



# Algorithmique

ENCG 2007

# Algorithmique

---

- L'algorithmique est un terme d'origine arabe, comme algèbre. Le mot "algorithme" viendrait du nom AL-KHOWÂRIZMI .
- Un algorithme, est un *Processus décrivant étape par étape comment résoudre un problème*
  - indiqué un chemin à quelqu'un ?
  - faire chercher un objet à quelqu'un par téléphone ?
- Si l'algorithme est juste, le résultat est le résultat voulu,
- Si l'algorithme est faux, le résultat est aléatoire

# Algorithmique

---

- Pour fonctionner, un algorithme doit contenir uniquement des instructions compréhensibles par celui qui devra l'exécuter .
- Il peut être très clair pour certains et incompréhensible pour d'autres.
- L'ordinateur est rapide, mais n'est pas intelligent, Il faut lui dire quoi faire, et comment le faire. il faut lui fournir un **algorithme** : *décrire étape par étape comment résoudre le problème*

# Algorithmique

---

Un algorithme doit respecter les deux contraintes suivantes:

- Il doit exploiter un ensemble restreint d'opérations de base (un ordinateur ne comprend que des instructions très simples)
- Il doit produire les résultats voulus en exécutant un nombre fini de ces opérations (sinon l'ordinateur ne résoudra jamais le problème)

# Maîtrise de l'algorithmique

---

La maîtrise de l'algorithmique requiert deux qualités, très complémentaires :

- il faut avoir une **certaine intuition**, car aucune recette ne permet de savoir a priori quelles instructions permettront d'obtenir le résultat voulu.
- il faut être **méthodique et rigoureux**. En effet, chaque fois qu'on écrit une série d'instructions qu'on croit justes, il faut systématiquement se mettre à la place de la machine, qui va les exécuter, afin de vérifier si le résultat obtenu est bien celui que l'on voulait. Cette opération reste indispensable.

# Résumé

---

- Un algorithme, c'est une suite d'instructions, qui une fois exécutée correctement, conduit à un résultat donné.
- un algorithme doit donc contenir uniquement des instructions compréhensibles par celui qui devra l'exécuter.
- la vérification méthodique, pas à pas, de chacun des algorithmes représente plus de la moitié du travail à accomplir.

# Organigrammes

---

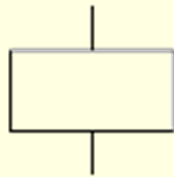
Historiquement, plusieurs types de notations ont représenté des algorithmes:

- Il y a une représentation graphique, avec des carrés, des losanges, etc. qu'on appelle des **organigrammes**.
- On utilise généralement une série de conventions appelée « **pseudo-code** », qui ressemble à un langage de programmation authentique.

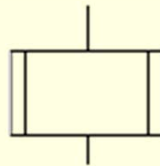
Donc, chaque cuisinier peut faire sa sauce à sa guise, avec ses petites épices bien à lui, sans que cela prêle à conséquence.

# Organigramme

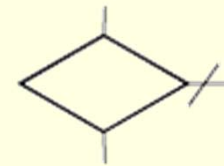
Représentation graphique de l'algorithme. Pour le construire, on utilise des symboles normalisés.



**Symbole général**  
Opération ou groupe d'opérations sur des données, des instructions,...



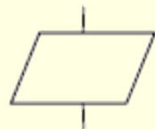
**Sous-programme**  
Portion de programme considérée comme une simple opération.



**Branchement**  
Exploitation de conditions variables impliquant un choix parmi plusieurs.



**Renvoi**



**Entrée-Sortie**



**Début, fin ,  
interruption**

Le sens général des lignes de liaison doit être :

- De haut en bas
- De gauche à droite



# L'algorithme

Indépendamment du type de procédure représenté, un algorithme est composé au minimum des trois éléments suivants:


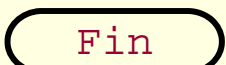
- Les données entrant dans l'algorithme (**entrées**)
- Les opérations transformant les données en résultats
- Les résultats sortant de l'algorithme (**sorties**)



Le parallélogramme est le symbole E/S

# Organigramme

Un algorithme doit avoir

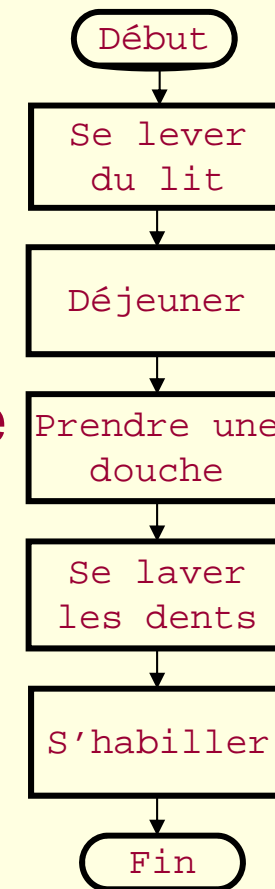
- Un et un seul point d'entrée: 
- Un et un seul point de sortie: 

Ils sont appelés **symboles terminaux**

Les **symboles** sont reliés par des **lignes de flux**

Chaque symbole d'opération dispose de

- Un et un seul flux d'entrée
- Un et un seul flux de sortie



# Pseudo-code

---

- ❑ **L'organigramme** représente graphiquement l'algorithme. C'est visuel (*un dessin vaut mille mots!*), mais ça exige plus de travail de mise en page et plus d'espace sur papier
- ❑ Le **pseudo-code** représente textuellement l'algorithme. Moins visuel, mais plus facile à mettre sur papier et requiert moins d'espace
- ❑ L'algorithme est décrit sous forme de mots et phrases; Semblable aux langages de programmation (Visual Basic, Pascal, C, ...)
- ❑ Le pseudo-code est généralement préféré à l'organigramme. Il est plus rapide à écrire et plus facile à traduire en un langage de programmation

# Avantage du pseudo-code

Le pseudo-code est facilement traduisible en un langage de programmation

## Pseudo-code

```
LIRE Nom, Numero, TotHeures
SI TotHeures > 40 ALORS
    HeuresSup = TotHeures - 40
    ÉCRIRE Nom, Numero, HeuresSup
FINSI
```

## Basic

```
DIM Nom as STRING
DIM Numero, TotHeures as INTEGER;
INPUT(Nom, Numero, TotHeures)
IF TotHeures > 40 THEN
    HeuresSup = TotHeures - 40
    PRINT(Nom, Numero, HeuresSup)
END IF
END
```

## Pascal

```
PROGRAM Exemple;
VAR
    Nom: STRING;
    Numero, TotHeures: INTEGER;
BEGIN
    READLN(Nom, Numero, TotHeures);
    IF TotHeures > 40 THEN BEGIN
        HeuresSup := TotHeures - 40;
        WRITELN(Nom, Numero, HeuresSup);
    END
END.
```

# Algorithmique et programmation

---

Pourquoi apprendre l'algorithmique pour apprendre à programmer ?

- Parce que l'algorithmique exprime les instructions résolvant un problème donné indépendamment des particularités de tel ou tel langage.
- Apprendre l'algorithmique, c'est apprendre à manier la structure logique d'un programme informatique.
- L'algorithmique utilise un ensemble de mots clés et de structures permettant de décrire de manière complète, claire, l'ensemble des opérations à exécuter sur des données pour obtenir des résultats.

# Algorithmique et programmation

---

Fondamentalement, les ordinateurs, quels qu'ils soient, ne comprennent que quatre catégories d'ordres (d'**instructions**).

Ces quatre familles d'instructions sont :

- l'affectation de variables
- la lecture / écriture
- les tests
- les boucles

Un algorithme informatique se ramène donc toujours à la combinaison de ces quatre petites briques de base.

Il peut y en avoir quelques unes, quelques dizaines, et jusqu'à plusieurs centaines de milliers.

# Constante et variable

---

Un ordinateur résout des problèmes ou contrôle du matériel en manipulant un "modèle" du problème.

Le problème est représenté dans la machine comme un ensemble de valeurs (nombres ou textes) qui le décrivent. Ces valeurs sont des données.

Certaines données sont fixes tout au long du problème. Ce sont des caractéristiques statiques du problème ou encore des constantes.

Lorsqu'une donnée n'est pas constante, elle est "variable".

# Exemple

Si on veut calculer la surface d'un cercle, on demande à l'ordinateur de calculer la formule:

$$A = PI \times r^2 \quad \text{PI est une constante.}$$

On donne à l'ordinateur une valeur (r) d'un rayon et on reçoit le résultat (A).

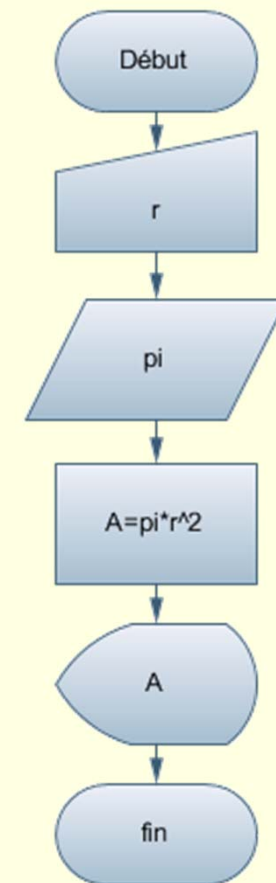
r ?

$r \leftarrow 2$

$A \leftarrow PI * r^2 = 12,56$

$A \leftarrow 12,56$

L'ordinateur a rangé la valeur (donnée) 2 dans une **variable** r, et utilisé une autre **variable** A pour le résultat.





# Affectation d'une variable

Affecter une variable c'est lui attribuer une valeur

En pseudo-code, l'instruction d'affectation se note avec le signe  $\leftarrow$

$r \leftarrow 2$  //Attribue la valeur 2 à la variable r

L'ordre dans lequel les instructions sont écrites va jouer un rôle essentiel dans le résultat final.

**Variable A en Numérique**

**Début**

A  $\leftarrow$  34

A  $\leftarrow$  12

**Fin**

**Variable A en Numérique**

**Début**

A  $\leftarrow$  12

A  $\leftarrow$  34

**Fin**

# Variable

---

- Une variable associe un nom à une valeur appartenant à un ensemble donné.
- Une variable correspond toujours à une valeur
- La valeur d'une variable peut être modifiée par l'opération d'affectation
- La valeur de la variable appartient à un domaine particulier défini par le type de la variable.
- Les variables permettent de réaliser le calcul car elles permettent de stocker les données, les calculs intermédiaires et les résultats.

# Types de base

---

les quatre types de base de toute algorithmique informatique sont

- Les booléens (BOOLEEN)
- Les entiers (ENTIER)
- Les flottants (REEL)
- Les caractères (CARACTERE)

Un type de base est un type qui ne peut se décomposer en assemblage d'autres types.

# Types

---

- 1. L'entier
  - 45, 36, -564, 0 ... en décimal
  - 45h, 0FBh, 64h ... en hexadécimal
  - % 10101111, %1011 ... en binaire
- 2. Le réel
  - -6, -3.67, 4.2569, -564.0, 18.36 10<sup>-6</sup>
- 3. Le booléen
  - VRAI ou FAUX
- 4. Le caractère
  - 'a', 'A', '\*', '7', 'z', '!' ....
- 5. La chaîne de caractères
  - 'électronique', 'cd ROM de 80mn'

# Syntaxe et instructions

---

La syntaxe est l'ensemble des règles d'écriture du langage de programmation. Cela comprend :

- **l'écriture des instructions** : opérateurs, variables, constantes et valeurs littérales, symboles
- **les mots clefs** qui ont une signification prédéfinie
- **l'écriture des structures de contrôle** : boucles, tests, déroutements
- **l'écriture des unités de programmation** : blocs, fonctions, routines, paquetages, librairies, etc...

# Les instructions de lecture et d'écriture

---

Lecture et écriture sont des termes qui doivent être compris du point de vue de la machine qui sera chargée de les exécuter.

Dès que le programme rencontre une instruction **Lire**, l'exécution s'interrompt, attendant la frappe d'une valeur au clavier

Pour entrer la valeur de T : **Lire** T

Pour écrire quelque chose à l'écran : **Écrire** A

# Exemple

---

Quel résultat produit le programme suivant ?

**Variables** val, double **numériques**

**Début**

Val ← 231

Double ← Val \* 2

**Ecrire** Val

**Ecrire** Double

**Fin**

# Structures de contrôle

---

- Un programme n'est jamais une suite linéaire d'ordres. On doit pouvoir faire des *choix* : **le programme devient conditionnel.**
- Si un traitement identique (ou presque identique) se répète plusieurs fois, on doit pouvoir faire en sorte de ne l'écrire qu'une fois : **le programme devient itératif**
- Les *choix* et les *boucles itératives* sont les **deux types majeurs** de structures de contrôle d'un algorithme.



# Qu'est ce qu'une condition

Une condition est une comparaison entre valeurs de même type

Elle signifie qu'une condition est composée de trois éléments :

- une **valeur**
- un **opérateur de comparaison**
- une **autre valeur**

Les **opérateurs de comparaison** sont :

égal à...	=	strictement plus grand que...
différent de...	<>	>
strictement plus petit que...	<	plus petit ou égal à... <=
	>	plus grand ou égal à... >=

# Conditions composées

Certains problèmes exigent parfois de formuler des conditions qui ne peuvent pas être exprimées sous la forme simple.

L'informatique met à notre disposition quatre opérateurs logiques : ET, OU, NON, et XOR.

## Exemple

« X est compris entre 5 et 8 ».

En fait cette phrase cache non une, mais **deux** conditions. Car elle revient à dire que

( X est supérieur à 5 ) **ET** ( X est inférieur à 8 )

Il y a donc bien là deux conditions, reliées par un **opérateur logique** « ET ».

# Conditions complexes

---

## Conditions simples

$a < b$ ,  $a \leq b$ ,  $a > b$ ,  $a \geq b$ ,  $a = b$ ,  $a \neq b$

## Conditions complexes

On peut combiner des conditions simples avec **et** et **ou**

$(a \geq 5)$  et  $(a \leq 100)$

$(a = 10)$  ou  $(a > 20)$

La négation (**non**) peut inverser une condition

$\text{non}((a \geq 5) \text{ et } (a \leq 100))$

# Conditions

---

Attention: il arrive de formuler dans un test **une condition qui ne pourra jamais être vraie, ou jamais être fausse**

( X est inférieur à 10) **ET** (X est supérieur à 15 )

il est très difficile de trouver un nombre qui soit à la fois inférieur à 10 et supérieur à 15 !

cela veut dire qu'on a écrit un test qui n'en est pas un.

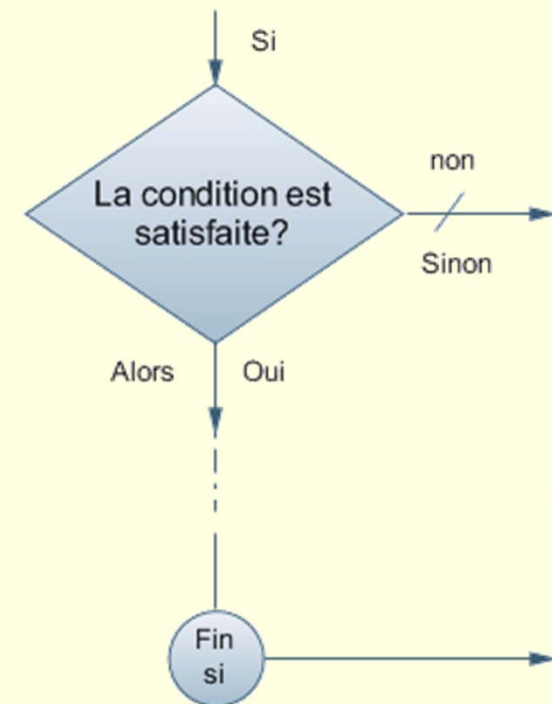
# Conditions

Pour représenter une séquence conditionnelle d'opérations dans l'organigramme

On utilise le **losange** : c'est le *symbole conditionnel*

Pour représenter la convergence des flux d'exécution

On utilise le **cercle** : c'est le *symbole de convergence*



# Structures conditionnelles

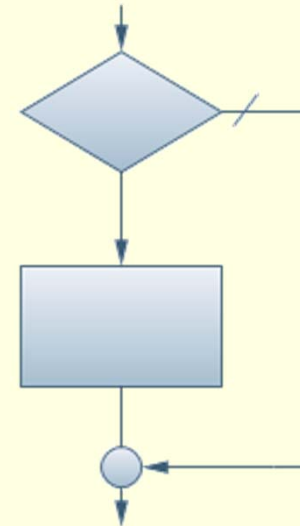
Exprimer un choix, c'est décider d'exécuter **ça**... ou **ça**. La machine doit pouvoir décider lequel des "ça" elle exécute. Elle va donc analyser une condition, puis prendre une décision.

## Conditionnalité

Le cas le plus simple du choix est la « *condition* »

**SI** *condition* **ALORS**  
... Instructions  
**FIN SI**

**Si** *booléen* **Alors**  
*Instructions*  
**Fin si**



# Structures conditionnelles

---

**Si** booléen **Alors**  
Instructions  
**Finsi**

Un **booléen** est une **expression** dont la valeur est VRAI ou FAUX.

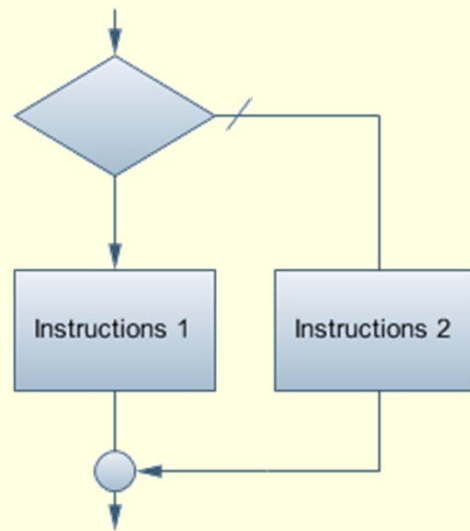
Donc un booléen ne peut être que :

- une **variable** (ou une expression) de type booléen
- une **condition**

# Structures conditionnelles

## Dualité (alternative)

La dualité suppose le choix entre une *alternative* de traitements (on rappelle qu'alternative s'emploie au singulier et désigne les deux cas en balance).



**SI** *condition* **ALORS**  
... instructions 1  
**SINON**  
... Instructions 2  
**FIN SI**



# Exemple

Allez tout droit jusqu'au prochain carrefour

**Si** la rue à droite est autorisée à la circulation **Alors**

Tournez à droite

Avancez

Prenez la deuxième à gauche

} Instructions 1

**Sinon**

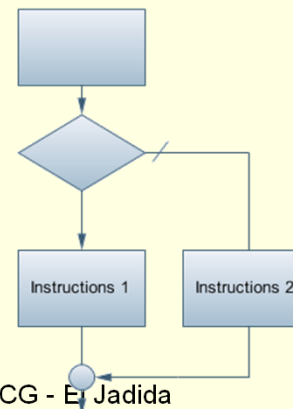
Continuez jusqu'à la prochaine rue à droite

Prenez cette rue

Prenez la première à droite

} Instructions 2

**Finsi**

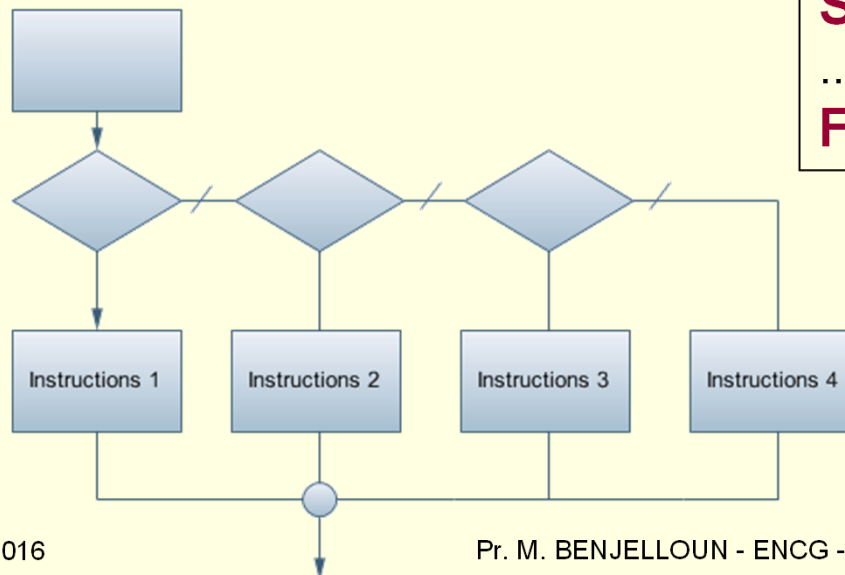


# Structures conditionnelles

## Branchement

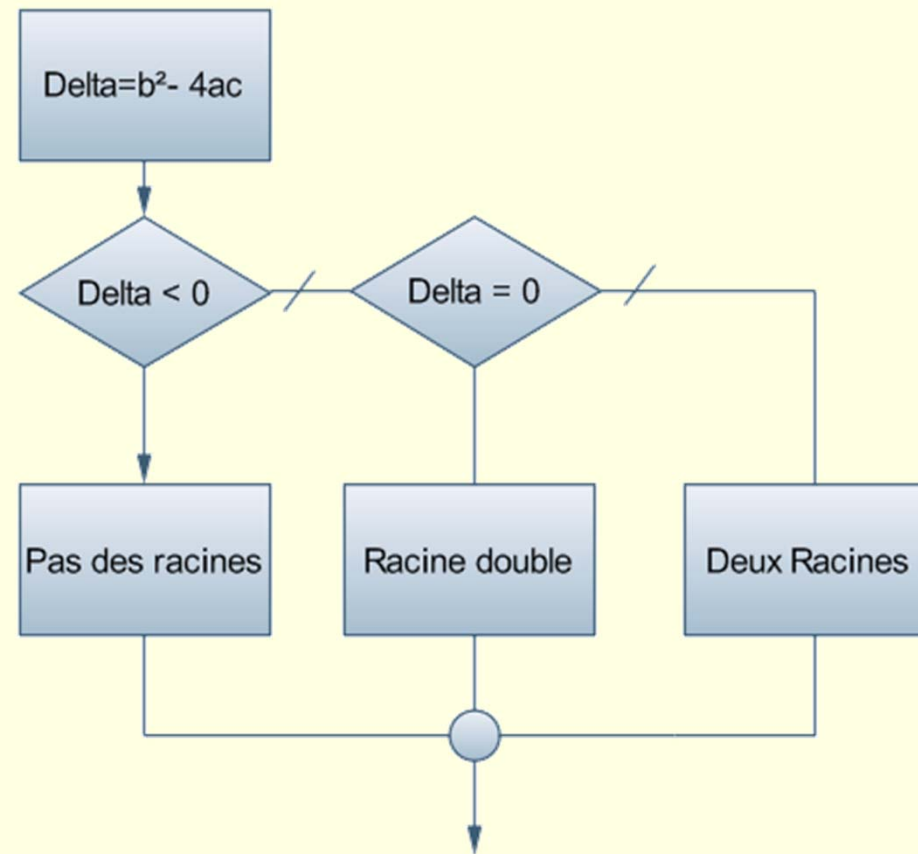
Le branchement est une généralisation de la structure de contrôle conditionnelle, lorsque le nombre de traitements différents est plus grand que deux.

```
SI condition1 ALORS  
... instructions ...  
SINON SI condition2 ALORS  
... instructions ...  
SINON SI condition3 ALORS  
... instructions ...  
SINON  
... Instructions  
FIN SI
```



# Exemple

Équation 2<sup>ème</sup> degré :  $ax^2+bx+c=0$



# Exemple : l'état de l'eau

Écrire un programme qui donne l'état de l'eau selon sa température : solide, liquide ou gazeuse.

**Variable T en Entier**

**Début**

**Ecrire** "Entrez la température de l'eau :"

**Lire** T

**Si**  $T \leq 0$  **Alors**

**Ecrire** "C'est de la glace"

**FinSi**

**Si**  $T > 0$  **Et**  $T < 100$  **Alors**

**Ecrire** "C'est du liquide"

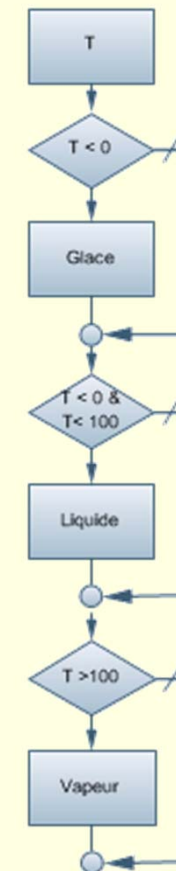
**Finsi**

**Si**  $T > 100$  **Alors**

**Ecrire** "C'est de la vapeur"

**Finsi**

**Fin**



# Exemple : l'état de l'eau

**Variable T en Entier**

**Début**

**Ecrire** "Entrez la température de l'eau :"

**Lire** T

**Si**  $T \leq 0$  **Alors**

**Ecrire** "C'est de la glace"

**Sinon**

**Si**  $T < 100$  **Alors**

**Ecrire** "C'est du liquide"

**Sinon**

**Ecrire** "C'est de la vapeur"

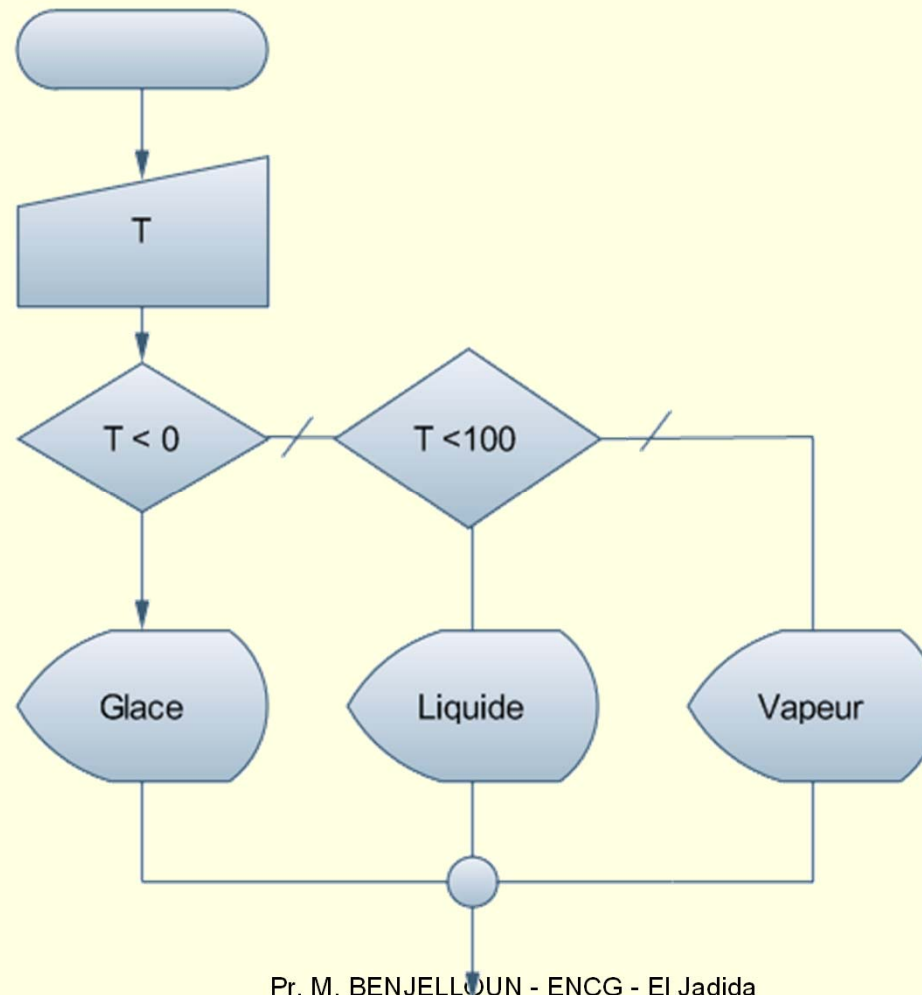
**Finsi**

**Finsi**

**Fin**

Pseudo-code plus simple

# Exemple : l'état de l'eau



# Structures de boucles

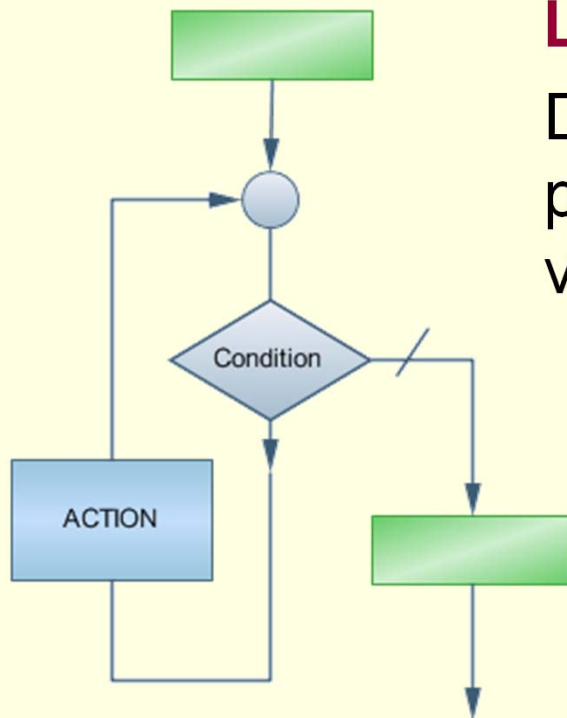
---

Les structures de boucles sont invoquées quand il s'agit de répéter un traitement plusieurs fois dans des conditions quasiment similaires.

La programmation obtenue par l'écriture de boucles est dite "*itérative*".

Les structures de boucle diffèrent par les nombres minimum et maximum d'itérations. On les classe en trois types :

# Structures de boucles



## La boucle conditionnelle

Dans cette structure, on commence par tester la condition ; si elle est vérifiée, le traitement est exécuté

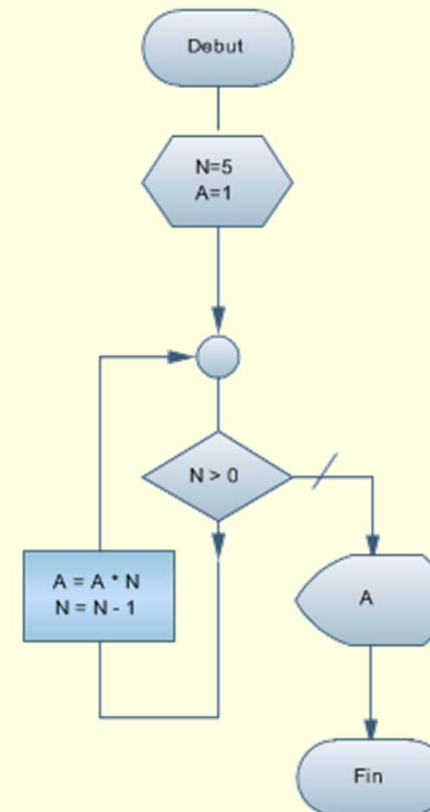
**TANT QUE** *condition* **FAIRE**  
... instructions

**L'action peut ne jamais être exécutée .**

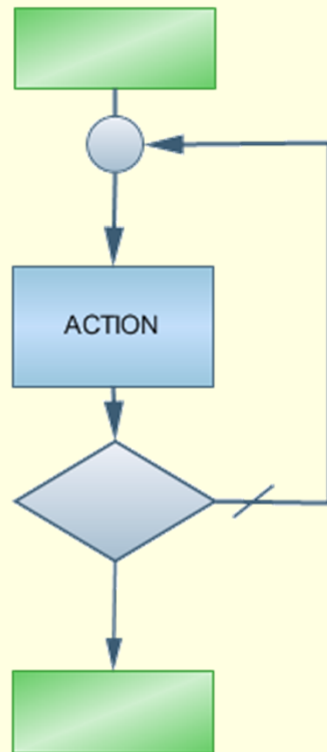


# Exemple

DEBUT  
 $N \leftarrow 5$   
 $A \leftarrow 1$   
**TANTQUE  $N > 0$  FAIRE**  
 $A \leftarrow A \times N$   
 $N \leftarrow N - 1$   
**FIN TANTQUE**  
AFFICHER A  
FIN



# Structures de boucles



## La boucle répétition indéterminée

Dans cette structure, le traitement est exécuté une première fois puis sa répétition se poursuit jusqu'à ce que la condition soit vérifiée.

**REPETER**

... instructions ...

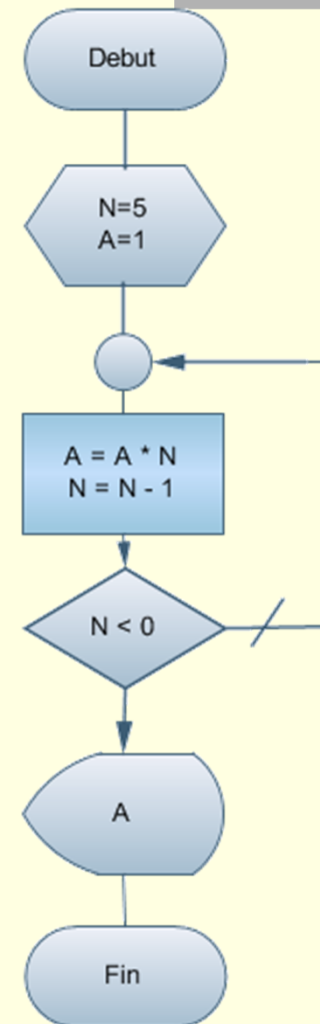
**JUSQU'À CE QUE** *condition*

... suite ...

**L'action est toujours exécutée au moins une fois.**

# Exemple

DEBUT  
 $N \leftarrow 5$   
 $A \leftarrow 1$   
REPETER  
 $A \leftarrow A \times N$   
 $N \leftarrow N - 1$   
JUSQU'À CE QUE  $N < 0$   
AFFICHER A  
FIN



# Structure RÉPÉTER-JUSQU'À

La condition de la structure RÉPÉTER-JUSQU'À est inversée par rapport à la structure TANTQUE équivalente

```
Nombre = 0
TANTQUE Nombre > 0 FAIRE
    ÉCRIRE "Nombre positif?"
    LIRE Nombre
FINTANTQUE
```

```
RÉPÉTER
    ÉCRIRE "Nombre positif?"
    LIRE Nombre
JUSQU'À Nombre < 0
```

# Structures de boucles

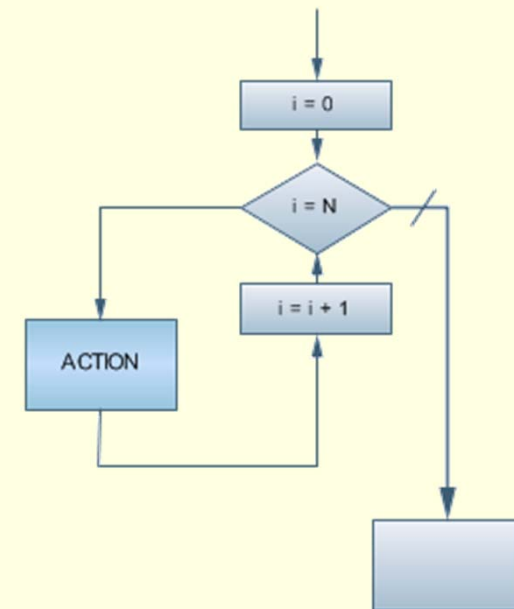
## La boucle déterminée

Dans cette structure, la sortie de la boucle d'itération s'effectue lorsque le nombre souhaité de répétition est atteint.

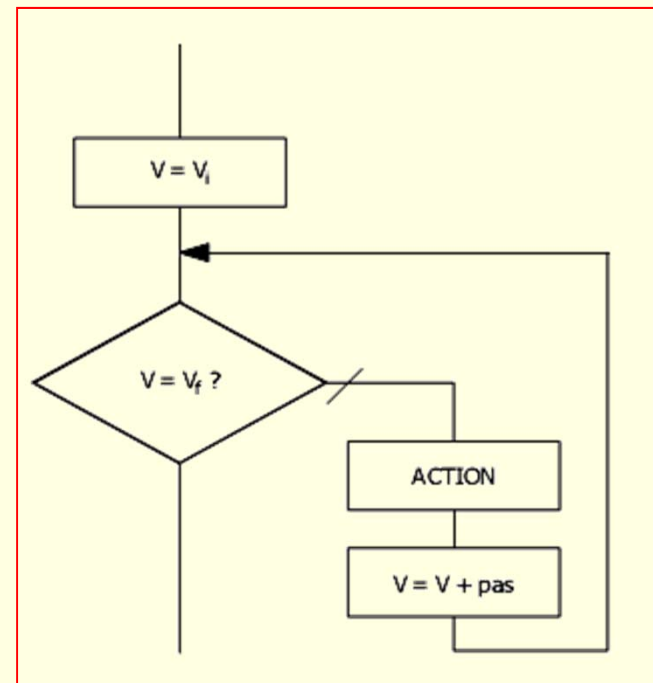
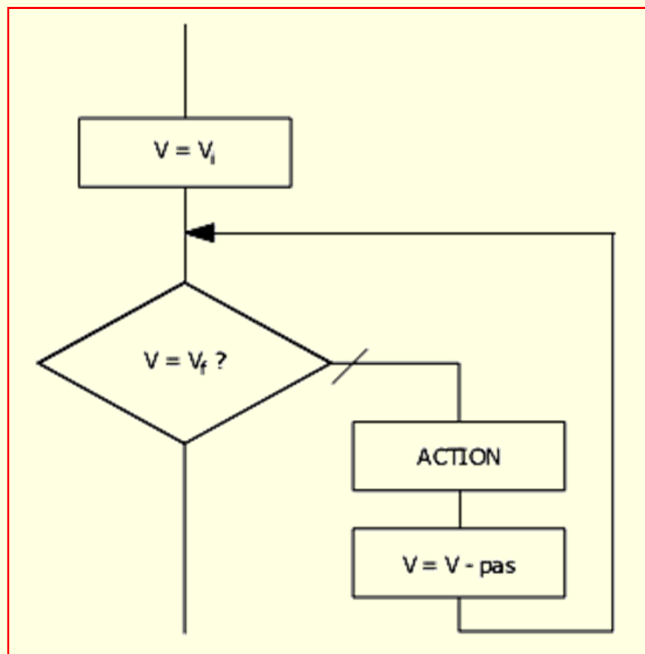
**Pour** Variable **allant de** V1 **à** V2  
**faire**  
... instructions ..  
**Fin Faire**

### Exemple

$A \leftarrow 1$   
Pour  $i$  allant de 1 à 5  
faire  
 $A \leftarrow A \times i$   
Fin Faire



# Structures de boucles



$V$  : variable

$V_i$  : valeur initiale de  $V$

$V_f$  : valeur finale de  $V$

# Composition des structures de contrôle

---

- La programmation résout des problèmes par des algorithmes qui sont des associations de ces structures de contrôle.
- On peut donc considérer les structures précédentes comme des « briques élémentaires » : Condition, Alternative, Choix, TantQue, Faire TantQue.
- Le nombre de ces structures de contrôle n'est pas infini. Par contre, le nombre de problèmes que l'on peut résoudre avec l'est.
- Dans certains cas, associer ces structures de contrôle ne produit que l'association "logique" des deux principes

# Quelle structure utiliser?

---

Trois structures répétitives sont disponibles  
Laquelle utiliser dans un algorithme?

Voici quelques règles à considérer

- Si le nombre d'itérations est déterminé par une variable compteur  
*Utiliser la structure **POUR***
- Si au moins une itération est requise  
*Utiliser la structure **RÉPÉTER-JUSQU'À***
- Dans les autres circonstances  
*Utiliser la structure **TANTQUE***