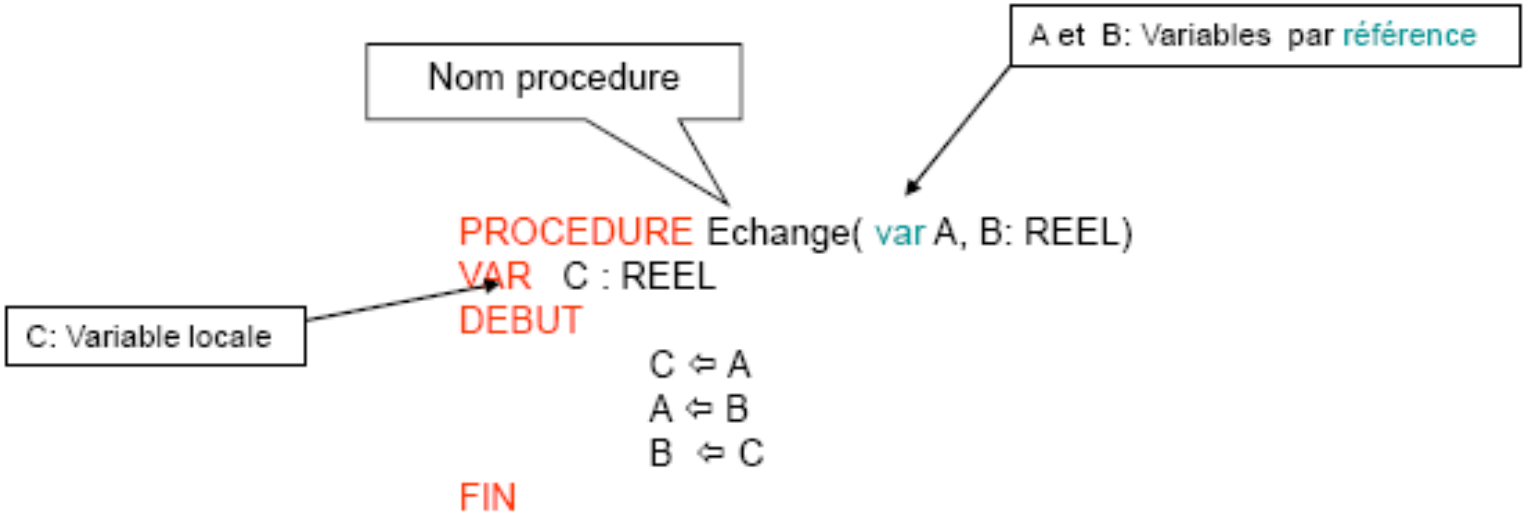
Il existe deux types de paramètres transmis:

- par valeur
- par référence

## V.Procedures et Fonctions :

### Exemple d'une Procedure

Ecrire un algorithme d'une procédure qui échange deux variables A et B



## V.Procedures et Fonctions :

### B. FONCTIONS

**Syntaxe**

FONCTION <nom_fonction> ( <liste des paramètres> ) : <type de résultat>
< déclaration des objets locaux à la fonction>
DEBUT
{ corps de la fonction}
RETOURNER(resultat )
FIN

À Ne Pas oublier !!

**Exemple d'une Fonction**

Ecrire une fonction maximum qui donne (Retourne ) le maximum de deux entiers :

## V.Procedures et Fonctions :

**Exemple:**

```
FONCTION maximum (x, y: ENTIER) : ENTIER
VAR   max :ENTIER
DEBUT
      SI     x < y
      ALORS              max ⇦ y
      SINON              max ⇦ x
      FINSI
      RETOURNER (max)
FIN
```

**Fonction** qui détermine le maximum de deux entiers

```
PROCEDURE minETmax( a,b: ENTIER)
VAR Min,Max:ENTIER
DEBUT
      Min  ⇦ -maximum (-a, -b)
      Max ⇦  maximum (a, b)
      ECRIRE (min, max)
FIN
```

**Procedure** qui affiche le minimum et le maximum de deux entiers, en faisant appel à la fonction maximum

## V.Procedures et Fonctions :

### Fonction récursive:

**La récursivité consiste à remplacer une boucle par un appel à la fonction elle-même.**

**Exemple :**

Considérons la suite factorielle, elle est définie par :

0!=1
n! = (n-1)! * n ⇒ factorielle(0)=1
factorielle(n)= factorielle(n-1) *n

La fonction peut s'écrire simplement

```
FONCTION factorielle (n: ENTIER):ENTIER
DEBUT
      SI  (n=0)
      ALORS          retourner(1)
      SINON          retourner( factorielle(n-1) *n)
      FINSI
FIN
```

FIN

ET

MERCI

TABLEAUX

# Cours
# Introduction à l'algorithmique

# Tableaux

**Ensemble de données du même type**

**Exemple de problème :**

Saisir une suite de nombres, puis afficher cette suite après avoir divisé tous les nombres par la valeur maximale de la suite.

Nécessité de conserver les nombres en mémoire

**val** | 132 | : Variable contenant une valeur variable

**Val**

| 132 | 0 | 8100 | -641 | 841 | 8902 | 57 | -21 |
|---|---|---|---|---|---|---|---|

: Variable contenant une collection de valeurs du même type

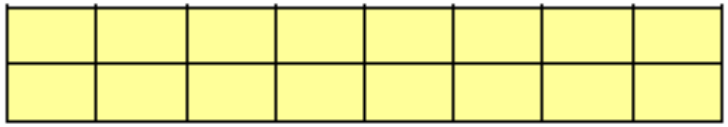
*Remarque : appeler cette variable **TabVal** plutôt que **Val***

# Tableaux

**Structure de données permettant d'effectuer un même traitement sur des données de même nature**

tableau à **une** dimension

tableau à **deux** dimensions



**Exemples d'applications:**

- Ensemble de valeurs entières, réelles, booléennes,....
- Ensemble de noms (type chaîne)
- Ensemble de caractères (type caractère)
- Ensemble d'adresses (type Adresse : nom,adresse, num téléphone)
- Ensemble d'ouvrages

.....

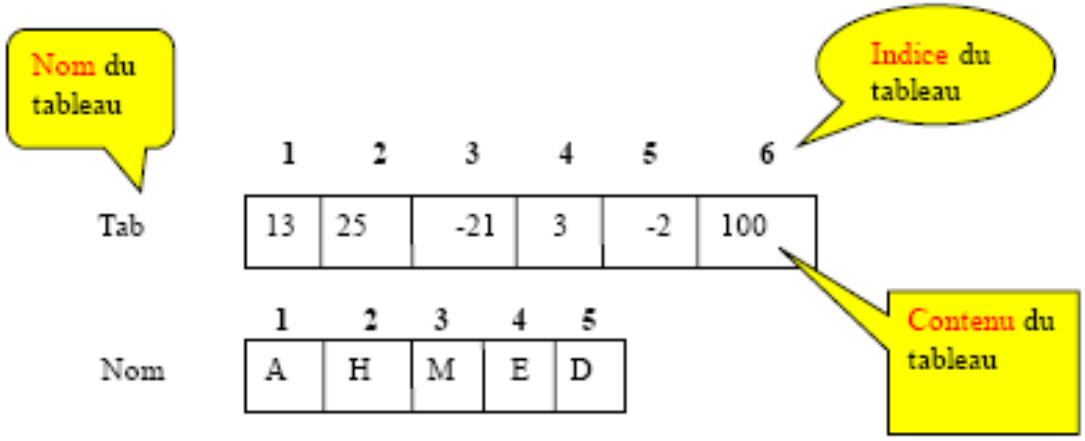
# Tableaux

**Traitements opérant sur des tableaux**

On veut pouvoir :

- ✓ **créer** des tableaux
- ✓ **ranger** des valeurs dans un tableau
- ✓ **récupérer, consulter** des valeurs rangées dans un tableau
- ✓ **rechercher** si une valeur est dans un tableau
- ✓ **mettre à jour** des valeurs dans un tableau
- ✓ **modifier** la façon dont les valeurs sont rangées dans un tableau (par exemple : les trier de différentes manières)
- ✓ effectuer des **opérations entre tableaux** : comparaison de tableaux, multiplication,...
- ✓ ...

# Tableaux



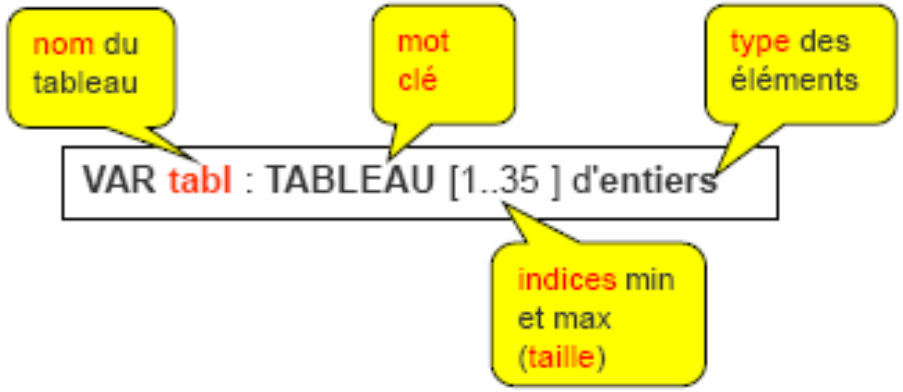

**Remarques** :

- Indices : en général, démarrage à 1, **mais en C++, démarrage à 0**
- Nombre d'octets occupés : dépend du type des valeurs enregistrées

# Tableaux

## Déclaration d'un tableau

**Exemple** : déclaration d'un tableau pouvant contenir jusqu'à 35 entiers




**Autre exemple** : déclaration d'un tableau qui contiendra les fréquences des températures comprises entre –40°C et 50°C

**VAR Temperatures : TABLEAU [1..91] de REEL**

Avec **Temperatures [1]** correspond à la fréquence de la température **-40°c**

**Question:** la fréquence de la température **25 °c → Temperatures [ ?]**

**VAR Temperatures : TABLEAU [-40..50] de REEL**
**Temperatures [-30] → Fréquence temp -30°C**

# Tableaux

## Définition d'un type tableau

TYPE <Nom> = <description>

**Exemple** : *déclaration d'un nouveau type Mot, tableau de 10 caractères*

TYPE **Mot** = **TABLEAU** [1..10 ] de **caractères**

**VAR** nom, verbe : **Mot**

# Tableaux

**Utilisation d'un tableau : par les indices**

• Accès en lecture :

- **x** ← T[2] */*le contenu du tableau à l'indice 2 est placé dans x */*

- **Ecrire** (T[4]) */*le contenu du tableau à l'indice 4 est affiché à l' écran*/*

• Accès en écriture :

-T[3] ←18 */*la valeur 18 est placée dans le tableau à l'indice 3*/*

-**Lire** (T[5]) */*la valeur entrée par l'utilisateur est enregistrée dans le tableau à l'indice 5*/*

# Tableaux

**CONST** TAILLE=50

**TYPE** TABL= **TABLEAU** [1..TAILLE] d'ENTIER

Ecrire la PROCEDURE **SAISIR(...)** qui permet de saisir un tableau T de **N** éléments
( N ≤ TAILLE )

```
PROCEDURE SAISIR( VAR T  : TABL;N :ENTIER)
VAR i : ENTIER
DEBUT
        POUR i DE 1 A  N
                Lire (T[i])
        FINPOUR
FIN
```

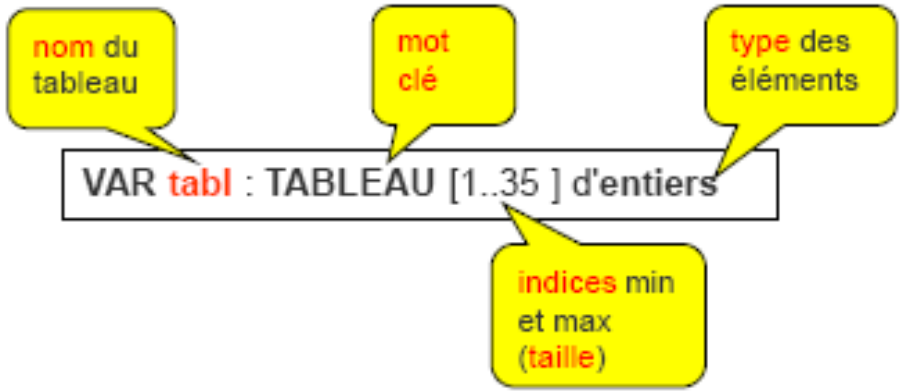
Ecrire la PROCEDURE AFFICHER(...) qui permet d'afficher un tableau T de **N** éléments
( N ≤ TAILLE )

```
PROCEDURE AFFICHER (T :TABL; N :ENTIER)
VAR i :ENTIER
DEBUT
        POUR i DE 1 A  N
                ECRIRE (T[i])
        FINPOUR
FIN
```

# Tableaux

## Déclaration d'un tableau

**Exemple** : déclaration d'un tableau pouvant contenir jusqu'à 35 entiers



# Tableaux

## Définition d'un type tableau

TYPE <Nom> = <description>

**Exemple** : *déclaration d'un nouveau type Mot, tableau de 10 caractères*

TYPE **Mot** = **TABLEAU** [1..10 ] de **caractère**

**VAR** nom, verbe : **Mot**

# Tableaux

**CONST** TAILLE=50

**TYPE** TABL= **TABLEAU** [1..TAILLE] d'entiers

Ecrire la PROCEDURE **SAISIR(...)** qui permet de saisir un tableau T de **N** éléments
( N ≤ TAILLE )

```
PROCEDURE SAISIR(VAR T  : TABL;N :entier)
VAR i : entier
DEBUT
        POUR i DE 1 A  N
                Lire (T[i])
        FINPOUR
FIN
```

Ecrire la PROCEDURE **AFFICHER(...)** qui permet d'afficher un tableau T de **N** éléments
( N ≤ TAILLE )

```
PROCEDURE AFFICHER (T :TABL; N :entier)
VAR i :entier
DEBUT
        POUR i DE 1 A  N
                ECRIRE (T[i])
        FINPOUR
FIN
```

# TRI TABLEAUX

Trier des objets est à la fois un problème fondamental de l'algorithmique, comme étape de prétraitement d'algorithmes plus complexes ou pour ordonner des structures de données (à quoi servirait un annuaire non trié)

Nous présentons ici des tris simples possibles sur les tableaux:

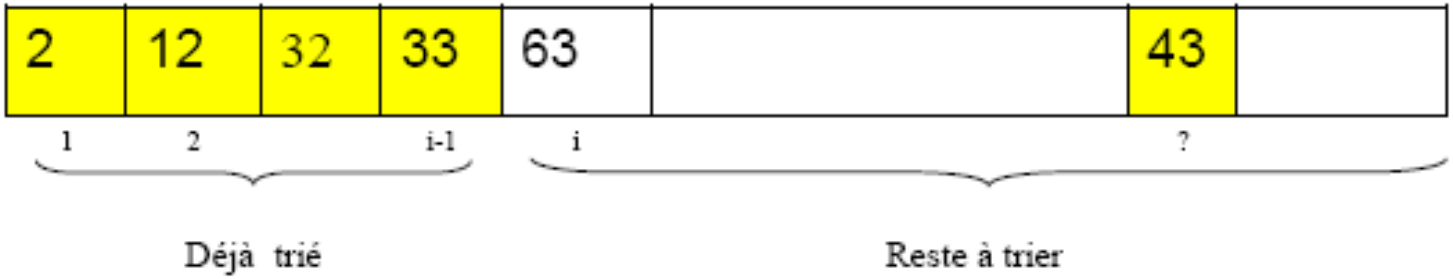
1. **Tri par sélection**
2. **Tri par Insertion**
3. **Tri à bulle**

# TRI TABLEAUX

## Tri par Sélection

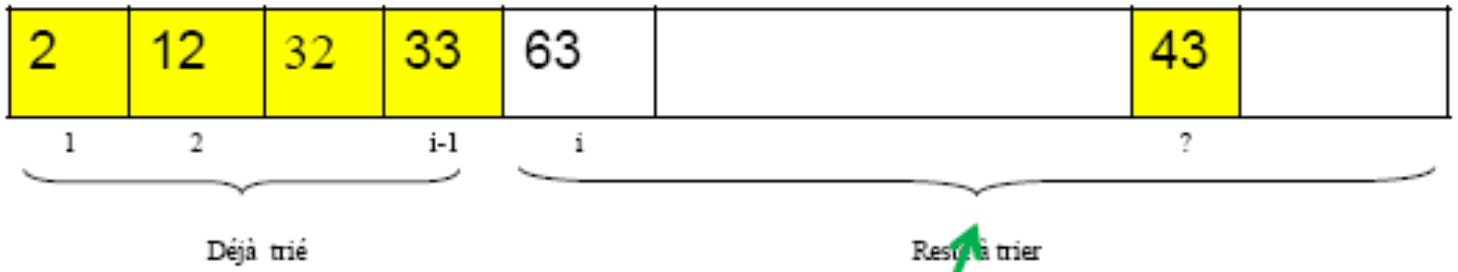
### Algorithme de tri par sélection

L'idée du tri consiste à chaque étape à rechercher le plus **petit élément** non encore trié et à le placer à la suite des éléments déjà triés. A une étape i, les i − 1 plus petits éléments sont en place, et il nous faut sélectionner le ième élément à mettre en position i.



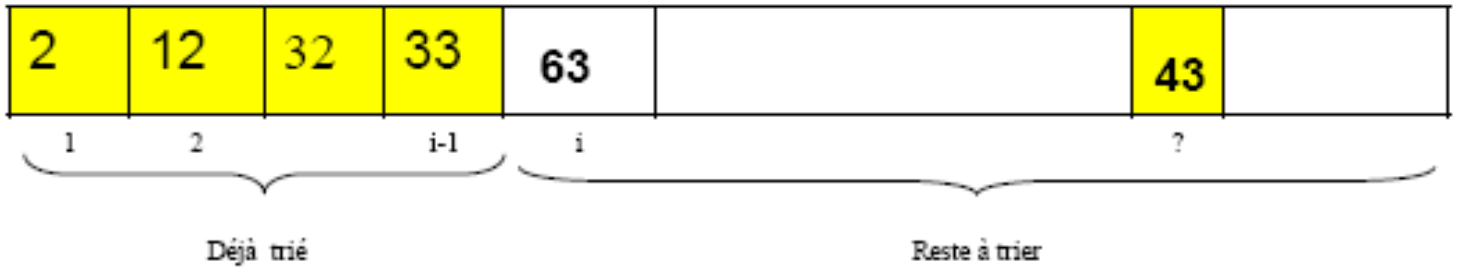

1. Chercher le plus petit élément du tableau restant
2. Echanger cet élément avec l'élément en position i

# TRI TABLEAUX




```
ALGORITHME TriSélection
CONST  TAILLE=50
TYPE TABL=TABLEAU[1..TAILLE] d'ENTIER
VAR  T:  TABL
N,i, pos :ENTIER
DEBUT
      ECRIRE(" Entrer le nombre d'élément du tableau < = ",TAILLE)
      LIRE(N)
      SAISIR(T,N)
      POUR  i DE  1 A  N  -1
      FAIRE
            /* Etapes de l'algorithme*/
            pos  ⇦ TROUVERMIN(T ,i,N)        /* Plus petit élément du tableau restant*/
            ECHANGER(T,i,pos)   /* ECHANGER avec l'élément  en position i*/
      FINPOUR
      AFFICHER(T,N)
FIN
```

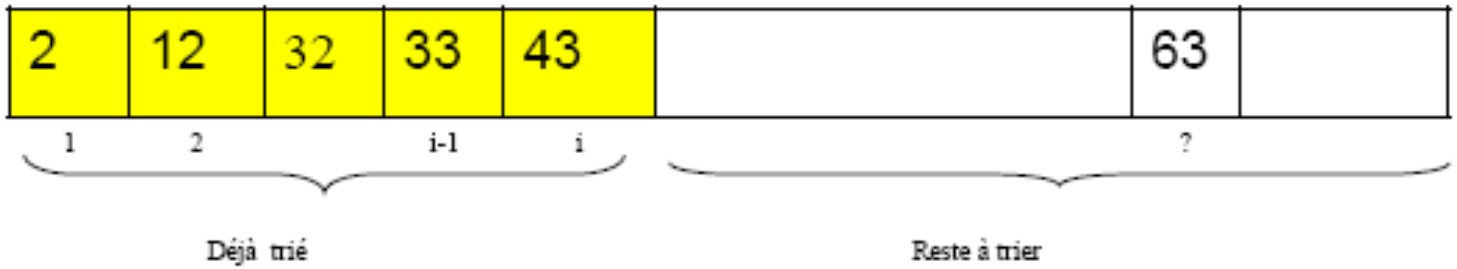
# TRI TABLEAUX




```
FONCTION  TROUVERMIN( T: TABL i,N: ENTIER ):ENTIER
VAR i_Min :ENTIER
DEBUT
        i_Min ⇦  i
        POUR  j DE   i_Min +1 A N
        FAIRE
                SI T [j] < T[i_Min]
                ALORS
                        i_Min ⇦  j
                FINSI
        FINPOUR
        Retourner( i_Min)
FIN
```

```
PROCEDURE ECHANGER(Var T:  TABL; i,j: ENTIER
VAR Aux :ENTIER
DEBUT
        Aux ⇦ T[i]
        T[i] ⇦ T[j]
        T[j] ⇦  Aux
FIN
```

# TRI TABLEAUX



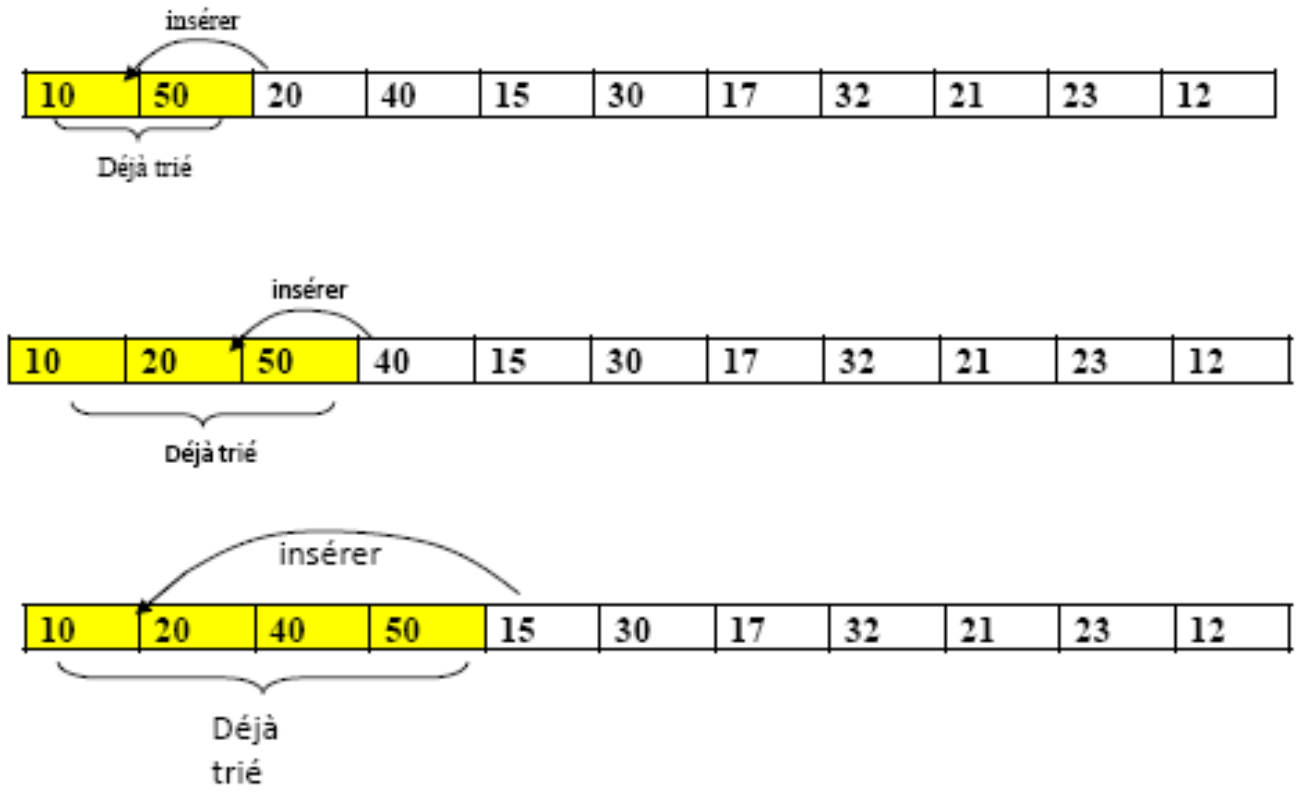

```
FONCTION  TROUVERMIN( T: TABL i,N: ENTIER ):ENTIER
VAR i_Min :ENTIER
DEBUT
        i_Min ⇦  i
        POUR  j DE   i_Min +1 A N
        FAIRE
                SI T [j] < T[i_Min]
                ALORS
                        i_Min ⇦  j
                FINSI
        FINPOUR
        Retourner( i_Min)
FIN
```

```
PROCEDURE ECHANGER(VAR T:  TABL; i,j: ENTIER
VAR Aux :ENTIER
DEBUT
        Aux ⇦ T[i]
        T[i] ⇦ T[j]
        T[j] ⇦  Aux
FIN
```

# TRI TABLEAUX

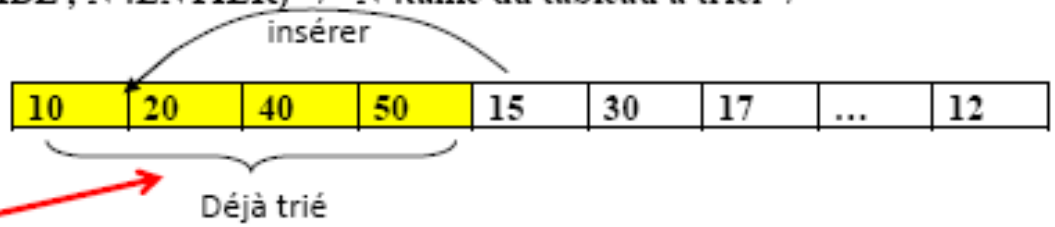
Écrire la Procedure **TRI_INSERTION** qui utilise la Procedure INSERER pour trier par ordre croissant les éléments d'un tableau de N éléments.

**Méthode**: Trier le tableau de gauche à droite en insérant à chaque fois l'élément i+1 dans le tableau (déjà trié) des i premiers éléments.



# TRI TABLEAUX

```
PROCEDURE TRI_INSERTION( VAR T :TABL ; N :ENTIER)   /* N :taille du tableau à trier*/
VAR  i, indice          :ENTIER
Trouve                  :BOOLEEN
DEBUT
      POUR  i DE 2 A N
      FAIRE
            indice ⇦1
            Trouve ⇦ Faux
            TANTQUE (Non Trouve) ET  (indice < i)
            FAIRE
                  SI T[indice] > T[i] ALORS
                        INSERER(T,indice,i)
                        Trouve ⇦ Vraie
                  FINSI
                  indice ⇦indice  +1
            FINTANTQUE
      FINPOUR
FIN
```




```
PROCEDURE  INSERER(var T :TABL ; indice,i :ENTIER)
VAR j :ENTIER
DEBUT
      Aux ⇦T[i]
      POUR j DE i-1 A indice Par pas -1
       FAIRE
            T[j+1] ⇦ T[j]
      FINPOUR
      T[indice] ⇦Aux
FIN
```

Décalage à droite des éléments

# TRI TABLEAUX

```
PROCEDURE TRI_INSERTION( VAR T :TABL ; N :ENTIER)   /* N :taille du tableau à trier*/
VAR  i, indice              :ENTIER
Trouve                      :BOOLEEN
DEBUT
      POUR  i DE 2 A N
      FAIRE
            indice ⇦1
            Trouve ⇦ Faux
            TANTQUE (Non Trouve) ET  (indice <  i)
            FAIRE
                  SI T[indice] > T[i] ALORS
                        INSERER(T,indice,i)
                        Trouve ⇦ Vraie
                  FINSI
                  indice ⇦indice  + 1
            FINTANTQUE
      FINPOUR
FIN
```

```
POUR  i DE 2 A N
FAIRE
      indice ⇦1
      TANTQUE (T[indice] ≤T[i]  ET ( indice < i)
      FAIRE
            indice ⇦indice  +1
      FINTANTQUE
      SI (indice ≠ i )
      ALORS INSERER(T,indice,i)
      FINSI
FINPOUR
```

Attention au cas de indice = i
Pas d'insertion !

# TRI TABLEAUX

## Tri à bulle

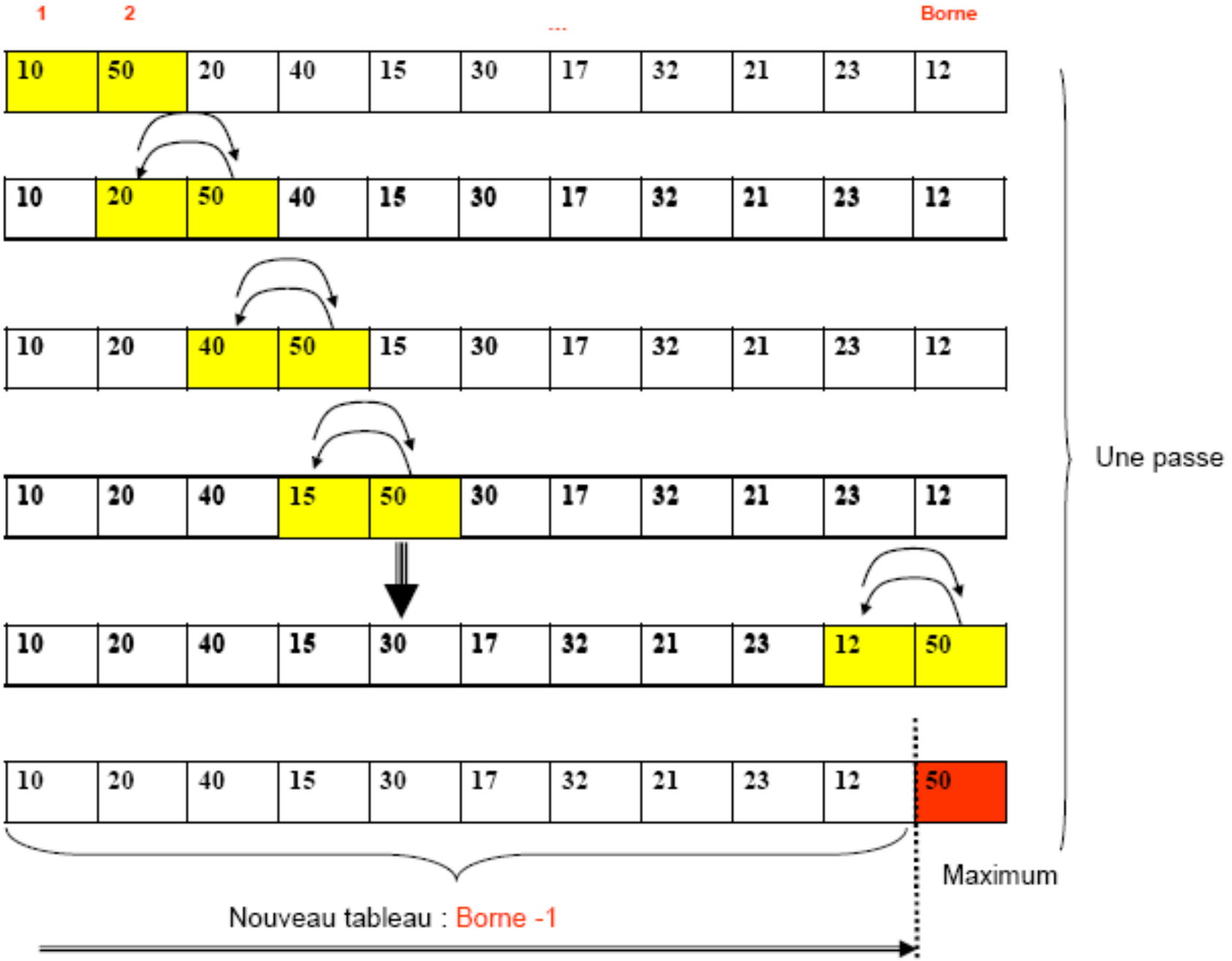
Le tri à bulle consiste à parcourir le tableau, par exemple de gauche à droite, en comparant les éléments côte à côte et en les permutant s' ils ne sont pas dans le bon ordre.

Au cours d'une passe du tableau, les plus grands éléments remontent de proche en proche vers la droite comme des bulles vers la surface.

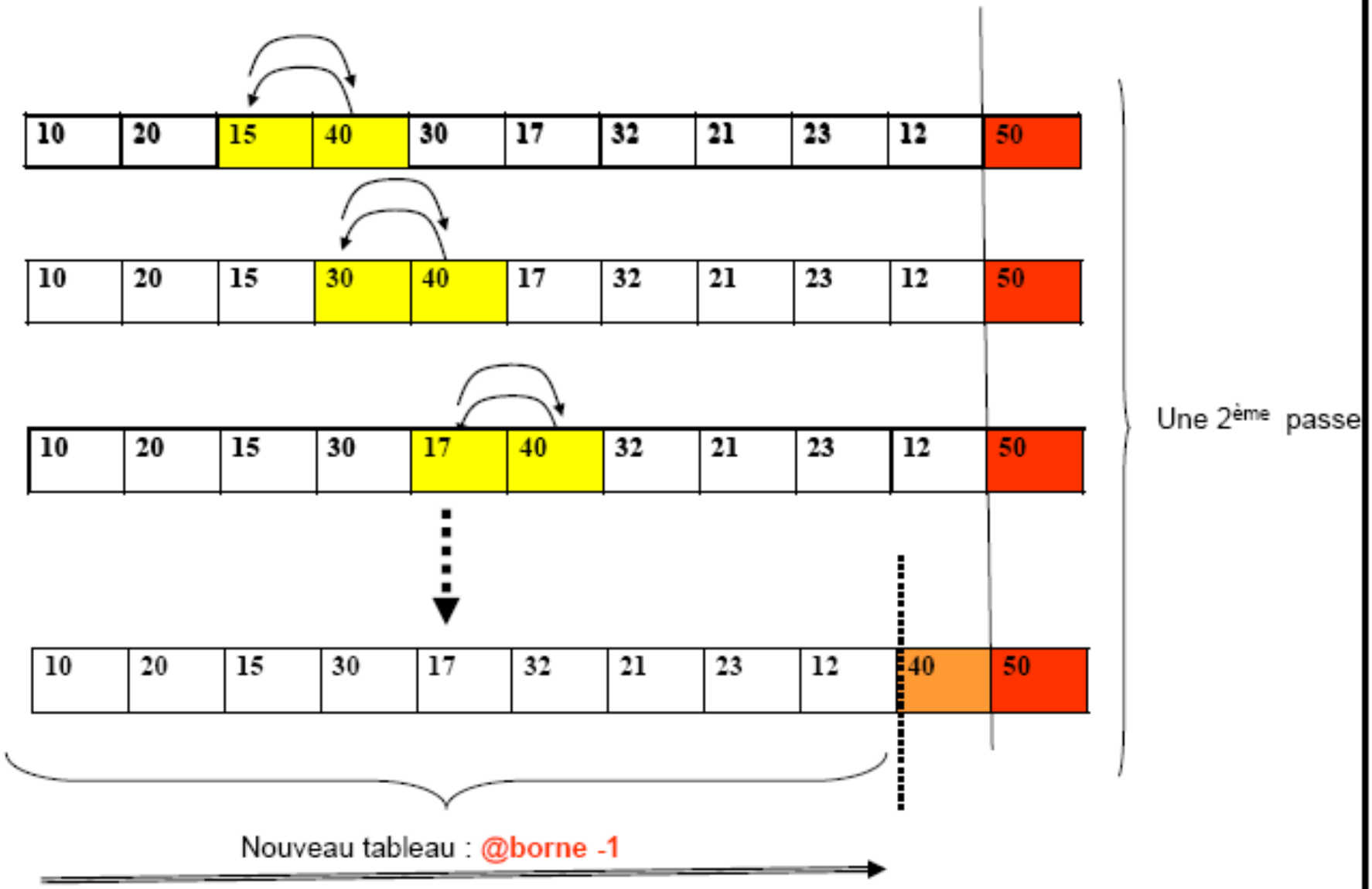
On s'arrête dès que l'on détecte que le tableau est trié : si aucune permutation n'a été faite au cours d'une passe.

Tant que le tableau n'est pas trié
1. Effectuer une passe !

a. Parcourir le tableau
b. Si 2 éléments successifs ne sont pas dans le bon ordre, les échanger


1
2
...
Borne
10 50 20 40 15 30 17 32 21 23 12
10 20 50 40 15 30 17 32 21 23 12
10 20 40 50 15 30 17 32 21 23 12
10 20 40 15 50 30 17 32 21 23 12
10 20 40 15 30 17 32 21 23 12 50
10 20 40 15 30 17 32 21 23 12 50
Une passe
Maximum
Nouveau tableau : Borne -1

Même méthode !! Pour le nouveau tableau de taille borne -1



```
ALGORITHME TriBulle
CONST   TAILLE=50
TYPE    TABL=TABLEAU [1..TAILLE] d'ENTIER
VAR     T          :  TABL
        i,N,borne  :ENTIER
        Est_trie   :BOOLEEN
DEBUT
    ECRIRE(" Entrer le nombre d'élément du tableau < = ",TAILLE)
    LIRE(N)
    SAISIR(T,N)
    Est_trie ⇦ faux                    /*Détecte si le tableau est trié*/
    borne ⇦  N
    TANTQUE Non(Est_trie )             /* Itération sur les passes*/
    FAIRE
        Est_Trie ⇦ vrai                /* Initialisation avant une passe*/
        POUR i DE 1 A borne-1          /* Effectue une passe*/
            Si T[i] > T[i+1]           /* Compare 2 éléments successifs*/
            Alors
                ECHANGER(T,i,i+1)
                Est_trie⇦faux
            FinSi
        FINPOUR
        borne⇦borne-1
    FINTANTQUE
    AFFICHER(T,N)
FIN
```

FIN

ET

MERCI