# INSTITUT SUPÉRIEUR D'ENSEIGNEMENT PROFESSIONNEL DE DIAMNIADIO
# FILIÈRE TIC

## ALGORITHME INTRODUCTION

**2021-2022**

Présenté par: **Abdoulaye MBAYE**

**+221772389823**

ambaye@isepdiamniadio.edu.sn

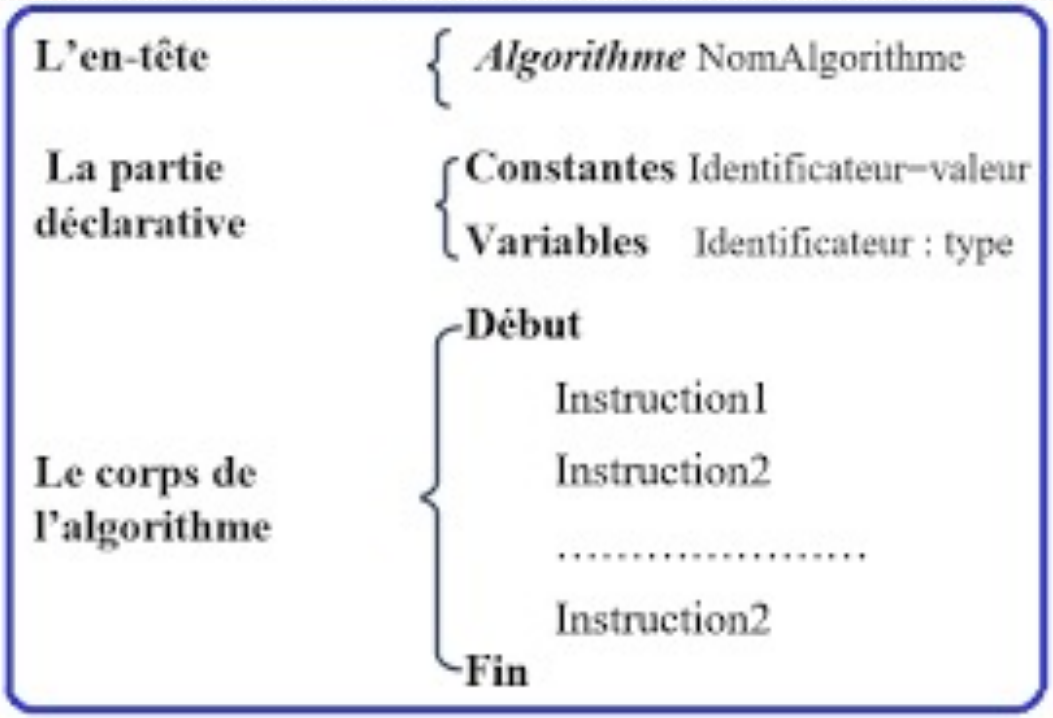
# PLAN

- STRUCTURE D'UN ALGORITHME
- LES STRUCTURES DE CONTRÔLE
  - STRUCTURES ALTERNATIVE

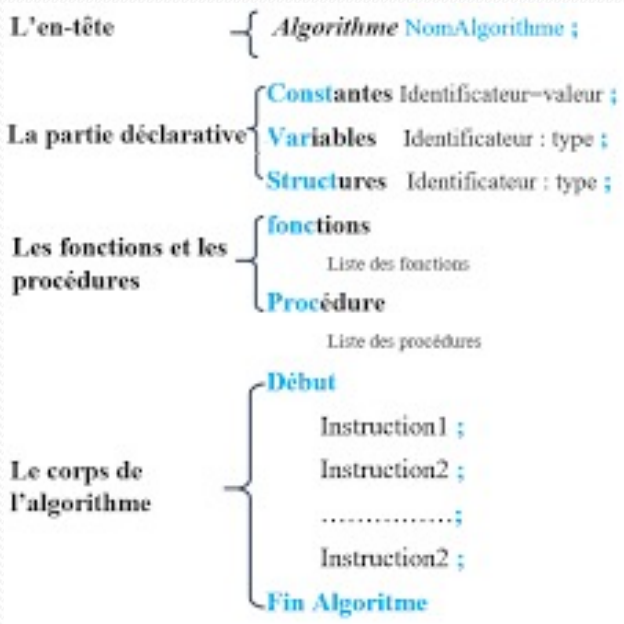
# Structure d'un algorithme

- Un algorithme (comme un programme) est composé de trois parties principales :
  1. La partie **définitions** permet de définir les «objets» qui pourront être manipulées dans l'algorithme. En particulier, on définit des constantes, des types et des sous-programmes.
  2. La partie **déclarations** permet de déclarer les données qui sont utilisées par le programme. Les données sont représentées par des variables.
  3. La partie **instructions** constitue le programme principal. Les instructions sont exécutées par l'ordinateur pour transformer les données.

# Structure d'un algorithme



# Structure d'un algorithme



# Structure d'un algorithme

- **Exemple:**

**Algorithme: calcul**
**Variable**
**A: Entier**
**C, B : Réel**
**D : Chaîne**
**E :Booléen**
**Début**
**A ← 30**
**B← A *2**
**Ecrire** ("B = ", B)
**C← (B+A)/4**
**B← C / 5**
**D←** "Amine"
**E ← (A > 40) OU (C <B)**
**ECRIRE**("Les valeurs obtenues sont : A = ", " B = ", B, " C = ", C, "D = ",D, "E = ", E)
**FIN**

# LES STRUCTURES DE CONTRÔLE

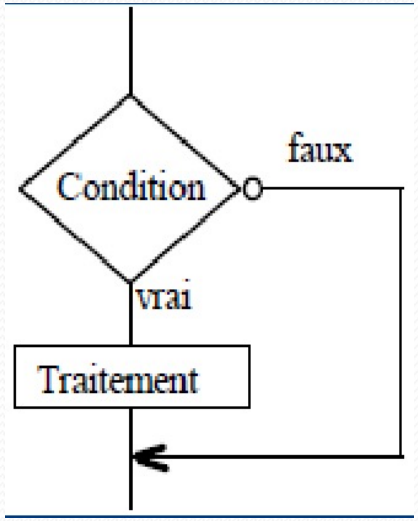
- En programmation impérative, une **structure de contrôle** est une commande qui contrôle l'ordre dans lequel les différentes instructions d'un algorithme ou d'un programme informatique sont exécutées.
- On appelle aussi cet enchaînement d'instructions le flot d'exécution d'un programme.
- Un programme s’arrête généralement après l’exécution de la dernière instruction.

# STRUCTURES ALTERNATIVE CONDITIONNELLE

- La conditionnelle permet d'exécuter une séquence d'instruction, seulement si une condition est vraie.
- Elle permet aussi à un programme de modifier son traitement en fonction d'une condition.
- Il existe trois forme d'instruction conditionnelles:
  - Forme simple
  - Forme généralisée
  - Forme à choix

# CONTRÔLE CONDITIONNELLE SIMPLE

- Une structure de contrôle conditionnelle est dit forme simple réduite lorsque le traitement dépend d'une condition. Si la condition est évaluée à « vraie », le traitement est exécuté.




- **Syntaxe**
  - **Si(expression logique(vaire)) alors**

    **Instruction 1**

    **Instruction 2**

    **....**

    **Instruction N**

    **FinSi**

    **La condition (expression logique) est de type booléen**

# CONTRÔLE CONDITIONNELLE SIMPLE

- **Exemple:** Ecrire un algorithme qui permet de faire la racine carrée d'un réel x.
  - **Algorithme:** racineCarrée

```
Variable  x: réel
Début
    Ecrire("Donner un réel")
    Lire(x)
    Si (x≥0) alors
            Ecrire ("La racine carrée est = ", sqrt(x))
    FinSi
Fin
```

# CONTRÔLE CONDITIONNELLE SIMPLE

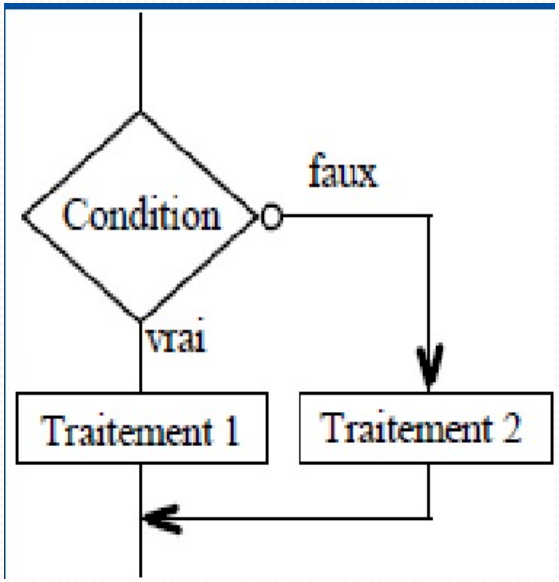
- **Exemple:** Ecrire un algorithme qui permet de vérifier si un entier donné est paire (en utilisant la forme simple).
  - **Algorithme**: parité

```
Variable  n: Entier
          message : Chaine
Début
   Ecrire("Donner un entier")
   Lire(n)
   message← "Pair"
   Si (n mod 2 ≠ 0) alors
          message← "impair"
   FinSi
          Ecrire(n," est nombre ",message)
Fin
```

# CONTRÔLE CONDITIONNELLE SIMPLE

- Une structure de contrôle conditionnelle est dite à forme alternative lorsque le traitement dépend d'une condition à deux état: Si la condition est évaluée à « vraie », le premier traitement est exécuter, si la condition est évaluée à « faux », le second traitement est exécuté.
  - **Syntaxe**:

**Si (condition « vraie ») Alors**
**Instruction 1 de TR1**
**Instruction 2 de TR1**
**.....**
**Instruction m de TR1**
**Sinon**
**Instruction 1 de TR2**
**Instruction 2 de TR2**
**.....**
**Instruction n de TR2**
**FinSi**



# CONTRÔLE CONDITIONNELLE SIMPLE

- **Exemple 2: nombre paire**
- **Algorithme : Parité2**

```
Variable n: entier
Début
        écrire("Donner un entier ")
        lire (n)
        si n mod 2 = 0 alors
                écrire(n," est pair")
        sinon
                écrire(n," est impair")
        FinSi
Fin
```

# CONTRÔLE CONDITIONNELLE GENERALISEE

- Une structure de contrôle conditionnelle est dite généralisée lorsqu'elle permet de résoudre des problèmes comportant plus de deux traitements en fonction des conditions. L'exécution d'un traitement entraîne automatiquement la non-exécution des autres traitements.
- **Syntaxe:**

```
Si condition 1 Alors
        Traitement 1
Sinon Si condition 2 Alors
        traitement 2
 Sinon Si condition 3 Alors
        traitement 3
        .....
Sinon Si condition N-1 Alors
        traitement N-1
Sinon
        traitement N
FinSi
```

# CONTRÔLE CONDITIONNELLE GENERALISEE

- **Remarque** :
  - Il est préférable de mettre les événements les plus probables en premier lieu.
  - Chaque traitement peut comporter une ou plusieurs instructions.
- **Exemple:** Écrire un programme qui permet de faire la résolution de l'équation du 1er degré : ax+b = 0

# CONTRÔLE CONDITIONNELLE GENERALISEE

- Algorithme: Equation

```
Variable a, b, S: réel
Début
        Ecrire (" Donner  a : ")
        lire(a)
        Ecrire ("Donner b : ")
        lire(b)
        Si (a ≠ 0) alors
                S← -b/a
                Ecrire ("La solution est ", S)
        Sinon
            si (b=0) alors
                Ecrire("La solution est IR ")
            sinon
                Ecrire("La solution est vide ")
            Finsi
        FinSi
Fin
```

# CONTRÔLE CONDITIONNELLE GENERALISEE

- **Exercice 1**:

Écrire un programme qui permet de saisir une moyenne (moy) puis affiche la décision correspondante :

| | |
|---|---|
| ADMIS | Si moy >=10 |
| CONTROLE | Si 9<=moy <10 |
| REDOUBLE | Si moy<9 |

- **Exercie 2:**

  Écrire un programme qui permet de saisir un temps (heure et minute) lui ajoute 5minutes puis l'affiche.

Exemples :

| | | |
|---|---|---|
| heure : 10 | heure : 10 | heure : 23 |
| minute : 20 | minute:57 | minute:55 |
| après 5 minutes : **10:25** | après 5 minutes : **11: 02** | après 5 minutes : **00:00** |

**Remarque :**

Pour afficher l'heure et minute correctement ( ajouter un zéro à gauche des chiffres <10 , exemple 05:08 )

# CONTRÔLE CONDITIONNELLE GENERALISEE

- Corrigé :**Exercice 1**
- **Pré-analyse**
  - **But : afficher la décision**
  - **données : moy**
  - **formules : structure si sur moy**

**Algorithme: décision**
**Variable :** Moy: reel
message: chaine
**Début**
**Écrire("Donner une moyenne")**
**lire(Moy)**

**si (moy >= 10) alors**
**message ← "ADMIS"**
**Sinon**
**Si (moy >= 9 )alors**
**message ← "CONTROLE"**
**Sinon**
**message ← "REDOUBLE"**
**FinSi**
**Finsi**
**Ecrire(message)**
**Fin**

# CONTRÔLE CONDITIONNELLE GENERALISEE

**Algorithme : plus5min**

**Variable** hh, mm : entier

**Début**

```
Ecrire("Heure: ")
lire(hh)
Ecrire("Minute: ")
lire(mm)

Si (mm < 55)  alors
        mm ← mm+5
sinon
        si (hh < 23) alors
                hh ← hh+1
                mm ← mm-55
        sinon
                hh ← 0
                mm ← mm-55
        Finsi
FinSi
Ecrire("Après 5 min : ", hh, " :", mm)
Fin
```

# CONTRÔLE CONDITIONNELLE À CHOIX

- Une structure de contrôle conditionnelle est dite à choix lorsque le traitement dépend de la valeur que prendra le sélecteur, ce sélecteur doit être de type scalaire (entier ou caractère).

**Syntaxe:**

```
selon (sélecteur) Faire
Valeur 1  :  Action 1
Valeur 2  :  Action 2-1
             Action 2-2
             Action 2-n
Valeur 3  :  Action 3
Valeur4,valeur 6, valeur 8: Action4
Valeur5,valeur 7,valeur 9 : Action 5
Valeur 10 .. Valeur 19 : Action 6
          .....
Valeur N  :  Action N
Sinon
    Action R
FinSelon
```

# CONTRÔLE CONDITIONNELLE À CHOIX

- **Remarques** : Pour le choix
  - Sinon est facultative
  - Un traitement qui comporte plusieurs instructions doit être délimité par début et Fin
  - Les valeurs du sélecteur sont de type scalaire(entier , booléen, caractère) ou type intervalle(entier, caractère)

- **Exercice 1:**

  Écrire un programme qui permet de saisir le n° de mois puis affiche la saison correspondante.

  Exemple : n°mois=7 affiche été

  12,1,2 saison hiver
  3,4,5 saison printemps
  6,7,8 saison été
  9,10,11 saison automne

# CONTRÔLE CONDITIONNELLE À CHOIX

- **Corrigé :**

**Algorithme : Saison**
**Variable mois: entier** //*mois à saisir*
**M: chaine** //*Contient le message à afficher*

**Début**
**Ecrire("Donner le n° de mois")**
**lire(mois)**
**Selon (mois) faire**
**12,1,2 : M ← "saison hiver"**
**3..5 : M ← "saison printemps"**
**6..8 : M ← "saison été"**
**9..11 : M ← "saison automne"**
**sinon M ← "Erreur"**
**FinSelon**
**Écrire(M)**
Fin

# CONTRÔLE CONDITIONNELLE À CHOIX

- **Exercice 2:**

  Écrire un programme intitulé TOUCHE, qui affiche selon le cas, la nature du caractère(consonne, voyelle, chiffre ou symbole) correspondant à une touche saisie. On considère que le clavier est verrouillé en minuscule.

- **Exercice 3**

  Écrire un programme qui permet d'afficher le nombre de jour d'un mois donné.

**NB:** Une année bissextile est une année comptant 366 jours au lieu de 365, c'est-à-dire une année comprenant un 29 février (exemple 2012) sont bissextiles les années:

divisibles par 4 mais non divisibles par 100

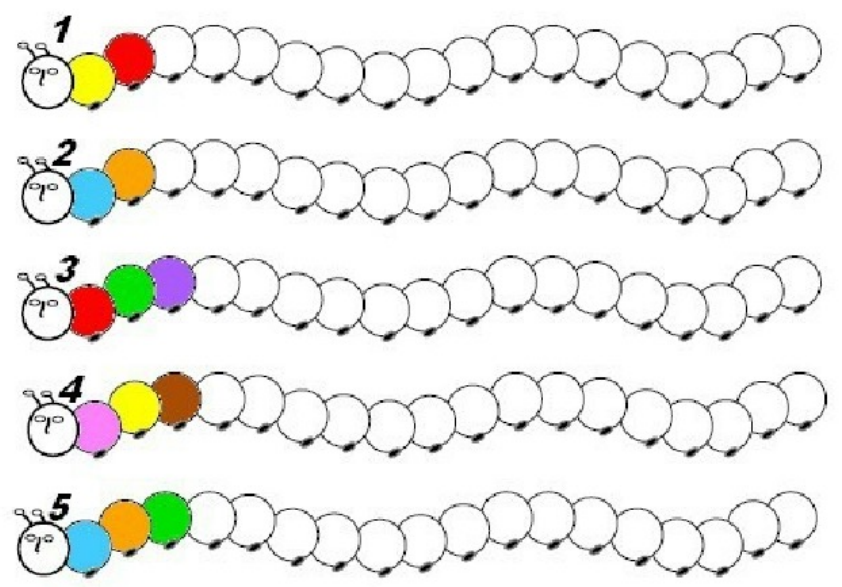
divisibles par 400.

**Correction RV au TD 3**

# STRUCTURES DE CONTRÔLE ITÉRATIVES

- **Exemple et rappel:**
  - Nous dresserons ici une activité algorithmique vue en maternelle et à école élémentaire.
  - Historiquement, les algorithmes sont très présents à l'école élémentaire. Cependant de nombreuses activités estampillées « algorithme » sont le plus souvent des « suites logiques » répondant à une action de mimétisme

# STRUCTURES DE CONTRÔLE ITÉRATIVES




Tant que (boule et boule+1 ne sont pas colorées) :

Colorer boule en jaune

Colorer boule+1 en rouge

boule ← boule+2

# STRUCTURES DE CONTRÔLE ITÉRATIVES

- En maternelle, il n'est pas rare de trouver des activités de tissage dans lesquelles l'élève doit « passer dessus » puis « passer dessous » de manière répétée : tissage avec des bandes de papier, avec un fil pour suivre les contours d'un personnage dessiné…

# STRUCTURES DE CONTRÔLE ITÉRATIVES

position ← dessous
Tant que (colonne est vide) :
    Tant que (ligne est vide) :
        Passer bande à position
        Si (position← dessous)
    alors :
            position ←dessus
        Sinon :
            position←dessous
        ligne← ligne + 1
    colonne← colonne + 1

# STRUCTURES DE CONTRÔLE ITÉRATIVES

- Enfin, nous pouvons également faire mention des suites algorithmiques gestuelles ou « rythmes » : on alterne x frappes des mains, y frappes des pieds…
- **Bilan:**
- **Les algorithmes vus jusqu'à lors servent donc trois principaux objectifs : la construction de la répétition et celle de la classification.**

# STRUCTURES DE CONTRÔLE ITÉRATIVES

- **Introduction:**
- Ecrire un algorithme intitulé message permettant d'afficher 25 fois le message « Bonjour »
  - On constate que l'algorithme MESSAGE contient une répétition.
  - Si nous voulions afficher dix milles « Bonjour »?
  - Pour éviter ce problème on utilise une nouvelle structure appelée structure itérative ou répétitive.

# STRUCTURES DE CONTRÔLE ITÉRATIVES

- Une structure de contrôle itérative permet à l'ordinateur de répéter l'exécution d'un ensemble d'instructions ou d'un traitement un nombre fini de fois pas forcement connu à l'avance.
- On distingue deux structures :
  - Structure Itérative complète.
  - Structure Itérative à condition d'arrêt.

# STRUCTURE ITÉRATIVE COMPLÈTE:

## La boucle pour …. Répéter

- Une structure répétitive est dite complète si le nombre de répétition est connu d'avance.
- **Syntaxe:**

<objet> ← [<init>]

**POUR** <**compteur**> ← <**valeur initiale**> **à** <**valeur finale**> [(pas ← <nb>)]

**Faire** ou bien **Répéter**

< instruction1 >

< instruction2 >

< instruction3 >

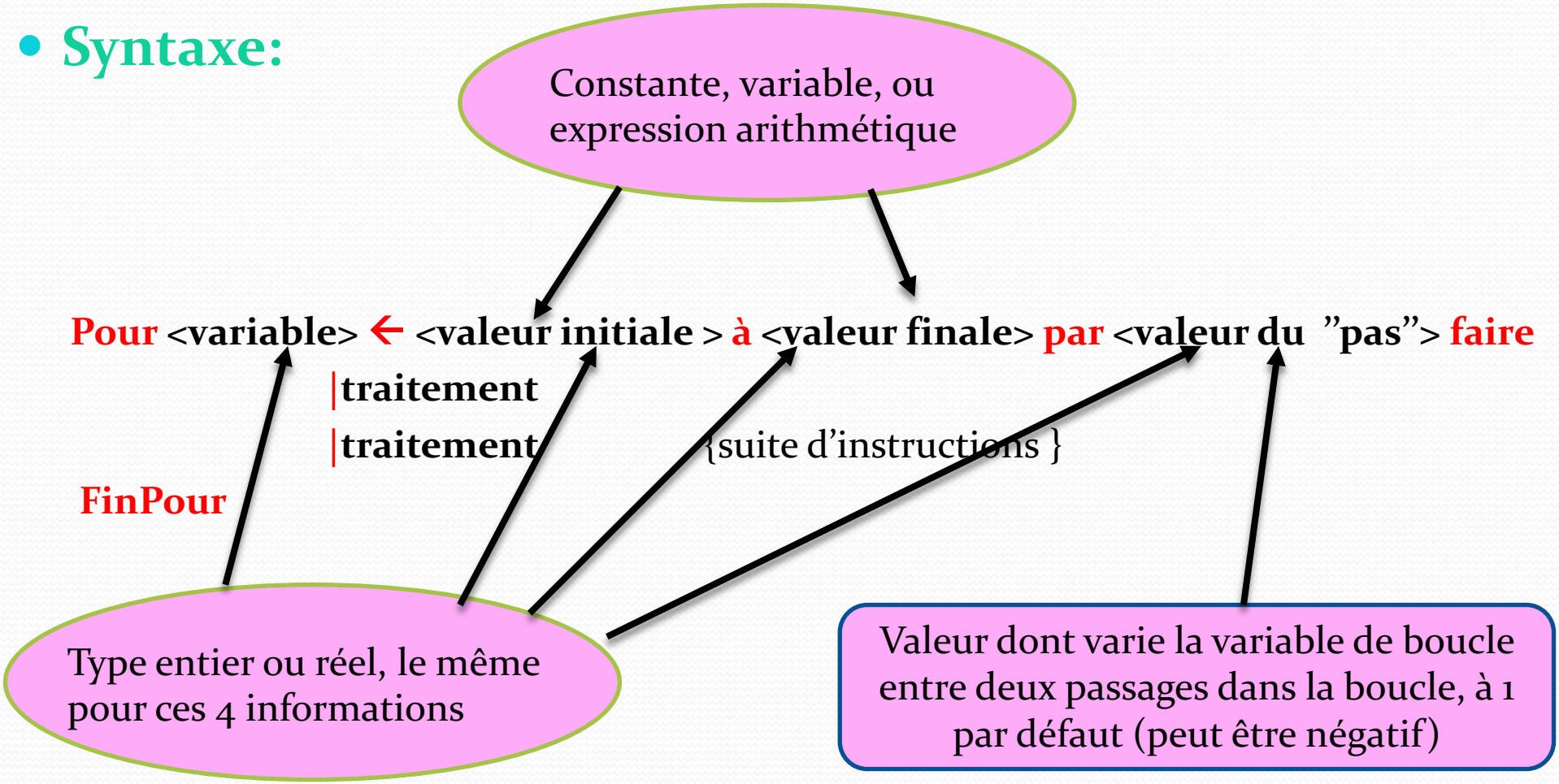
……..

< instructionN >

**FINPOUR**

# STRUCTURE ITÉRATIVE COMPLÈTE: La boucle pour …. Répéter

- **Syntaxe:**



# STRUCTURE ITÉRATIVE COMPLÈTE:
## La boucle pour …. Répéter

- **Remarque:**
  - Le compteur doit être une variable de type scalaire. (entier et caractère)
  - La structure itérative complète est utilisée lorsque le nombre de répétition est connu d'avance
  - **Valeur initiale** : Valeur initial du compteur.
  - **Valeur finale** : Valeur du compteur lors de la dernière exécution.
  - **Valeur initial** et **Valeur finale** doivent être des expressions de même type que le compteur.
  - Si le **pas** est égal à 1 il n'est pas nécessaire de l'indiquer.
  - Le nombre de répétition = | Valeur finale –Valeur initiale | +1.

# STRUCTURE ITÉRATIVE COMPLÈTE:
## La boucle pour …. Répéter

- Sémantique de la boucle pour
  - Implicitement, l'instruction pour:
    - Initialise une variable de boucle (le compteur)
    - Incrément cette variable à chaque pas
    - Vérifie que cette variable ne dépasse pas la borne supérieur
  - Attention:
    - Le traitement ne doit pas modifier la variable de boucle

**pour** cpt ← 1 à Max **faire**

**si**( .. ) **alors** cpt ← Max **interdit!**

**FinPour**

# STRUCTURE ITÉRATIVE COMPLÈTE:
## La boucle pour …. Répéter

- **Exemple:**

```
Algorithme FaitLeTotal
        {cet algorithme fait la somme des nbVal données qu'il saisit}
        Variable    nbVal, cpt: entier
                    Valeur, totalValeurs: réel
        Debut
            {initialisation du traitement}
                    Ecrire( "Combien de valeurs voulez-vous saisir?")
                    Lire(nbVal)
                        {initialisation du total à 0 avant cumul}
                    totalValeurs ← 0
                        {traitement qui se répète nbVal fois}
                    Pour cpt ← 1 à nbVal faire
                                Ecrire("Donner une valeur: ")
                                Lire(valeur)
                                totalValeurs ← totalValeurs +valeur     {cumul}
                    FinPour
                       {édition des résultats}
                    Ecrire("Le total des ", nbVal, "valeurs est ", totalValeurs)
        Fin
```

# STRUCTURE ITÉRATIVE COMPLÈTE:

## La boucle pour …. Répéter

- **Activité 1:**
- Ecrire un algorithme en pseudo-code qui détermine et affiche le factoriel de n'importe quel entier donné par l'utilisateur.
  - 5mn de réflexion
  - Correction au tableau
- **Activité 2:**
- Ecrire un algorithme en pseudo-code qui calcul et affiche la somme de 5 entiers saisi au clavier par l'utilisateur.
  - 5mn de réflexion
  - Correction au tableau

# Les structures itératives à condition d'arrêt

- Présentation :
  - Pour certains cas de traitement, le nombre de répétition n'est pas connu à l'avance (mais la condition d'arrêt est connue). Pour cela il faut utiliser cette condition d'arrêt pour arrêter le traitement répétitif.
    - Remarque :
      - Malgré que le nombre de répétition n'est pas connu à l'avance, on peut savoir si le traitement à réaliser peut se faire au moins une ou zéro fois. C'est la principale différence entre les deux structures itératives à condition d'arrêt:
    - **REPETER … JUSQU'A et TANTQUE … FAIRE**
- Définition :
  - La structure itérative à condition d'arrêt permet de répéter un traitement donné et que l'arrêt soit géré par une condition.

# Les structures itératives à condition d'arrêt

# TANTQUE … FAIRE

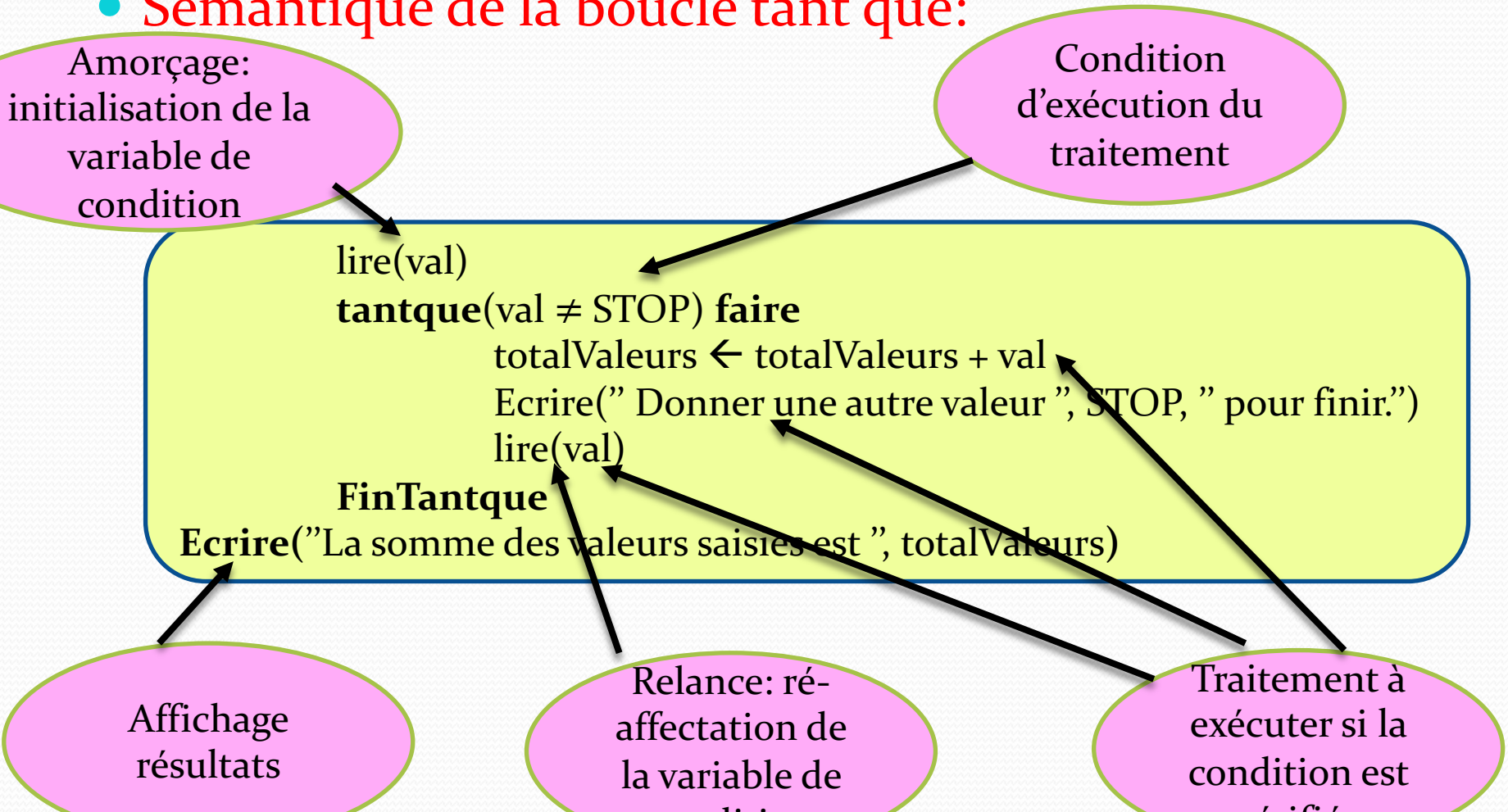
- **Syntaxe:**

```
Amorçage                    {initialisation de la (des) variable(s) de condition}
tantque(<expression logique>(vraie)) faire
    |traitement             {suite d'instructions}
    |relance                {réaffectation de la (des) variable(s) de condition}
FinTantque
```

- **Fonction:**
  - Répéter une suite d'instruction tant qu'une condition est remplie.
  - **Remarque:** si la condition est fausse dés le départ, le traitement n'est jamais exécuté.

# Les structures itératives à condition d'arrêt
# TANTQUE … FAIRE

- Sémantique de la boucle tant que:



# Comparaison boucles pour et tantque

```
Pour cpt ← 1 à nbVal faire
        Ecrire(”Donner une valeur: ”)
        Lire(valeur)
        totalValeurs ← totalValeurs +valeur      {cumul}
FinPour
```

… équivaut à :

```
cpt ← 0
tantque( cpt < nbVal) faire
        Ecrire(”Donner une valeur: ”)
        Lire(valeur)
        totalValeurs ← totalValeurs + val     {cumul}
        cpt ← cpt + 1    {compte le nombre de valeur traitées}
Fintantque
```

# Comparaison boucles pour et tantque

- Implicitement, l'instruction pour:
  - Initialise un compteur
  - Incrémente le compteur à chaque pas
  - Vérifie que le compteur ne dépasse pas la borne supérieure
- Explicitement, l'instruction tantque doit
  - Initialiser un compteur {amorçage}
  - Incrémenter  le compteur à chaque pas {relance}
  - Vérifier que le compteur ne dépasse pas la borne supérieure {test de boucle}

# Choisir pour … Choisir tant que

- Si le nombre d'itérations est connu à l'avance,
  - CHOISIR LA BOUCLE pour

- Si la boucle doit s'arreter quand survient un évènement,
  - CHOISIR LA BOUCLE tantque

# Les structures itératives à condition d'arrêt répéter … tantque

- Exemple

Algorithme Essai

{cet algorithme a besoin d'une valeur positive paire}

**Variable** valeur: **entier**

**Début**

**Répéter**

**Ecrire**("Donner une valeur positive non nulle : ")

**Lire**(valeur)

**Tantque** (valeur ≤ 0 )

**Ecrire**("La valeur positive non nulle que vous avez saisie est")

**Ecrire**(valeur) {traitement de la valeur saisie}

**Fin**

# Les structures itératives à condition d'arrêt

## répéter … tantque

- **Syntaxe :**
- **répéter**
  (ré)affectation de la (des) variable(s) de condition
  traitement {suite d'instructions}
  **tantque**( <expression logique (vraie)>)
- **Fonction:**
  - Exécuter une suite d'instructions au moins une fois et la répéter tant qu'une condition est remplie.
  - **Remarque:**
    - le traitement dans l'exemple précédent se limite à la ré-affectation de la variable de condition.

# Comparaison boucles répéter et tantque

```
répéter
        Ecrire(”Donner une valeur positive paire: ”)
        lire(valeur)
tantque(  valeur< 0 ou (valeur mod 2) ≠ 0)
```

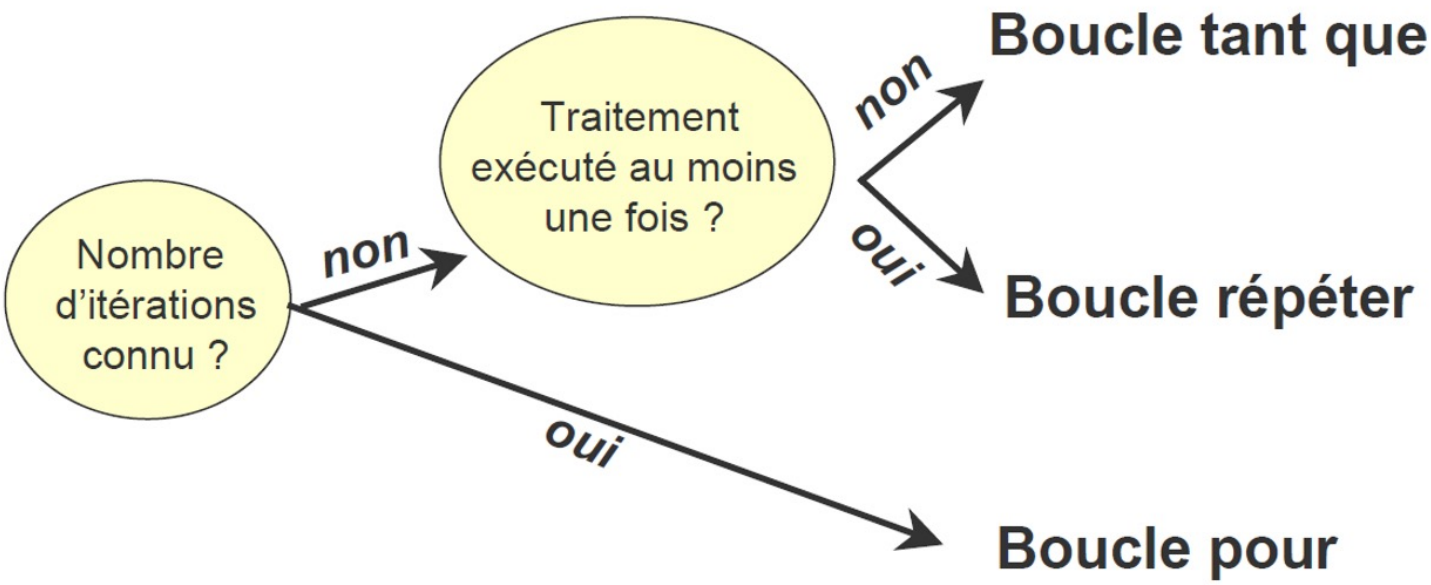
```
Ecrire(” Donner une valeur positive paire :”)
Lire(valeur)
tantque(valeur< 0 ou (valeur mod 2) ≠ 0) faire
        Ecrire(”Donner une valeur positive paire: ”)
        lire(valeur)
Fintantque
```

# Comparaison boucles répéter et tantque

- **Boucle tant que**
  - Condition vérifiée avant chaque exécution du traitement
  - Le traitement peut donc ne pas etre exécuté
  - De plus: la condition porte surtout sur la saisie de nouvelles variables(relance)
- Boucle répéter tant que
  - Condition vérifiée apres chaque exécution du traitement
  - Le traitement est exécuter au moins une fois
  - De plus: la condition porte surtout sur le résultat du traitement

**Remarque** : la boucle répéter est typique pour les saisies avec vérification.

# Choisir pour  tant que  répéter



# Le problème d'une boucle: il faut en sortir!

- Tant que A faire B
- Répéter B tant que A
- Quelque chose dans la suite d'instruction B doit amener A à prendre la valeur Faux.
- La suite d'instruction B doit modifier au moins une variable de l'expression logique A
- (mauvais)exemple
  - Val1 ←2
  - Val2← 3
  - Tant que (val1<100) faire
  - Val2←val2*val1
  - Fintantque
- C'est l'expression logique A (et elle seule!) qui en prenant la valeur Faux provoque l'arret de la boucle.

# De l'énoncé à la boucle

- Afficher le carré des valeurs saisies tant qu'on ne saisit pas 0

```
Lire(val)
Tantque(val ≠ 0) faire
        Ecrire(val*val)
        Lire(val)
Fintantque
```

- Saisir des données et s'arrêter dés que leur somme dépasse 500

```
Lire(val)
Somme ← val
Tantque(somme ≤ 500) faire
        Lire(val)
        somme ← somme +val
Fintantque
```

# De l'énoncé à la boucle

- Saisir des données et s'arrêter dés que leur somme dépasse 500

```
Lire(val)
Somme ← 0
répéter
        Lire(val)
        somme ← somme +val
Tantque(somme ≤ 500)
```

- Saisir des données tant que leur somme ne dépasse un seuil donné

# De l'énoncé à la boucle

- Exemple d'un mauvais choix de boucle
- Algorithme Somme
- {cet algorithme fait la somme d'une suite de nombres tantque cette somme ne dépasse un seuil donné}
- **Constante** (SEUIL : entier)← 1000
- **Variable** val, somme: entier
- **début**

  Somme ←0

  **Répéter**

  **Ecrire**("Entrez un nombre")

  **Lire**(val)

  Somme ← somme +val

  **tantque**(somme ≤ SEUIL)

  **Ecrire**("La somme atteinte est ", somme – val)
- **Fin**

# De l'énoncé à la boucle

- Version Corrigée
- Algorithme Somme
- {cet algorithme fait la somme d'une suite de nombres tantque cette somme ne dépasse un seuil donné}
- Constante (SEUIL : entier)← 1000
- Variable  val, somme: entier
- début

# Quand utiliser la boucle tantque

- Structure itérative ”universelle”
  - n’importe quel contrôle d’itération peut se traduire par le ”tantque”
- Structure itérative irremplaçable dés que la condition d’itération devient complexe
- Exemple:
  - Saisir des valeurs, les traiter, et s’arrêter a la saisie de la valeur d’arrêt -1 ou après avoir saisi 5 données.

# Quand utiliser la boucle tantque

- Exemple

```
Algorithme: TestArret
Constante       (STOP : entier) ← -1
                (Max : entier) ← 5
Variable        nbVal, val: entier
Début
        nbVal←0                 {compte les saisie traitées}
        lire(val)               {saisie de la 1ére donnée}
        tantque(val ≠ STOP et nbVal < Max) faire
                nbVal ← nbVal + 1
                                {traitement de la valeur saisie}
                lire(val)       {relance}
        FinTantque
Ecrire(val, nbVal)              {valeurs en sortie de boucle}
```

- Attention:
  - la valeur d'arrêt n'est jamais traitée (et donc, jamais comptabilisée)

# Quand utiliser la boucle tantque

- Interpréter l'arrêt des itérations

nbVal←0 **{compte les saisie traitées}**
**lire**(val) **{saisie de la 1ére donnée}**
**tantque**(val ≠ STOP et nbVal < Max) **faire**
nbVal ← nbVal + 1
**{traitement de la valeur saisie}**
**lire**(val) **{relance}**
**FinTantque**
**si**(val = STOP) **alors**
{la dernière valeur testée était la valeur d'arrêt}
**Ecrire**("Sortie de boucle car saisie de la valeur d'arrêt; toutes les données significatives ont été traitées")
**Sinon** {il y'avait plus de 5 valeurs à tester}
**Ecrire**("Sortie de boucle car nombre maximum de valeurs à traiter atteint; des données significatives n'ont pas pu été traitées")
**Finsi**

# L'importance du test de sortie de boucle

- **La logique:**

**tantque**(val ≠ STOP et nbVal < Max) **faire**

- Dans la boucle : val ≠ STOP et nbVal < Max est vrai
- à la sortie de boucle:
  - Soit val ≠ STOP est faux → val = STOP
  - Soit nbVal < Max est faux → nbVal ≥ Max
- Que tester à la sortie de boucle?
  - Si val = STOP alors … voir transparent précédent
  - Si nbVal ≥ Max alors … mauvais test car message dépend de la dernière valeur saisie.

# Conclusion:
# Quelques leçons à retenir

- **Le moule d'un algorithme**

```
Algorithme  AuNomEvocateur
{Cet algorithme fait.....en utilisant telle et telle donnée.....}
constantes
variables
début
        {préparation du traitement : saisies,....}
        {traitements, si itération, la décrire }
        {présentation des résultats: affichages,... }
fin
```

- Il faut avoir une **écriture rigoureuse**
  Il faut avoir une écriture soignée : respecter **l'indentation**
  Il est nécessaire de **commenter** les algorithmes

- **Il existe plusieurs solutions algorithmiques à un problème posé**
  - Il faut rechercher **l'efficacité** de ce que l'on écrit