

Rappel Algorithmique

- Introduction
- Instructions de base
- Logique propositionnelle

Souhila.GUERFI
2015-2016

Pourquoi un cours d' "Algo" ?

- **Objectif** : obtenir de la «machine» qu'elle effectue un travail à notre place
- **Problème** : expliquer à la «machine» comment elle doit s'y prendre

Mais... comment le lui dire ?

Comment le lui apprendre ?

Comment s'assurer qu'elle fait ce travail aussi bien que nous ?

Mieux que nous?

Objectif de cet enseignement

- résoudre des problèmes «comme» une machine
- savoir ***expliciter*** son raisonnement
- savoir ***formaliser*** son raisonnement
- concevoir (et écrire) des ***algorithmes*** :
 - séquence d'instructions qui décrit comment résoudre un problème particulier

Thèmes abordés en «Algo»

- Apprentissage d'un langage
- Notions de base
 - algorithmes de « base » pour problèmes élémentaires
- Structures de données
 - des plus simples aux plus complexes
- Résolution de problèmes complexes
 - algorithmes astucieux et efficaces

L'algorithme, vous la pratiquez tous les jours et depuis longtemps...

Briques de LEGO

→
suite de dessins

Camion de pompiers

Meuble en kit

→
notice de montage

Cuisine équipée

Cafetière

→
instructions

Expresso

Laine

→
modèle

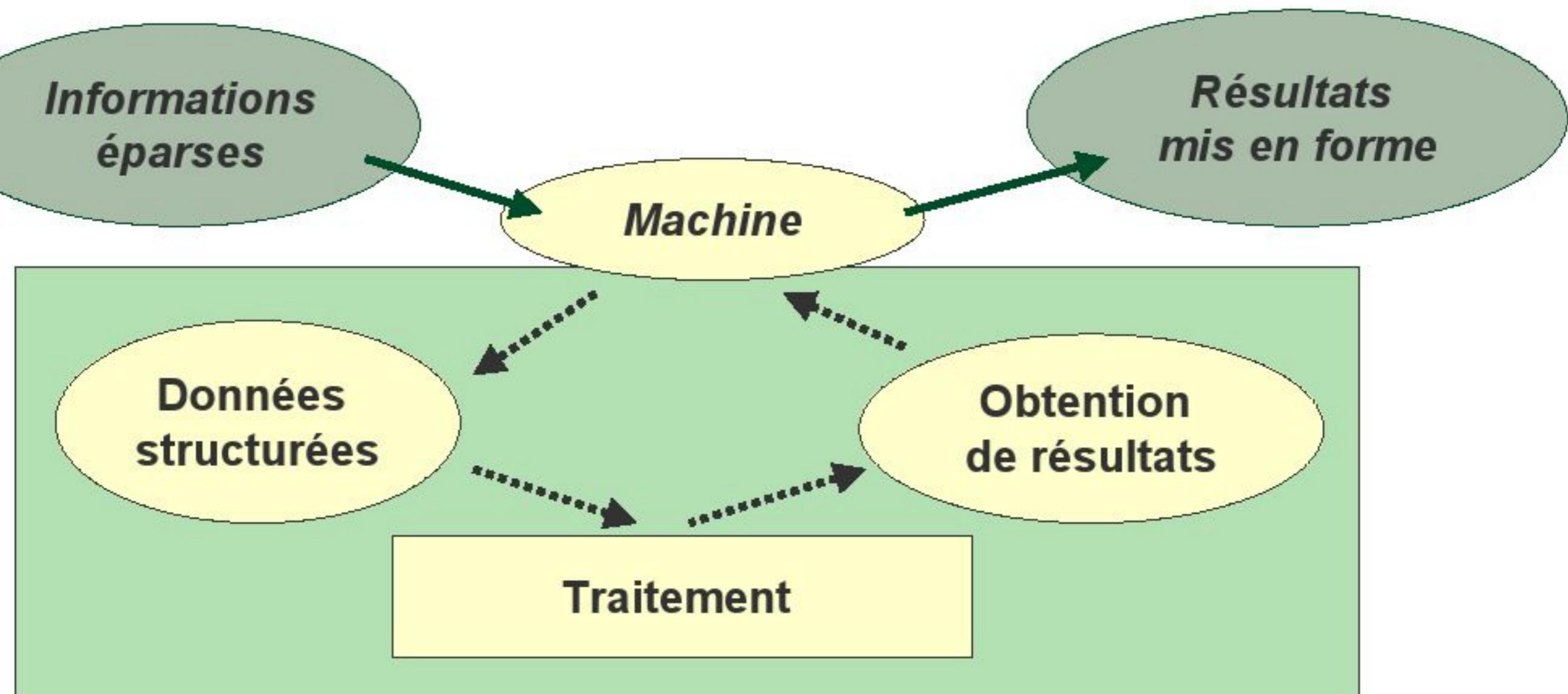
Pull irlandais

Farine, oeufs, chocolat, etc....

→
recette

Forêt noire

De l'importance de l'algorithme



Un **algorithme**, traduit dans un langage compréhensible par l'ordinateur (ou langage de programmation, ici le C++), donne un **programme**, qui peut ensuite être exécuté, pour effectuer le **traitement** souhaité.

- Savoir expliquer comment faire un travail sans la moindre ambiguïté
 - langage simple : des instructions (pas élémentaires)
 - suite finie d'actions à entreprendre en respectant une chronologie imposée
- L'écriture algorithmique : un travail de programmation à visée «universelle»
 - un algorithme ne dépend pas
 - du *langage* dans lequel il est implanté,
 - ni de la *machine* qui exécutera le programme correspondant.

Les problèmes fondamentaux en algorithmique

- **Complexité**
 - En combien de temps un algorithme va-t-il atteindre le résultat escompté?
 - De quel espace a-t-il besoin?
- **Calculabilité :**
 - Existe-t-il des tâches pour lesquelles il n'existe aucun algorithme ?
 - Etant donnée une tâche, peut-on dire s'il existe un algorithme qui la résolve ?
- **Correction**
 - Peut-on être sûr qu'un algorithme répond au problème pour lequel il a été conçu?

Les instructions de base



Un premier algorithme

Algorithme ElèveAuCarré

{*Cet algorithme calcule le carré du nombre que lui fournit l'utilisateur*}

variables unNombre, sonCarré: entiers

{*déclarations: réservation d'espace-mémoire*}

début

{*préparation du traitement*}

afficher("Quel nombre voulez-vous éléver au carré?")

saisir(unNombre)

{*traitement : calcul du carré*}

sonCarré \leftarrow unNombre \times unNombre

{*présentation du résultat*}

afficher("Le carré de ", unNombre)

afficher("c'est ", sonCarré)

fin

Les trois étapes d'un algorithme

- Préparation du traitement
 - données nécessaires à la résolution du problème
- Traitement
 - résolution pas à pas, après décomposition en sous-problèmes si nécessaire
- Edition des résultats
 - impression à l'écran, dans un fichier, etc.

Déclarer une variable

variable <liste d'identificateurs> : type

- **Fonction :**

Instruction permettant de réserver de l'espace mémoire pour stocker des données (dépend du type de ces données : entiers, réels, caractères, etc.)

- **Exemples :**

variables val, unNombre : **entiers**
 nom, prénom : **chaînes de caractères**

Saisir une donnée

saisir(<liste de noms de variables>)

- Fonction :
Instruction permettant de placer en mémoire les informations fournies par l'utilisateur.
- Exemples:
saisir(unNombre)
saisir(nom, prénom)
saisir(val)

Afficher une donnée, un résultat

afficher(<liste de noms de variables, de constantes ou d 'expressions>)

- **Fonction :**

Instruction permettant de visualiser les informations placées en mémoire.

- **Exemples:**

afficher(unNombre, "est différent de 0")

**afficher("La somme de", unNombre, "et" , val , "est",
unNombre + val)**

Déclarer une constante

constante (<identificateur> : type) \leftarrow <expression>

- **Fonction :**

Instruction permettant de réserver de l'espace mémoire pour stocker des données dont la valeur est fixée pour tout l'algorithme

- **Exemples :**

constantes (MAX : entier) \leftarrow 100
 (DOUBLEMAX : entier) \leftarrow MAX \times 2

Saisies et affichages : exemples

Algorithme ParExemple

{Saisit un prix HT et affiche le prix TTC correspondant}

constantes (TVA : réel) \leftarrow 20.6

(Titre : chaîne) \leftarrow "Résultat"

variables prixHT, prixTTC : **réels** {déclarations}

début

{préparation du traitement}

afficher("Donnez-moi le prix hors taxe :")

saisir(prixHT)

prixTTC \leftarrow prixHT * (1+TVA/100)

{calcul du prix TTC}

afficher(Titre)

{présentation du résultat}

afficher(prixHT, « euros H.T. devient », prixTTC, « euros T.T.C. »)

Fin

Affichage :

Affecter une valeur à une variable

```
<identificateur> ← <expression> ou  
<constante> ou <identificateur>
```

- **Fonction :**

Instruction permettant d'attribuer à la variable identifiée par l'élément placé à gauche du symbole \leftarrow la valeur de l'élément placé à droite de ce symbole.

- **Exemple:**

nom \leftarrow "Venus"

val \leftarrow 50

val \leftarrow val \times 2

Affectation : exemples

constante
variables

(SEUIL : réel) $\leftarrow 13.25$
valA, valB : réels
compteur : entier
mot , tom : chaînes

valA $\leftarrow 0.56$

valB \leftarrow valA

valA \leftarrow valA $\times (10.5 + \text{SEUIL})$

compteur $\leftarrow 1$

compteur \leftarrow compteur + 10

mot \leftarrow " Bonjour "

tom \leftarrow "Au revoir ! "

tableau de simulation :

valA	valB	compteur	mot	tom

Affectation : exemples (suite)

```
afficher(mot)
afficher(" valA = ", valA)
afficher(" valB = ", valB)
afficher(" compteur =", compteur )
afficher(tom)
```

Affichage :

Simulation d'un algorithme

Algorithme CaFaitQuoi?

{Cet algorithme

variables valA, valB : réels

{déclarations}

début

{préparation du traitement}

afficher("Donnez-moi deux valeurs :")

saisir (valA, valB)

afficher("Vous m'avez donné ", valA, " et ", valB)

{traitement mystère}

valA \leftarrow valB

valB \leftarrow valA

{présentation du résultat}

afficher("Maintenant , mes données sont : ", valA, " et ", valB)

Fin

Affichage :

Ce qu'il fallait faire ...

- Déclarer une variable supplémentaire
variables valA, valB, valTemp : **entiers**
- Utiliser cette variable pour stocker provisoirement
une des valeurs
 saisir(valA, valB)
 valTemp \leftarrow valA
 valA \leftarrow valB
 valB \leftarrow valTemp

Traitement à faire si ...

Algorithme SimpleOuDouble

{Cet algorithme saisit une valeur entière et affiche son double si cette donnée est inférieure à un seuil donné.)

```
constante (SEUIL : entier) ← 10
variable    val : entier
début
    afficher("Donnez-moi un entier :")      { saisie de la valeur entière}
    saisir(val)
    si val < SEUIL
        alors afficher ("Voici son double :" , val × 2)
        sinon afficher ("Voici la valeur inchangée :" , val)
    fsi
fin
```

L'instruction conditionnelle

```
si <expression logique>
    | alors instructions
    | [sinon instructions]
fsi
```

Si l'expression logique (la condition) prend la valeur **vrai**, le premier bloc d'instructions est exécuté; si elle prend la valeur **faux**, le second bloc est exécuté (s'il est présent, sinon, rien).

Une autre écriture

Algorithme SimpleOuDouble

{Cet algorithme saisit une valeur entière et affiche son double si cette donnée est inférieure à un seuil donné.)

constante (SEUIL : entier) $\leftarrow 10$

variable val : entier

début

afficher("Donnez-moi un entier : ") { saisie de la valeur entière}

saisir(val)

si val < SEUIL

alors val \leftarrow val $\times 2$

 {comparaison avec le seuil }

fsi

afficher("Voici la valeur finale : ", val)

fin

Quand la condition se complique : les conditionnelles emboîtées

Problème : afficher "Reçu avec mention" si une note est supérieure ou égale à 12, "Passable" si elle est supérieure à 10 et inférieure à 12, et "Insuffisant" dans tous les autres cas.

```
si note ≥ 12
    alors afficher( "Reçu avec mention" )
    sinon si note ≥ 10
        alors afficher( "Passable" )
        sinon afficher( "Insuffisant" )
    fsi
fsi
```

La sélection sur choix multiples

selon <identificateur>

(liste de) valeur(s) : instructions

(liste de) valeur(s) : instructions

...

[autres : instructions]

S'il y a plus de deux choix possibles, l'instruction **selon** permet une facilité d'écriture.

L'instruction selon : exemple

selon abréviation

```
"M" :      afficher( " Monsieur " )  
"Mme" :     afficher( " Madame " )  
"Mlle" :     afficher( " Mademoiselle " )  
autres :    afficher( " Monsieur, Madame " )
```

```
Comparer :   si abréviation = "M"  
              |   alors afficher( "Monsieur" )  
              |   sinon si abréviation = "Mme"  
              |       |   alors afficher("Madame")  
              |       |   sinon si abréviation = "Mlle"  
              |           |   alors afficher( "Mademoiselle" )  
              |           |   sinon afficher( "Monsieur, Madame" )  
              |   fsi  
          fsi
```

Quand il faut répéter un traitement ...

Algorithme FaitLeTotal

{Cet algorithme fait la somme des nbVal données qu'il saisit}

variables nbVal, cpt : **entiers**
valeur, totalValeurs : **réels**

début

{initialisation du traitement}

afficher("Combien de valeurs voulez-vous saisir ?")

saisir(nbVal)

{initialisation du total à 0 avant cumul}

totalValeurs \leftarrow 0

{traitement qui se répète nbVal fois}

pour cpt \leftarrow 1 **à** nbVal **faire**

afficher("Donnez une valeur :")

saisir(valeur)

 totalValeurs \leftarrow totalValeurs + valeur

{cumul}

fpour

{édition des résultats}

afficher("Le total des ", nbVal, "valeurs est " ,

fin

Simulation de la boucle pour

- Données : 3 3 -1 10
- Tableau de simulation :

--	--	--	--	--

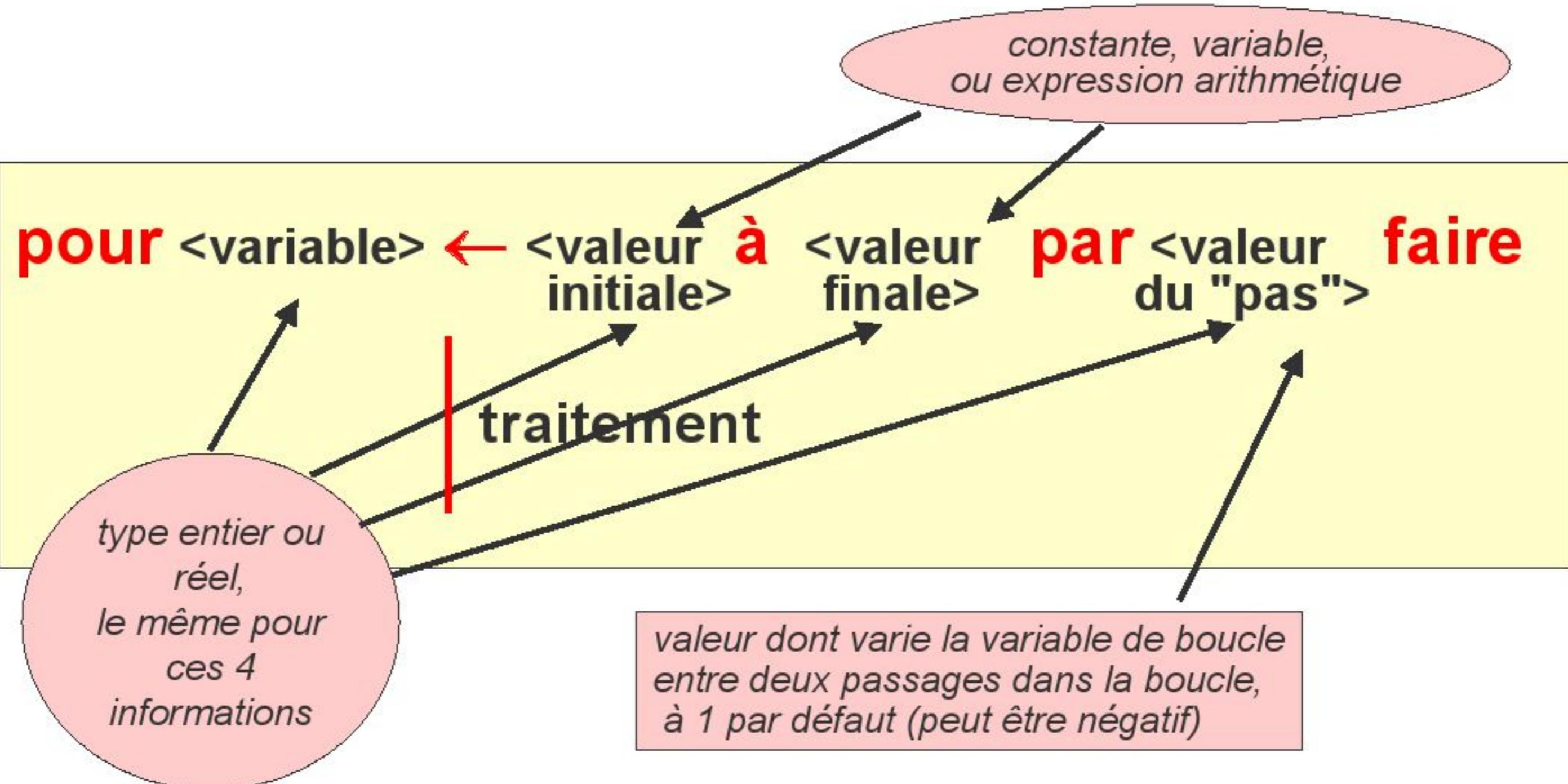
- Affichage :

La boucle « pour »

```
pour <var> ← vallnit à valfin [par <pas>] faire
    | traitement {suite d'instructions}
fpour
```

- Fonction:
répéter une suite d'instructions un certain nombre de fois

Les champs de la boucle pour



Sémantique de la boucle pour

- Implicitement, l'instruction pour:
 - initialise une variable de boucle (le compteur)
 - incrémente cette variable à chaque pas
 - vérifie que cette variable ne dépasse pas la borne supérieure
- Attention :
 - le traitement ne doit pas modifier la variable de boucle

~~pour cpt ← 1 à MAX faire
 si (...) alors cpt ← MAX
fpour~~

Interdit !

Quand le nombre d'itérations n'est pas connu...

Algorithme FaitLeTotal

{Cet algorithme fait la somme des données qu'il saisit, arrêt à la lecture de -1}

constante (STOP : entier) \leftarrow -1

variables val, totalValeurs : entiers

début

 totalValeurs \leftarrow 0

afficher("Donnez une valeur, " , STOP, " pour finir.")

{amorçage}

saisir(val)

tant que val \neq STOP **faire**

 totalValeurs \leftarrow totalValeurs + val

{traitement}

afficher("Donnez une autre valeur, " , STOP, " pour finir.")

saisir(val)

{relance}

ftq

afficher("La somme des valeurs saisies est " , totalValeurs)

fin

Simulation de la boucle tant que

- Données : 3 -3 10 -1
- Tableau de simulation :

				STOP = -1

- Affichage :

La boucle « tant que ... faire »

amorçage

{initialisation de la (des) variable(s) de condition}

tant que <expression logique (vraie)> **faire**

 |
 | traitement

{suite d'instructions}

 | relance

{ré-affectation de la (des) variable(s) de condition}

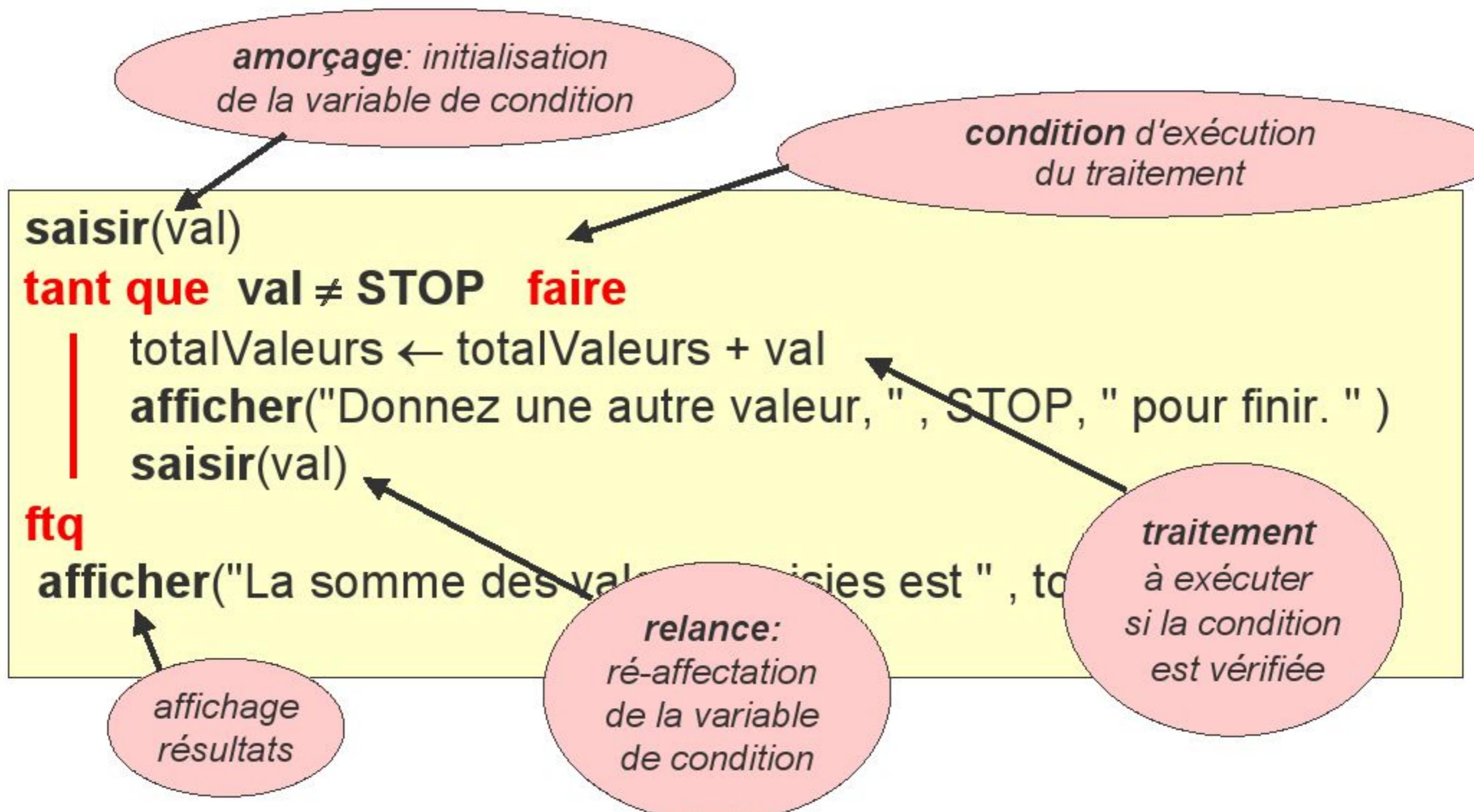
ftq

- Fonction:

- répéter une suite d'instructions tant qu'une condition est remplie

remarque : si la condition est fausse dès le départ, le traitement n'est **jamaïs** exécuté

Sémantique de la boucle tant que



Comparaison boucles pour et tant que

```
pour cpt ← 1 à nbVal faire
    afficher("Donnez une valeur :")
    saisir(valeur)
    totalValeurs ← totalValeurs + valeur {cumul}
fpour
```

...équivaut à :

```
cpt ← 0
tant que cpt < nbVal faire
    afficher("Donnez une valeur :")
    saisir(valeur)
    totalValeurs ← totalValeurs + valeur {cumul}
    cpt ← cpt + 1 {compte le nombre de valeurs traitées}
ftq
```

Comparaison boucles pour et tant que

(suite)

- 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 tant que 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'arrêter quand survient un évènement ,
→ choisir la boucle **tant que**

La boucle répéter : un exemple

Algorithme Essai

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

variables valeur : entier

début

répéter

afficher("Donnez une valeur positive non nulle : ")

saisir(valeur)

tant que valeur \leq 0

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

afficher(valeur)

 ...

{traitement de la valeur saisie}

fin

Simulation de la boucle répéter

- Données : -2 0 4
- Tableau de simulation :

--	--	--

- Affichage :

La boucle « répéter ...tant que »

répéter

(ré)affectation de la (des) variable(s) de
condition
traitement *{suite d'instructions}*

tant que <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 tant que

répéter

```
| afficher("Donnez une valeur positive paire :")
| saisir(valeur)
tant que (valeur < 0 ou (valeur % 2) ≠ 0)
```

...équivaut à :

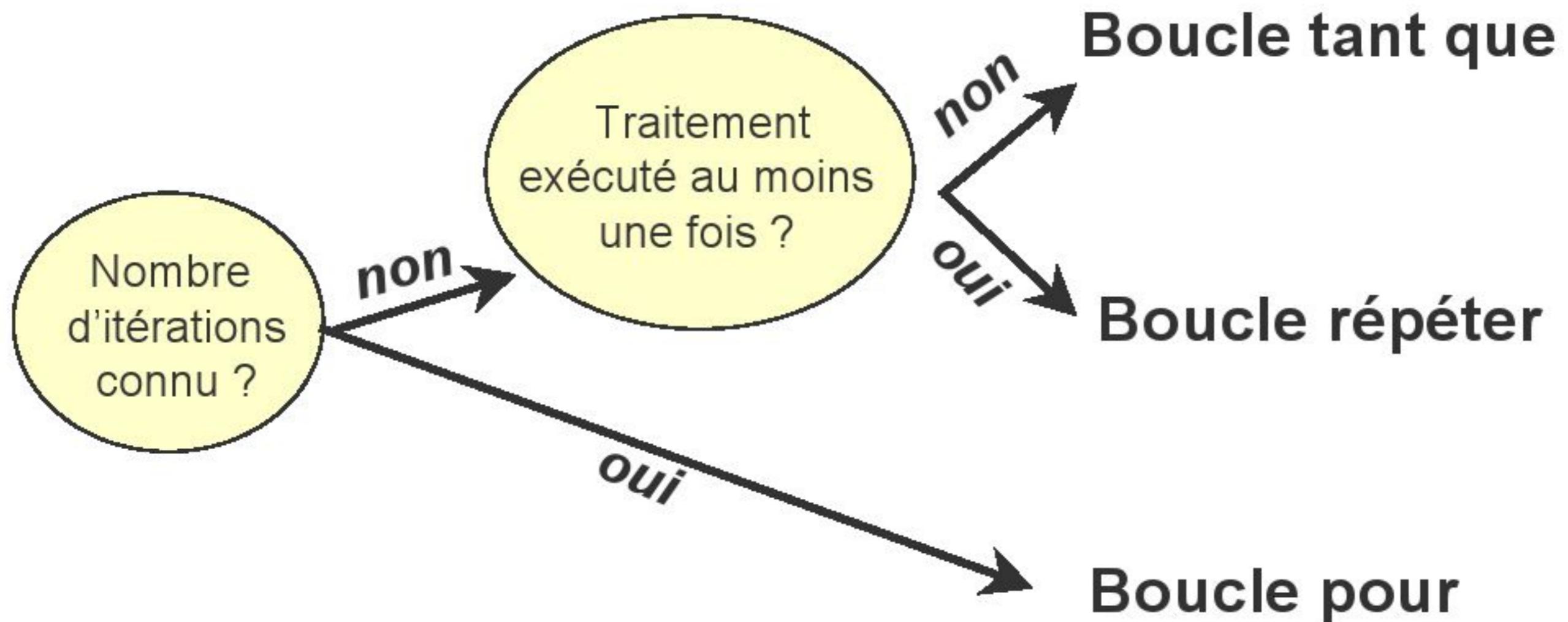
```
afficher("Donnez une valeur positive paire :")
saisir(valeur)
tant que (valeur < 0 ou (valeur % 2) ≠ 0) faire
| afficher("Donnez une valeur positive paire:")
| saisir(valeur)
ftq
```

Comparaison boucles répéter et tant que (suite)

- boucle tant que
 - condition vérifiée **avant** chaque exécution du traitement
 - le traitement peut donc ne pas être exécuté
 - de plus : la condition porte surtout sur la saisie de nouvelles variables (relance)
- boucle répéter tant que
 - condition vérifiée **après** chaque exécution du traitement
 - le traitement est exécuté 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...



Remarque

fsi, ftq et fpour peuvent être omis si le corps se limite à une seule instruction

Exemples:

si val > 0 **alors** afficher(« fini! »)

pour i \leftarrow 1 à MAX **faire** afficher(i \times val)

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'instructions B doit amener A à prendre la valeur Faux.
 - la suite d'instructions B doit modifier au moins une variable de l'expression logique A
 - (mauvais) exemple :
- c'est l'expression logique A (**et elle seule!**) qui en prenant la valeur Faux provoque l'arrêt de la boucle.

```
val1 ← 2 ; val2 ← 3
tant que val1 < 100 faire
    val2 ← val2 × val1
ftq
```

De l'énoncé à la boucle

afficher le carré des valeurs saisies **tant qu'on ne saisit pas 0**

```
saisir(val)
tant que val ≠ 0 faire
    |   afficher(val × val)
    |   saisir(val)
ftq
```

saisir des données et s'arrêter **dès que** leur somme dépasse 500

```
saisir(val)
somme ← val
tant que somme ≤ 500 faire
    |   saisir(val)
    |   somme ← somme + val
ftq
```

De l'énoncé à la boucle (suite)

saisir des données
et s'arrêter **dès que**
leur somme
dépasse 500

```
somme ← 0
répéter
    | saisir(val)
    | somme ← somme + val
tant que somme ≤ 500
```

saisir des données
tant que leur somme
ne dépasse un seuil
donné

Exemple d'un mauvais choix de boucle

Algorithme Somme

{Cet algorithme fait la somme d'une suite de nombres tant que cette somme ne dépasse un seuil donné)

constante (SEUIL : entier) $\leftarrow 1000$

variables val, somme : **entiers**

début

 somme $\leftarrow 0$

répéter

 | **afficher**("Entrez un nombre")

 | **saisir**(val)

 | somme \leftarrow somme + val

tant que somme \leq SEUIL

 | **afficher**("La somme atteinte est" , somme - val)

fin

Version corrigée

Algorithme Somme

{Cet algorithme fait la somme d'une suite de nombres tant que cette somme ne dépasse un seuil donné}

constante (SEUIL : entier) $\leftarrow 1000$
variables val, somme : **entiers**
début

Quand utiliser la boucle tant que?

- Structure itérative "universelle"
n'importe quel contrôle d'itération peut se traduire par le "tant que "
- Structure itérative irremplaçable dès que la condition d'itération devient complexe
Exemple:
saisir des valeurs, les traiter, et s'arrêter à la saisie de la valeur d'arrêt **-1** ou après avoir saisi **5** données.

Exemple

constantes	(STOP : entier) \leftarrow -1 (MAX : entier) \leftarrow 5
variables	nbVal , val : entiers
début	
nbVal \leftarrow 0	{compte les saisies traitées}
saisir (val)	{saisie de la 1 ^{ère} donnée}
tant que val \neq STOP et nbVal < MAX faire	
nbVal \leftarrow nbVal + 1	
...	{traitement de la valeur saisie}
saisir (val)	{relance}
ftq	
afficher (val, nbVal)	{valeurs en sortie de boucle}
...	

Attention :

La valeur d'arrêt n'est jamais traitée (et donc, jamais comptabilisée)

Simulation de la boucle

test 1 : 3 5 -1

test 2 : 3 5 -6 4 0 8

test 3 : 3 5 -6 4 0 -1

test 4 : -1

Interpréter l'arrêt des itérations

```
nbVal ← 0                                {compte les saisies traitées}
saisir(val)                            {saisie de la 1ère donnée}
tant que val ≠ STOP et nbVal < MAX faire
    |
    |   nbVal ← nbVal + 1
    |   ...
    |   saisir(val)          {traitement de la valeur saisie}
    |   {relance}
    |
ftq
si val = STOP
    |
    |   alors {la dernière valeur testée était la valeur d'arrêt}
    |       afficher(« 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}
    |       afficher(« Sortie de boucle car nombre maximum de valeurs
    |                   à traiter atteint; des données significatives n'ont pas
    |                   pu être traitées. »)
    |
fsi
```

De l'importance du test de sortie de boucle (... et donc de la logique)

tant que val \neq STOP et nbVal < MAX faire

- dans la boucle : **val \neq STOP et nbVal < MAX** est vrai
- à la sortie de boucle :
 - soit **val \neq STOP** est faux \rightarrow **val = STOP**
 - soit **nbVal < MAX** est faux \rightarrow **nbVal \geq MAX**
- que tester à la sortie de boucle?
 - **si val = STOP alors ...** voir transparent précédent.
 - **si nbVal \geq 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

Logique propositionnelle

En quoi la logique est-elle utile au programmeur ?

- La logique : une façon de formaliser notre raisonnement
- Il n'y a pas une logique mais DES logiques
- **La logique propositionnelle** : modèle mathématique qui nous permet de raisonner sur la nature vraie ou fausse des expressions logiques

Retour sur les conditions d'itération

tant que somme \leq SEUIL faire...

tant que val \neq STOP et nbVal $<$ MAX faire ...

tant que valeur $<$ 0 ou (valeur % 2) \neq 0 faire...

Proposition :

expression qui peut prendre la valeur VRAI ou FAUX

Exemples de propositions:

2 et 2 font 4

1 et 1 font 10

il pleut

x > y

Eléments de logique propositionnelle

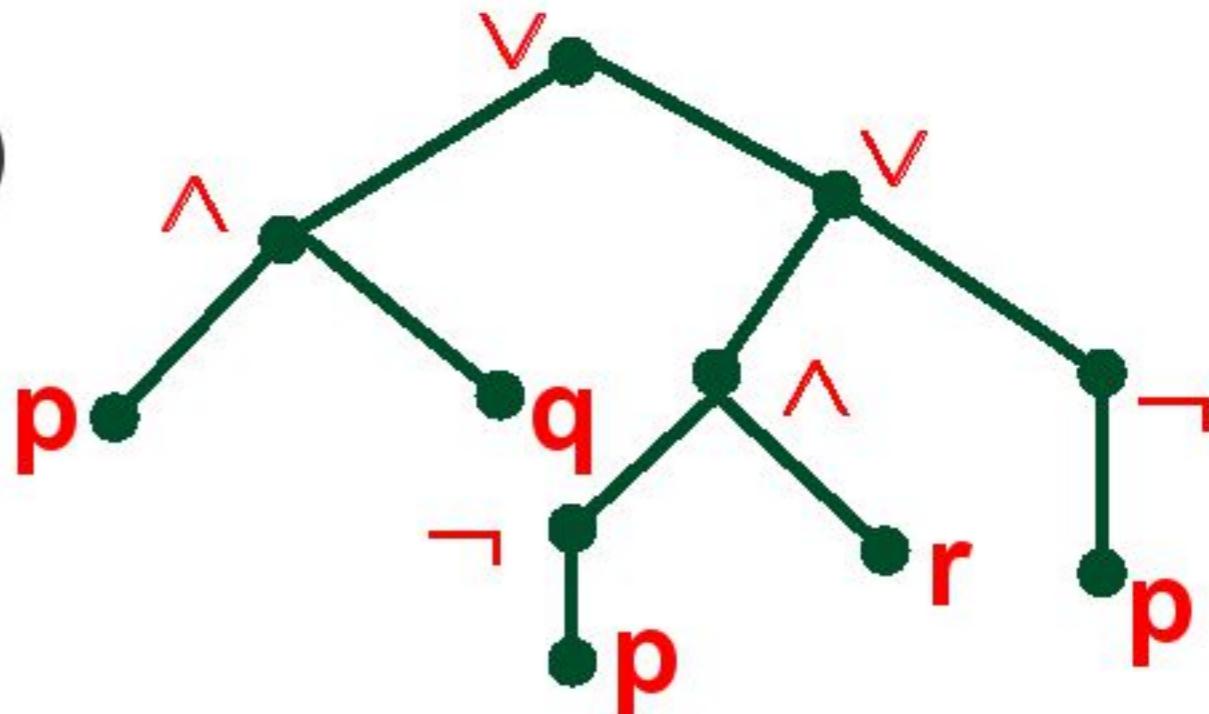
- Formule :
 - expression logique composée de variables propositionnelles et de connecteurs logiques
- Variable propositionnelle :
 - une proposition considérée comme indécomposable
- Connecteurs logiques:

- négation	non, \neg	- conjonction	et, \wedge
- implication	\Rightarrow	- disjonction	ou, \vee
- Exemple : p et q variables propositionnelles
 $((\neg p \vee q) \wedge \neg q) \vee (p \vee \neg q)$

Représentations d'une formule

$$(p \wedge q) \vee ((\neg p \wedge r) \vee \neg p)$$

Par un arbre syntaxique :



En utilisant la notation préfixée (polonaise) :

V A p q V A ¬ p r ¬ p

En utilisant la notation postfixée :

p q A p ¬ r A p ¬ V V

Tables de vérité

Représentation des valeurs de vérité associées à une expression logique

Négation Conjonction Disjonction Implication

p	$\neg p$
V	F
F	V

p	q	$p \wedge q$
V	V	V
V	F	F
F	V	F
F	F	F

p	q	$p \vee q$
V	V	V
V	F	V
F	V	V
F	F	F

p	q	$p \rightarrow q$
V	V	V
V	F	F
F	V	V
F	F	V

p et q : variables propositionnelles

Equivalences classiques

- Commutativité

- $p \wedge q$ équivalent à $q \wedge p$

- $p \vee q$ équivalent à $q \vee p$

- Associativité

- $p \wedge (q \wedge r)$ équivalent à $(p \wedge q) \wedge r$

- $p \vee (q \vee r)$ équivalent à $(p \vee q) \vee r$

- Distributivité

- $p \wedge (q \vee r)$ équivalent à $(p \wedge q) \vee (p \wedge r)$

- $p \vee (q \wedge r)$ équivalent à $(p \vee q) \wedge (p \vee r)$

Equivalences classiques (suite)

- Lois de Morgan

- $\neg(p \wedge q)$ équivalent à $(\neg p) \vee (\neg q)$
- $\neg(p \vee q)$ équivalent à $(\neg p) \wedge (\neg q)$

p	q	$p \wedge q$	$\neg(p \wedge q)$	$\neg p$	$\neg q$	$\neg p \vee \neg q$

Formules : quelques classes et relations

- Les tautologies :

- vraies pour toute assignation de valeurs de vérité aux variables.
- exemple : $p \vee \neg p$

p	$\neg p$	$p \vee \neg p$

- Les formules contradictoires :

- fausses pour toute assignation de valeurs de vérité aux variables.
- exemple : $p \wedge \neg p$

p	$\neg p$	$p \wedge \neg p$

Formules : quelques classes et relations (suite)

- Les formules équivalentes:

- même valeur de vérité pour toute assignation de la même valeur de vérité aux variables.

- exemples : $p \Rightarrow q$ est équivalent à $\neg p \vee q$
 $p \Rightarrow q$ est équivalent à $\neg q \Rightarrow \neg p$

p	q	$p \Rightarrow q$	$\neg p$	q	$\neg p \vee q$
True	True	True	False	True	True
True	False	False	False	False	False

Du bon usage de la logique

Vérification de l'équivalence de deux formules

"être mineur (p) ou majeur ($\neg p$) non imposable (q) "

équivaut à "être mineur (p) ou non imposable (q) "

p	q	$\neg p \wedge q$	$p \vee (\neg p \wedge q)$	$p \vee q$

Applications à l'algorithmique

- Interpréter (et bien comprendre!) l'arrêt des itérations à la sortie d'une boucle.

tant que <cond> faire

À la sortie :	non(<cond>) est vrai
donc si	cond = p et q
à la sortie :	non (p et q)
c'est à dire	non p ou non q

Exemple : avec **<cond>** égal à : **val ≠ STOP et nbVal < MAX**
non(<cond>) égal à : **val = STOP ou nbVal ≥ MAX**

Applications à l'algorithmique (suite)

- **Simplifier une écriture par substitution d'une formule équivalente**

si (Age = "Mineur"
 ou (non (Age = "Mineur") et non (Fisc = "Imposable"))) alors...
Equivalent à :
 si (Age = "Mineur" ou non (Fisc = "Imposable")) alors...
- **Vérifier la validité d'une condition**

si Valeur < 10 et Valeur > 100 alors... ← cas improbable
- **Ecrire la négation d'une condition**

si on veut P et Q et R :
 répéter tant que **non P ou non Q ou non R ou ...**

Le Type BOOLEEN

- Deux constantes booléennes :
VRAI , FAUX
- Des variables de type booléens :
variables **ok, continuer** : booléen
 ok \leftarrow (rep = ‘ O ’ ou rep = ‘ o ’)
 continuer \leftarrow (val > 0 et val < 9)
- Dans les conditionnelles et itératives :
tant que ok faire ...
si continuer alors ...

Le Type BOOLEEN : exemple

Algorithme Logique

constantes (MAX : entier) \leftarrow 5
 (STOP : entier) \leftarrow -1
variables nbVal, val : entiers
 ok : booléen

début

nbVal \leftarrow 0

saisir (val)

ok \leftarrow val \neq STOP et nbVal < MAX

{initialisation de la variable
de boucle booléenne }

tant que ok faire

 | nbVal \leftarrow nbVal + 1

 | saisir(val)

 | **ok \leftarrow val \neq STOP et nbVal < MAX**

{relance}

ftq

si val = STOP alors ...

Booléens : encore des exemples

```
variables val : entier
          encore : booléen
début
    encore ← faux
    val ← 0
    répéter
        afficher( "bonjour " )
        val ← val – 1
        encore ← val > 0
    tant que encore
        afficher( "fini " )
fin
```

```
encore ← faux
val ← 0
tant que non encore faire
    val ← val + 1
    afficher(val )
    encore ← val > 2
ftq
afficher( "fini " )
```

fin Volume 1