

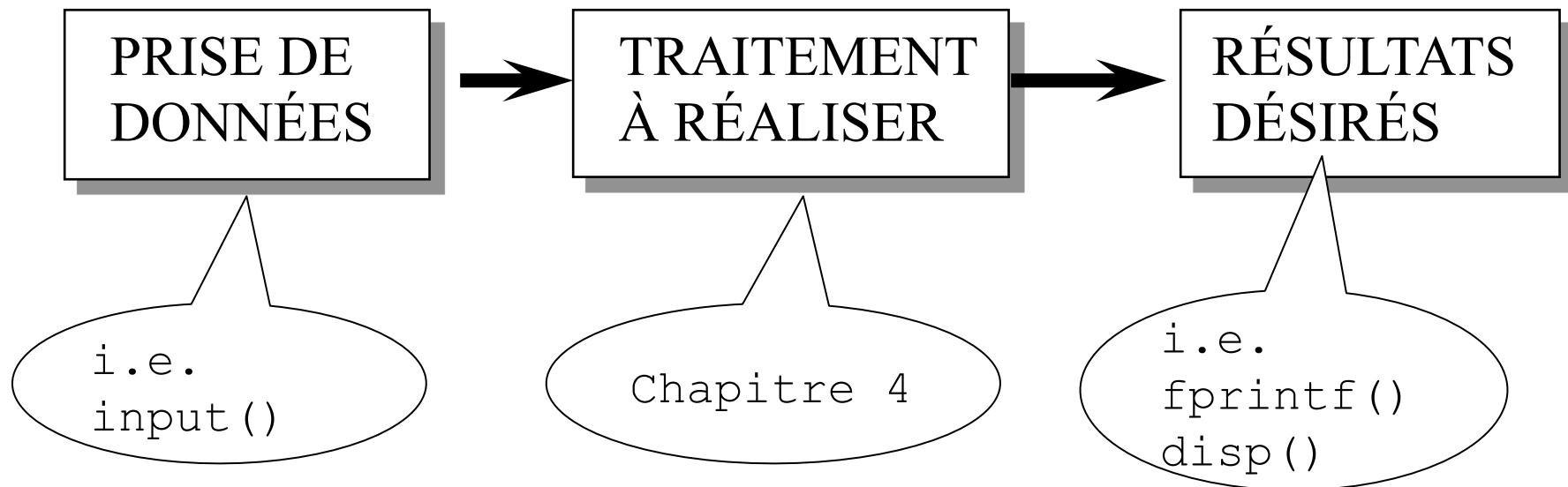
# INF1005A: Programmation procedurale

## Chapitre 4: Structures de programmation

# L'informatique

L'informatique se définit comme la science du traitement automatique de l'information.

Le traitement de l'information se résume par étapes suivantes:



# Agenda

- Expression booléenne, type logique
- Structures de programmation, pseudo-code schématique
- Structures de décision
- Structures de répétition

# Expression booléenne, type logique – **opérateurs relationnels**

- Une expression booléenne est une mise en relation de deux éléments (matrices, variables, nombres, chaînes de caractères) qui donne comme résultat une matrice logique composée des booléens faux (0) ou vrai (1).
- Deux éléments sont mis en relation avec les **opérateurs relationnels** suivants:
  - > (plus grand), < (plus petit),  $\geq$  (plus grand ou égal),  
 $\leq$  (plus petit ou égal), = = (égal),  $\sim$  = (différent)

# Expression booléenne, type logique – opérateurs relationnels

Soient  $A = [1 \ 3 \ 4]$ ,  $B = 2$ ,  $C = 22$  et  $D = 'MATLAB'$ , on obtient pour les relations suivantes:

$A > B$  %réponse [0 1 1]

$B < 12$  %réponse 1

$C < B$  %réponse 0

$D > 'A'$  %réponse [1 0 1 1 0 1]

$D == 'Allo'$  %réponse (??? Error using ==  
Matrix dimensions must agree.)

# Expression booléenne, type logique – opérateurs relationnels

- Pour les matrices dans les expressions booléennes, les comparaisons sont faites élément par élément.

Exemple:

Soient  $A = [1 \ 3 \ 4 \ 5]$ ,  $B = [4 \ 6 \ 3 \ 5]$

$A \sim = B$  %réponse [1 1 1 0]

- Si une comparaison est faite entre deux matrices de dimensions différentes, MATLAB affichera un message d'erreur, sauf si une des matrices est un scalaire.
- La fonction `isequal()` vérifie si deux matrices sont semblables peu importe leur taille.

Exemple:

Soient  $A = [1 \ 2 \ 3]$ ,  $B = [3 \ 2]$ ,  $C = [1 \ 2 \ 3]$ .

`isequal(A, B)` %réponse 0

`isequal(A, C)` %réponse 1

### Expression booléenne, type logique – `all()`, `any()`

Les fonctions `all()` et `any()` peuvent aussi être utiles pour transformer:

- un vecteur logique en un scalaire logique.
- une matrice logique en un vecteur logique.

### Expression booléenne, type logique – `all()`, `any()`

i.e. un vecteur logique en un scalaire logique.

Soient  $A = [1 \ 0 \ 3 \ 5]$ ,  $B = [1 \ 2 \ 3 \ 5]$ ,  $C = [1 \ 2 \ 3 \ 5]$

`all(A)` %réponse 0

`any(A)` %réponse 1

`Res=B==C` %réponse `Res=[1 1 1 1]`

`any(Res)` %réponse 1

`all(Res)` %réponse 1

`Res=A==B` %réponse `Res=[1 0 1 1]`

`any(Res)` %réponse 1

`all(Res)` %réponse 0

### Expression booléenne, type logique – `all()`, `any()`

i.e. une matrice logique en un vecteur logique.

Soit  $A = [1 \ 0 \ 3 \ 5 \ 0;$

$1 \ 2 \ 3 \ 5 \ 0;$

$1 \ 2 \ 3 \ 5 \ 0]$

`all(A)` %réponse [1 0 1 1 0]

`any(A)` %réponse [1 1 1 1 0]

# Expression booléenne, type logique - indexation

- Indexation avec les matrices logiques

Exemple:

Soient  $A = [1 \ 2 \ 3 \ 5]$ ,  $B = [1 \ 3 \ 4 \ 5]$

Res = A == B %réponse [1 0 0 1]

Val = A(Res) (même chose Val = A(A == B))

%réponse [1 5]

On peut aussi faire:

$A(A == B) = A(A == B) + 10$  %réponse [11 2 3 15] (!!)

Donner la réponse de:

$A = A(A == B) + 10$  %réponse [11 15]

### Expression booléenne, type logique – **isempty()**

`isempty(matrice)` : permet de savoir si une matrice est vide. Cette fonction retourne un scalaire logique.

i.e.

```
matriceP = [1 0 4];  
matriceV = [];  
rep = isempty(matriceP)  
rep = 0 % pas vide  
rep = isempty(matriceV)  
rep = 1 % vide
```

# Expression booléenne, type logique – opérateurs logiques

On peut combiner les résultats de plusieurs opérateurs relationnels avec les opérateurs logiques :

- & (et, conjonction, intersection),
- | (ou, disjonction, union),
- ~ (négation)
- XOR (ou exclusif)

Leur résultat dépendent de la table de vérité suivante:

A	B	&		XOR()
FAUX(0)	FAUX(0)	FAUX(0)	FAUX(0)	FAUX(0)
FAUX(0)	VRAI(1)	FAUX(0)	VRAI(1)	VRAI(1)
VRAI(1)	FAUX(0)	FAUX(0)	VRAI(1)	VRAI(1)
VRAI(1)	VRAI(1)	VRAI(1)	VRAI(1)	FAUX(0)

# Expression booléenne, type logique – opérateurs logiques

- Pour l'opérateur de négation  $\sim$ , le résultat est l'inverse logique de l'entrée.
- Pour  $A = 0$ ,  $\sim A$  donne 1.
- Pour les matrices, ces opérateurs sont appliqués élément par élément.
- Tout élément non nul est considéré comme vrai (1), alors qu'un élément nul est considéré faux (0) (**Très important**).

# Expression booléenne, type logique – opérateurs logiques

Exemples:

$$\underline{A = [1 \ 2 \ 3 \ 4], \ B = [1 \ 3 \ 1 \ 1]}$$

$$C = (A > 2) \& (B == 1)$$

%réponse  $C = [0 \ 0 \ 1 \ 1] \ \& \ [1 \ 0 \ 1 \ 1]$  donc  $C = [0 \ 0 \ 1 \ 1]$

$$C = \sim (A > 2) \ | \ (B == 1)$$

%réponse  $C = [1 \ 1 \ 0 \ 0] \ | \ [1 \ 0 \ 1 \ 1]$  donc  $C = [1 \ 1 \ 1 \ 1]$

$$\underline{A = [1 \ 2 \ 3 \ 4], \ B = [1 \ 3]}$$

$$C = (A > 2) \ \& \ (B == 1)$$

%réponse  $C = [0 \ 0 \ 1 \ 1] \ \& \ [1 \ 0]$  donc erreur !

(Error using &

Matrix dimensions must agree.)

# Expression booléenne, type logique – opérateurs logiques

Les opérateurs de conjonction et de disjonction sont disponibles dans une seconde version plus optimisée:

`||` (disjonction, union, ou) et `&&` (conjonction, intersection, et).

Pour ces opérateurs, MATLAB évalue la deuxième partie de la condition seulement si c'est nécessaire et fonctionne que pour les scalaires.

Pour `A=4, B=5, C=[1 2]`

`A==3 && B==5` % `B==5` n'est pas évalué. Pas nécessaire.

`A==4 && B==5` % `A==4` et `B==5` sont évalués

`A==4 || B==1` % `B==1` n'est pas évalué. Pas nécessaire.

`A==3 || B==3` % `A==3` et `B==3` sont évalués

## Expression booléenne, type logique – opérateurs logiques

Pour matrice: soient  $aij$  un élément de la matrice A et  $bij$  un élément de la matrice B, ayant les mêmes indices (les matrices doivent être de même dimension à moins que l'une soit un scalaire)

A&B	1	$aij \neq 0$ et $bij \neq 0$
	0	$aij = 0$ ou $bij = 0$
A B	1	$aij \neq 0$ ou $bij \neq 0$
	0	$aij = 0$ et $bij = 0$
$\sim A$	1	$aij = 0$
	0	$aij \neq 0$

# Agenda



## Expression booléenne, type logique

- Structures de programmation, pseudo-code schématique
- Structures de décision
- Structures de répétition

# Structures de programmation - Qu'est-ce qu'une structure de programmation?

## Structure de répétition

1\*2

2\*2

3\*2

4\*2

Équivalent à répéter  $x^2$  avec  $x = [1 \ 2 \ 3 \ 4]$

En d'autres mots: POUR  $x$  FAIRE  $x^2$

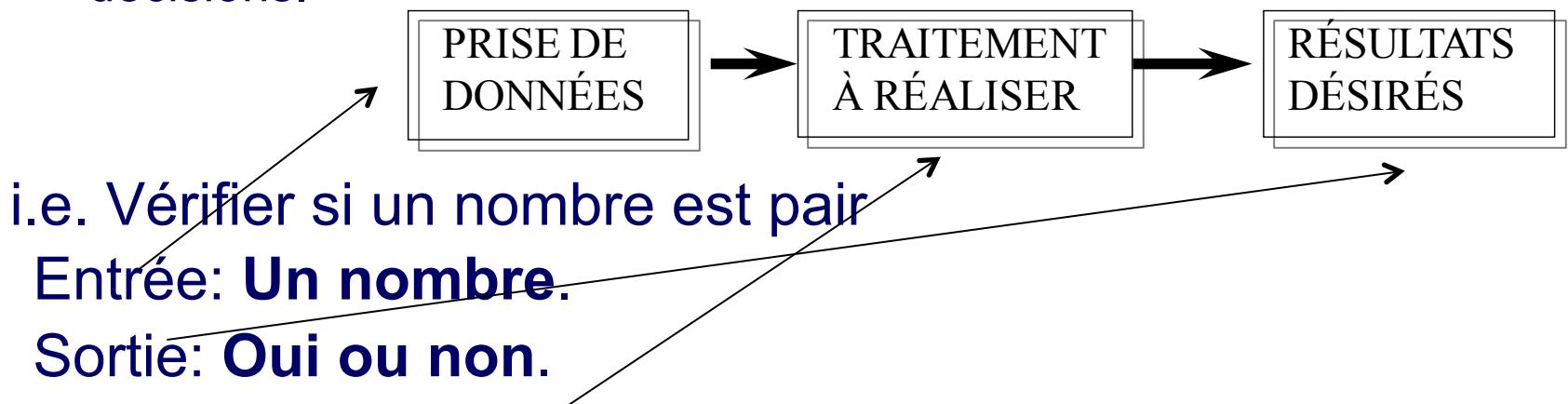
## Structure de décision

SI le reste de la division de C par 2 est zéro ALORS C est paire SINON C est impaire

# Programmation procédurale – introduction

Un programme est:

- des valeurs d'entrée qui sont transformées en des valeurs de sorties par une séquence d'opérations, de répétitions et de décisions.



Séquence d'opérations: Calculer le reste de la division du nombre par 2 et ensuite vérifier si le résultat est 0 (pas de reste, donc paire).

# Programmation procédurale – astuces de programmation

- Identifier les entrées et les sorties:
  - Combien d'entrées? Combien de sorties?
- Identifier les différentes opérations requises et préciser la séquence qui exprime la tâche.
- Préciser les répétitions en regroupant les opérations à répéter:
  - - Est-ce qu'il y a des répétitions ? Combien ? Et elles répètent quelles opérations?
- Préciser les décisions en regroupant les opérations dépendantes d'une condition à vérifier:
  - - Est-ce que des décisions sont requises? Combien? Quelles sont les alternatives? Elles influencent quelles opérations ?

### Pseudo-code schématique - **définition**

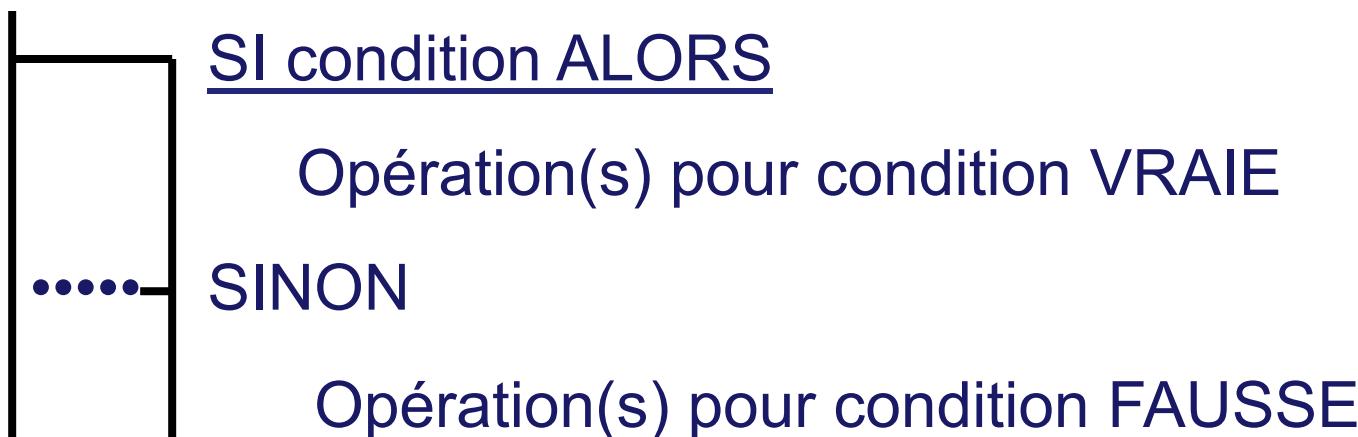
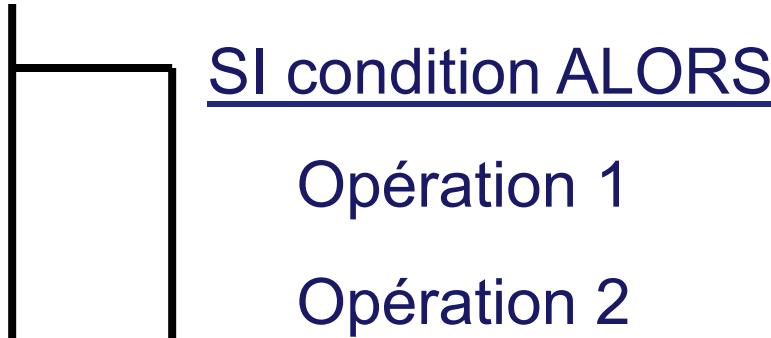
Bien réfléchir à la structure du programme avant de programmer en MATLAB.

Il y a des moyens pour s'aider: l'utilisation du pseudo-code schématique, et répondre à quelques questions.

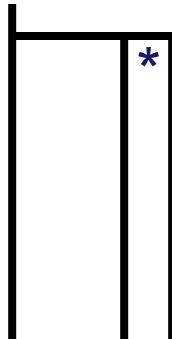
Description de tâches à partir des opérations élémentaires suivantes:

- 
- Lire
  - Écrire
  - TANT QUE condition FAIRE
  - POUR ensemble\_de\_données FAIRE
  - SI condition ALORS --- SINON
  - Affecter =
  - Additionner
  - Soustraire
  - Multiplier
  - Diviser
-

### Pseudo-code schématique - décisions



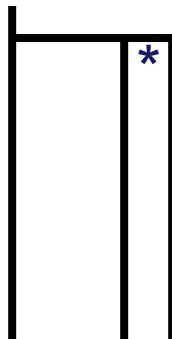
### Pseudo-code schématique - répétitions



TANT QUE condition FAIRE

Opération 1

Opération 2



POUR ensemble de données FAIRE

Opération 1

Opération 2

### Pseudo-code schématique – exemple #1

Lire un nombre et écrire la valeur absolue du nombre

- Quelles sont les entrées? **Un nombre.**
- Quelles sont les sorties? **Un nombre.**
- Comment trouvons-nous une valeur absolue? **Il faut changer le signe du nombre s'il est négatif.**
- Est-ce qu'il y a des répétitions ? **Non.**
- Est-ce que des décisions sont requises? **Oui.**
  - Combien? **1.**
  - Quelles sont les alternatives? **Inversion du signe si le nombre est négatif, sinon on fait rien.**
  - Elles influencent quelles opérations ? **Écriture du nombre.**

### Pseudo-code schématique – exemple #1

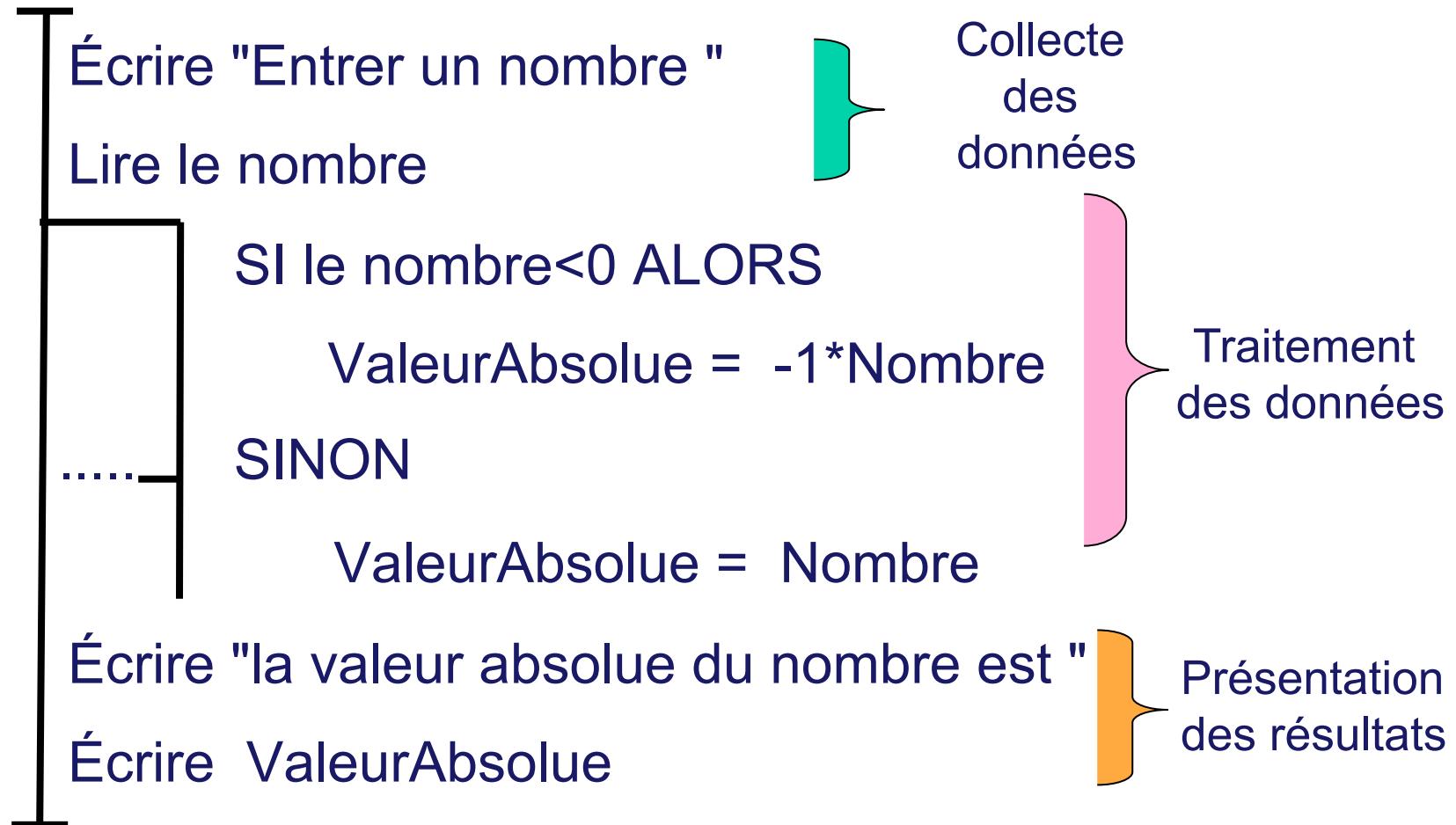
Pseudo-code

```
Écrire "Entrer un nombre"  
Lire le nombre  
SI le nombre<0 ALORS  
    Écrire -1*Nombr  
SINON  
    Écrire Nombre
```

Décalage pour démontrer la dépendance du test nombre<0

### Pseudo-code schématique – exemple #1

Autre façon:



### Pseudo-code schématique – exemple #2

Deviner un nombre choisi au hasard. Indiquer si le nombre proposé est inférieur ou supérieur au nombre à deviner.

- Quelles sont les entrées? **Un ou des nombres (dépend si on devine le nombre facilement ou non).**
- Quelles sont les sorties? **Bravo ou trop petit ou trop grand.**
  - Comment trouvons-nous un nombre choisi au hasard?  
**On propose des nombres tant qu'on n'a pas trouvé le bon.**
  - Est-ce qu'il y a des répétitions ? **Oui.**
    - Combien ? **Une.**
    - Et elles répètent quelles opérations? **Demander un nombre à l'utilisateur et vérifier ce nombre.**

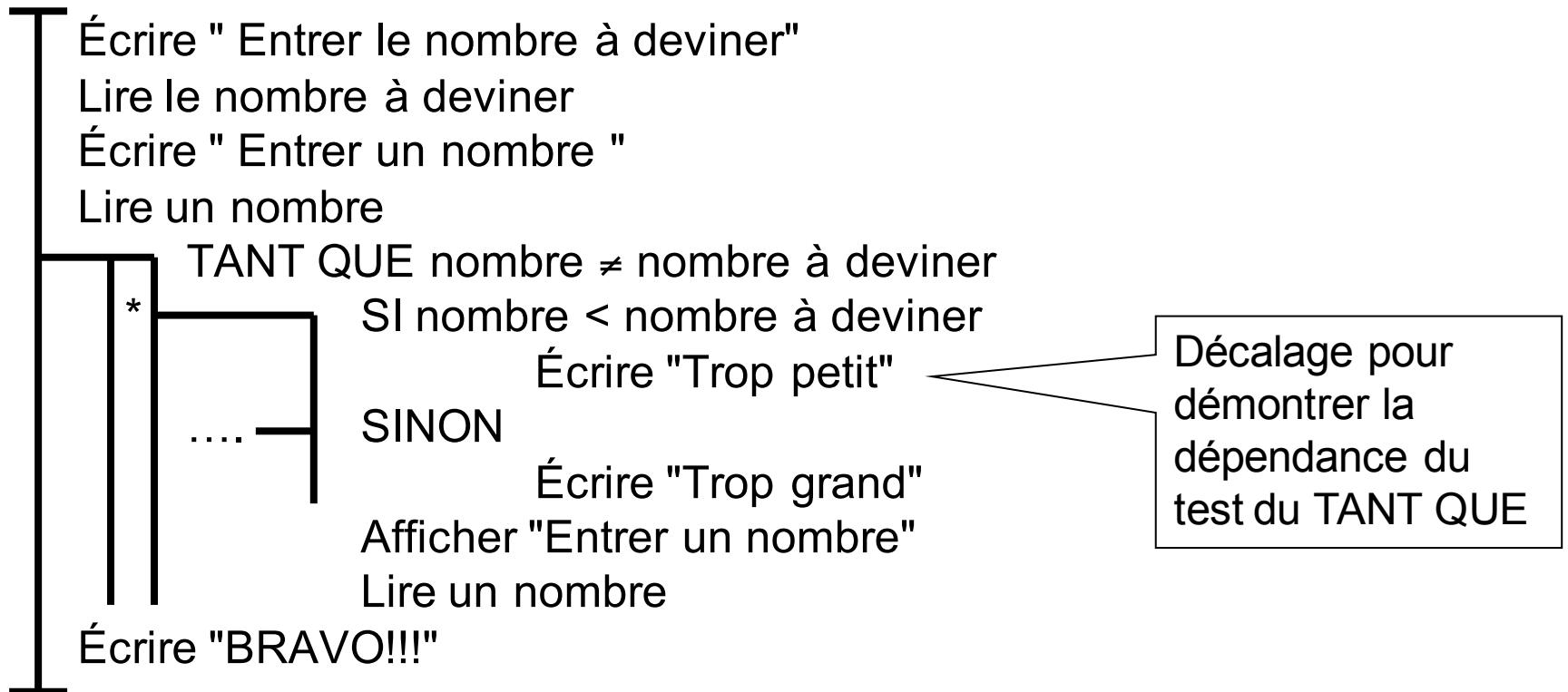
### Pseudo-code schématique – exemple #2

Deviner un nombre choisi au hasard (cont.).

- Est-ce que des décisions sont requises? **Oui.**
  - Combien? **Deux.**
  - Quelles sont les alternatives? **Si le nombre est bien deviné, terminer le programme et écrire à l'écran bravo, ou si le nombre n'est pas deviné, vérifier le sens de l'erreur et écrire trop petit, trop grand.**
  - Elles influencent quelles opérations ? **La répétition, car on recommence tant que le nombre n'est pas deviné, et aussi ce qui est écrit à l'écran (trop petit, trop grand).**

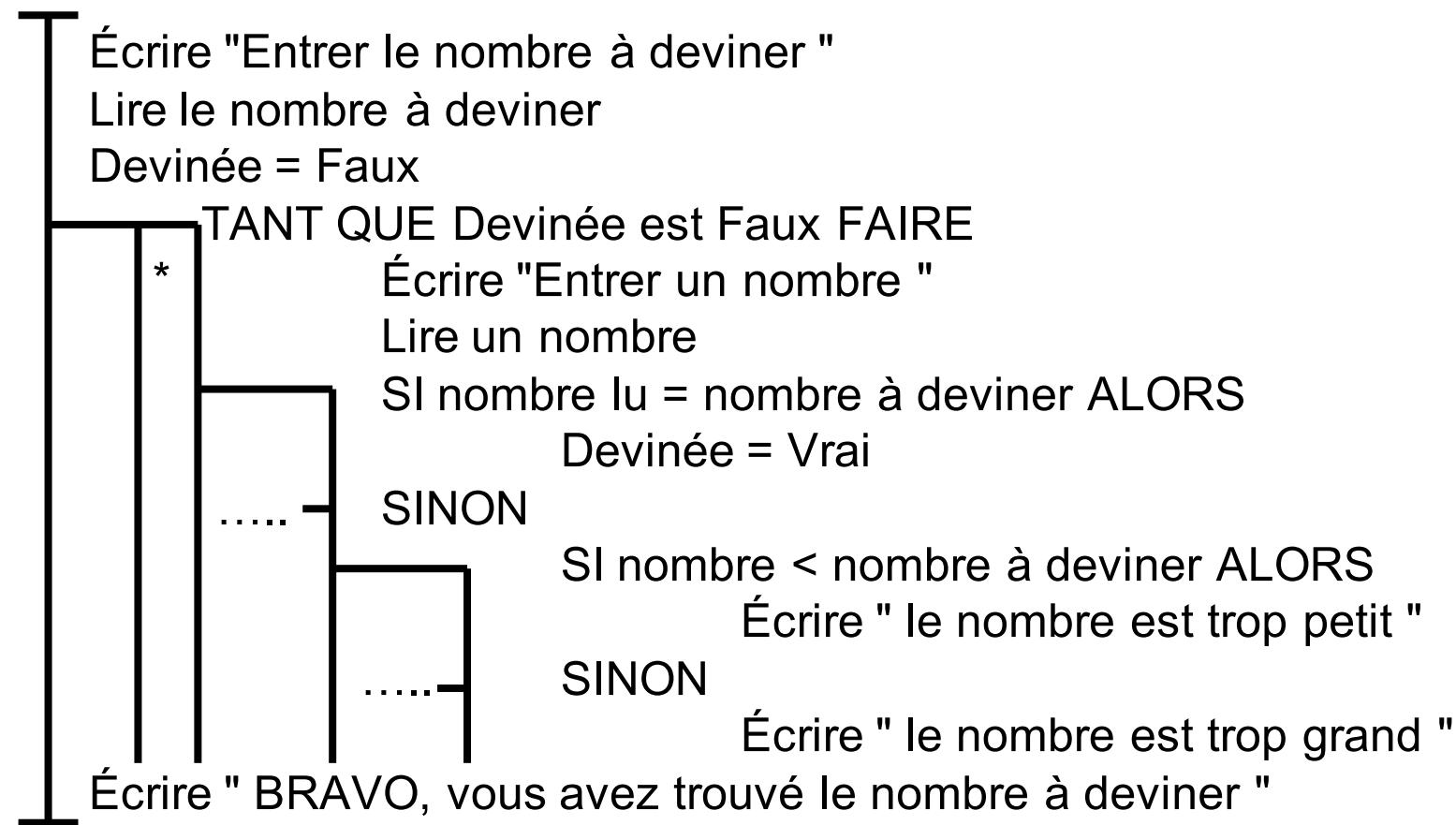
### Pseudo-code schématique – exemple #2

- Deviner un nombre choisi au hasard (Version 1)



### Pseudo-code schématique – exemple #2

- Deviner un nombre choisi au hasard (Version 2)



### Pseudo-code schématique – raffinement graduel

- Lors de l'élaboration d'un algorithme, il est souvent utile d'envisager un processus de raffinement graduel.
- Il s'agit dans un premier temps d'énumérer les opérations « évidentes » qui réalisent l'application désirée.
- Ces opérations évidentes ne sont pas nécessairement des opérations élémentaires.
- Par la suite il suffit de détailler les opérations non élémentaires à l'aide d'opérations élémentaires.

### Pseudo-code schématique – exemple #3

Lire trois nombres entiers quelconques et vérifier si l'un de ces nombres est le produit des 2 autres. Si 2, 10 et 5 sont lus, alors le deuxième nombre est le résultat de la multiplication des deux autres.

- Quelles sont les entrées? **Trois nombres.**
- Quelles sont les sorties? **Oui ou non.**
- Comment vérifie-t'on qu'un nombre est le produit de deux autres?  
**On vérifie si en multipliant deux des nombres, on obtient le troisième.**
- Est-ce qu'il y a des répétitions ? **Oui.**
  - Combien ? **Une.**
  - Et elles répètent quelles opérations? **La vérification du produit pour chaque combinaison.**

### Pseudo-code schématique – exemple #3

Lire trois nombres entiers quelconques et vérifier si l'un de ces nombres est le produit des 2 autres. Si 2, 10 et 5 sont lus, alors le deuxième nombre est le résultat de la multiplication des deux autres.

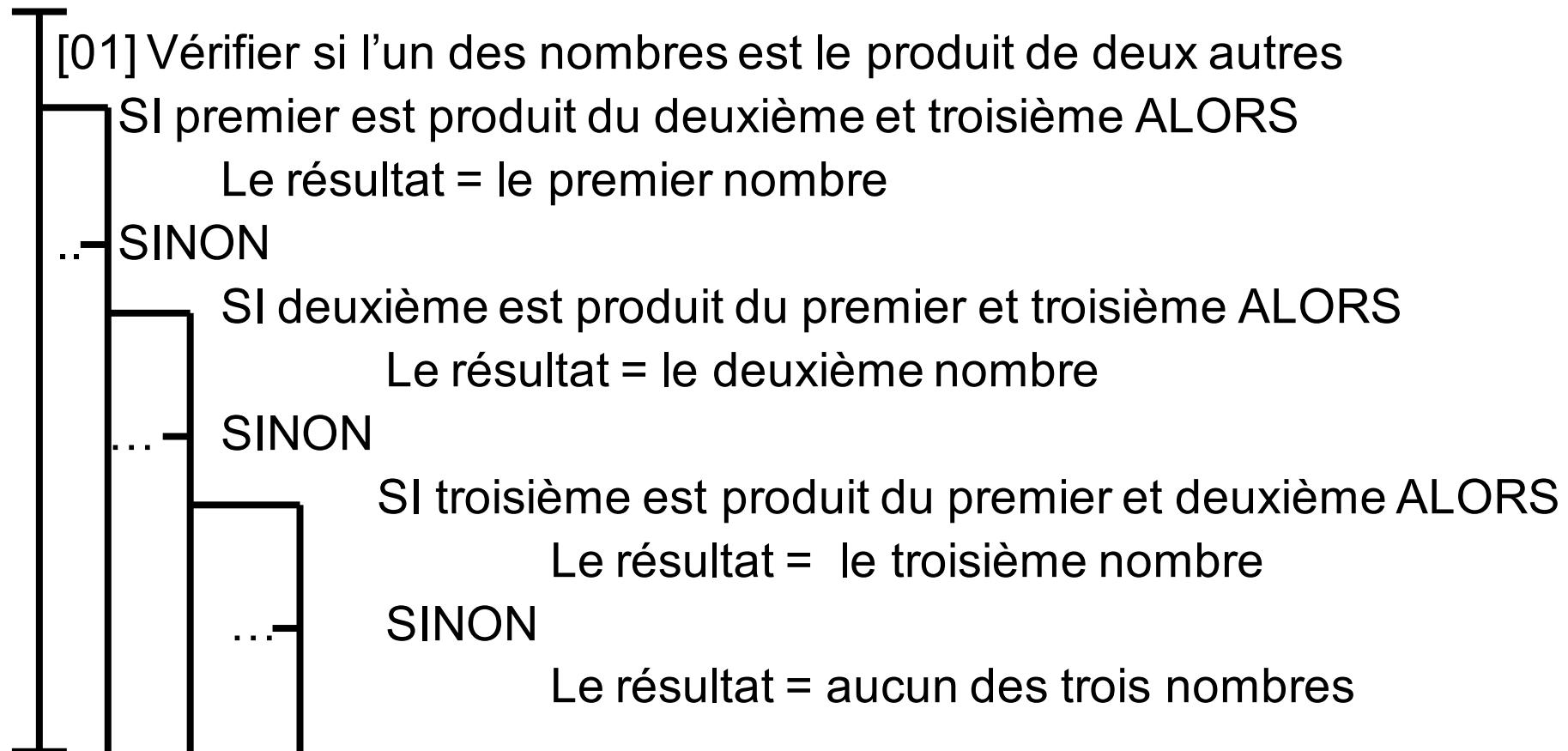
- Est-ce que des décisions sont requises? **Oui.**
- Combien? **Une par combinaison de nombres.**
- Quelles sont les alternatives? **Pour chaque combinaison de nombres (3), il faut vérifier si un nombre égale le produit des deux autres.**
- Elles influencent quelles opérations ? **La réponse finale.**

### Pseudo-code schématique – exemple #3

Écrire "Entrer trois nombres"  
Lire les trois nombres  
[01]Vérifier si l'un des nombres est le produit de deux autres  
Écrire le résultat

Opération à raffiner

### Pseudo-code schématique – exemple #3



### Pseudo-code schématique – exemple #3

**(Très, très important !) Astuce de débogage**

- En décomposant un programme par des opérations plus générales, il est plus facile d'identifier les erreurs de programmation. Très utile pour programmes plus complexes.
- Remplace l'opération qu'on croit qui ne fonctionne pas par un résultat typique de ce code. On sait alors si les autres opérations fonctionnent.

[01] Vérifier si l'un des nombres est le produit de deux autres  
Le résultat = le premier nombre % Résultat fictif

# Agenda



Expression booléenne, type logique



Structures de programmation, pseudo-code schématique

- Structures de décision
- Structures de répétition

### Structures de décision – **if** – **elseif** – **else**

En MATLAB:

```
if expression booléenne
    opérations; % exécutées si l'expression booléenne
    % est « vrai »
end
```

Exemple:

```
A=input ('Entrer un premier nombre ');
B=input ('Entrer un deuxième nombre ');
if A==B
    fprintf ('%d est égal à %d ',A,B);
end
```

### Structures de décision - **if** - **elseif** - **else**

En MATLAB:

```
if expression booléenne
    operations; % exécutées si l'expression booléenne
    % est « vrai »
else
    operations; % exécutées si l'expression booléenne
    % est « faux »
end
```

Exemple:

```
if A==B
    fprintf(' %d est égal à %d ',A,B);
else
    fprintf(' %d n\'est pas égal à %d ',A,B);
end
```

### Structures de décision - **if** - **elseif** - **else**

En MATLAB

**if** *expression booléenne 1*

*operations\_if; % exécutées si l'expression booléenne 1*  
    *% est « vrai »*

**elseif** *expression booléenne 2*

*operations\_elseif; % exécutées si l'expression booléenne*  
    *% 2 est « vrai »*

**else**

*operations\_else; % exécutées si aucune autre expression*  
    *% n'est « vrai »*

**end**

### Structures de décision - **if** - **elseif** - **else**

Exemple:

```
A=input('Entrer un premier nombre: ');\nB=input('Entrer un deuxième nombre: ');\nif A==B\n    fprintf('%d est égal à %d',A,B);\nelseif A<B\n    fprintf('%d est plus petit que %d',A,B);\nelse\n    fprintf('%d est plus grand que %d',A,B);\nend
```

# Structures de décision – opérateurs relationnels et logiques

Les opérateurs relationnels et logiques sont souvent utilisés dans les structures de décisions pour formuler plusieurs conditions à respecter.

Exemple:

```
if A ~= 0 & A/B > 18
    disp('OK');
else
    disp('erreur');
end
```

Rappel: La conjonction (`&` - et ) est prioritaire sur la disjonction (`|` - ou). Les opérateurs relationnels sont prioritaires sur le opérateurs logiques (Voir chapitre 3).

### Structures de décision – **switch** – **case**

**switch variable** (scalaire ou chaîne de caractères)

**case valeur\_1 % si variable est un scalaire: variable == valeur\_1**  
**% si chaîne de caractères on a une comparaison**  
**% (variable, valeur\_1)**

**operations\_1; % exécutées si la vérification faite est « vrai ».**  
**% les autres cas seront ignorés.**

**case valeur\_2 % vérifie variable et valeur\_2 (les mêmes**  
**% commentaires que celles du case valeur\_1)**

**operations\_2 ; % même commentaire que pour operations\_1**

...

**case valeur\_x**

**operations\_x;**

**otherwise**

**operations; % exécutées si aucune autre expression n'est «vrai»**

**end**

### Structures de décision – **switch** – **case**

Exemple:

```
Note=input('entrer une note (A,A*,B,C,D)', 's');
switch upper(Note)
    case {'A', 'A*' }      % Ensemble de cellules
        Points =4;
    case 'B'
        Points =3;
    case 'C'
        Points =2;
    case 'D'
        Points =1;
    otherwise
        Points =0;
end
fprintf('Cette note vaut %i',Points);
```

### Structures de décision – **if-else vs. switch - case**

Switch case	If-elseif
Facile à lire	Peut être difficile à lire
Peut comparer des chaînes de caractères de différentes longueurs	<p>Il est nécessaire d'utiliser <code>strcmp()</code> pour comparer des chaînes de caractères de différentes longueurs</p> 
Vérifie seulement l'égalité 	Vérifie l'égalité ou l'inégalité

# Agenda



Expression booléenne, type logique



Structures de programmation, pseudo-code schématique



Structures de décision

- Structures de répétition

### Structures de répétition - **while**

**%initialisation**

**while expression booléenne (critère) % la boucle est  
% répétée jusqu'au moment où l'expression devient fausse  
opérations;  
% incrémentation;  
end**

- une initialisation pour entrer ou non dans le while
- critère d'arrêt (expression booléenne)
- opérations affectant l'expression booléenne (pour terminer le while)

### Structures de répétition - **while**

Exemple:

```
A=24;
```

```
while A>1
```

```
    Somme=Somme+A;
```

```
    A=A+1;
```

```
end
```

il manque l'initialisation  
de la variable Somme

boucle infinie car A reste  
toujours >1

Trouvez les erreurs!

### Structures de répétition - **while**

Exemple:

A = 24;

pour que le `while` ait un sens A doit exister

Somme = 0;

A est dans l'expression booléenne

while A > 1

Somme = Somme + A;

A = A - 1;

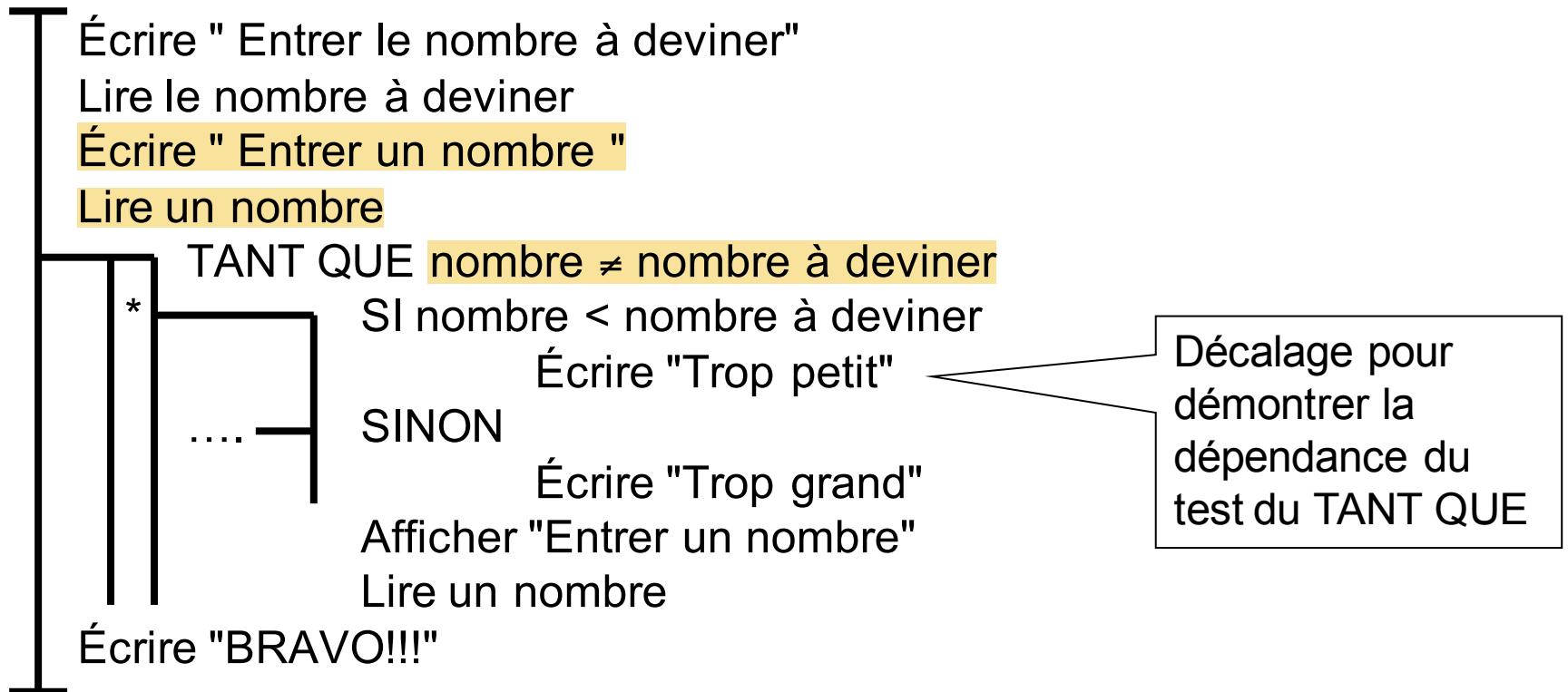
pour que le `while` se termine, A doit être modifiée de façon à ce que l'expression booléenne devienne éventuellement fausse.

end

Règle de base: les variables de l'expression booléenne doivent apparaître avant le `while` et à l'intérieur de celui-ci

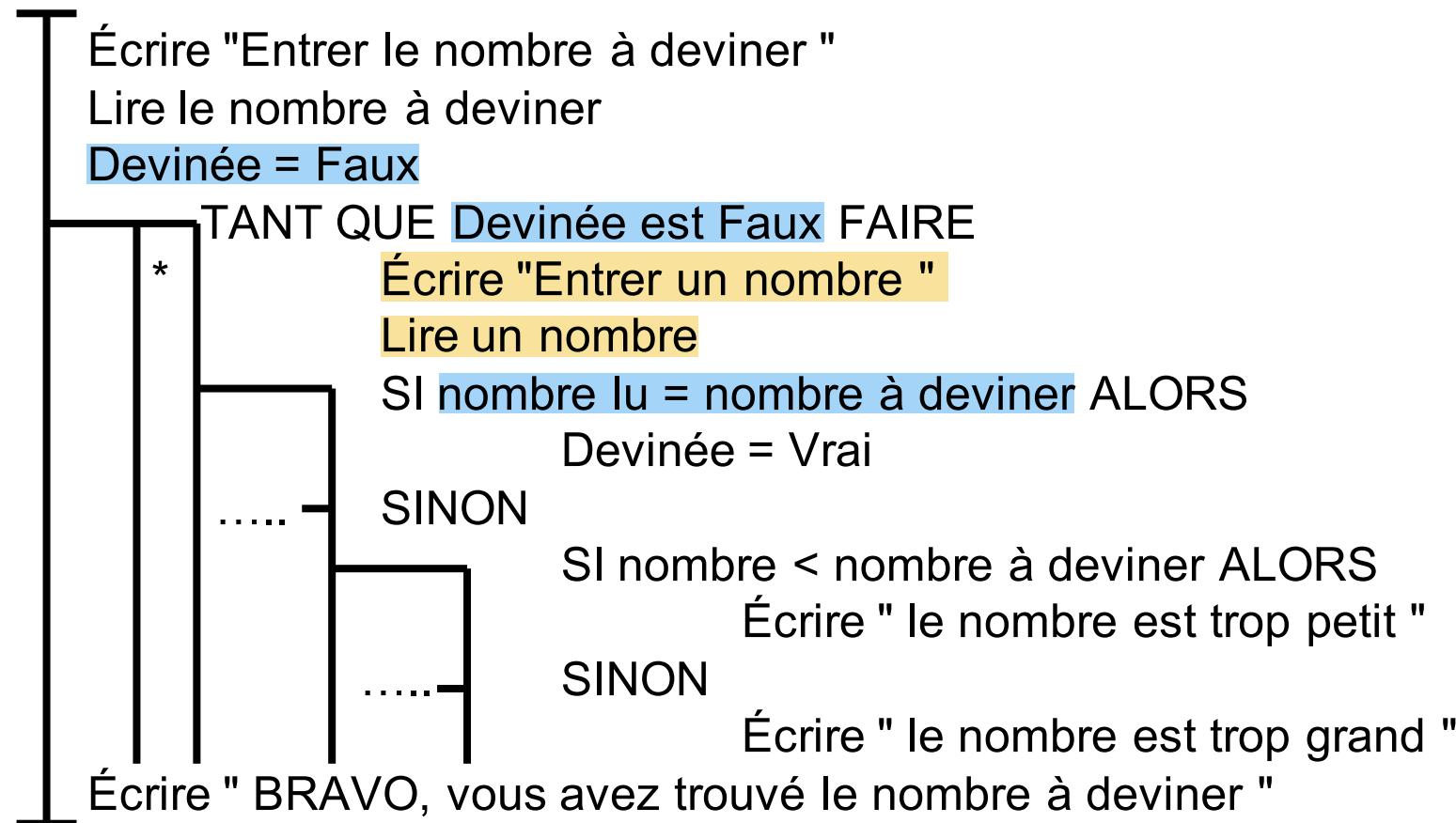
### Pseudo-code schématique – exemple #2

- Deviner un nombre choisi au hasard (Version 1)



### Pseudo-code schématique – exemple #2

- Deviner un nombre choisi au hasard (Version 2)



### Structures de répétition - **for**

1

2

**for** *variable* = [*matrice de valeurs*] % identification de  
% la boucle et définition d'une variable (indice) qui  
% se modifie chaque fois qu'on passe par la boucle

instructions;

**end**

1

Nom de la variable qui sera modifiée.

2

Valeurs de modification (chaque colonne de la matrice).

#### Fonctionnement:

- À chaque répétition de la structure de répétition, la prochaine colonne de *matrice\_de\_valeurs* est copiée dans *variable*.
- Il y a autant de répétitions que de colonnes dans *matrice\_de\_valeurs*.
- Les opérations à l'intérieur du **for** peuvent (pas obligatoire) utiliser les valeurs de la colonne qui est dans *variable*.
- Il est impossible de modifier *matrice\_de\_valeurs*.

# Structures de répétition - **for**

**Exemple:**

```
A=[ 1 18 4 5 ];
B=[ 4 5 ];
C=[];
for col=A
    Res=col*B(1)+col*B(2);
    C=[C Res];
end
```

4 colonnes =  
4 répétitions

Répétition 1: col=1  
Répétition 2: col=18  
Répétition 3: col=4  
Répétition 4: col=5

Répétition 1: C=9  
Répétition 2: C=[9 162]  
Répétition 3: C=[9 162 36]  
Répétition 4: C=[9 162 36 45]

### Structures de répétition – **while** vs. **for**

for	while
<p>Le nombre de répétitions est connu à l'avance, i.e. le nombre de colonnes de la matrice, même si la taille de la matrice change d'une exécution à l'autre</p>	<p>Le nombre de répétitions est généralement indéfini (ne peut pas être déterminé avant que l'exécution du while ne soit terminé). Il dépend du nombre d'opérations nécessaires pour que l'expression booléenne devienne fausse.</p>

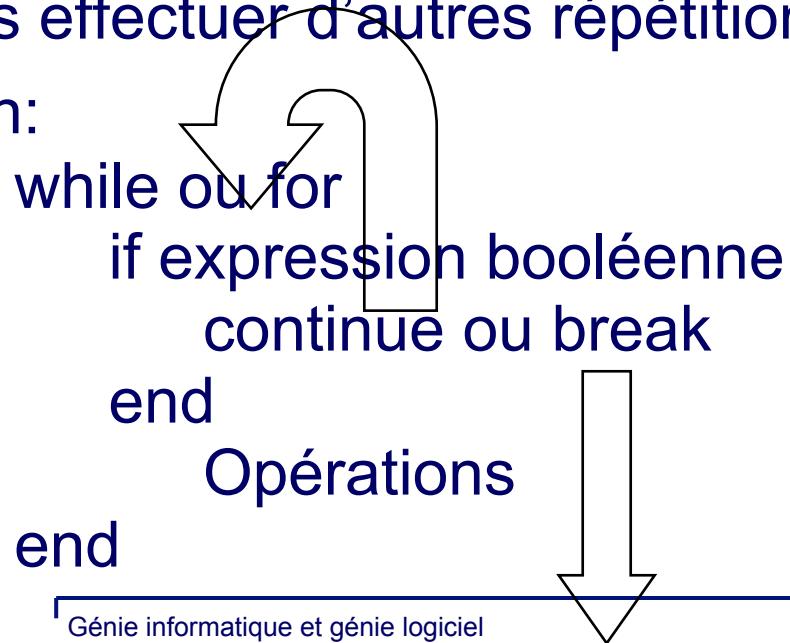
### Structures de répétition – **break()** et **continue()**

Utilisées seulement dans une structure de décision imbriquée dans une structure de répétition

Continue : Permet de sauter directement à la prochaine répétition.

Break : Permet de sortir immédiatement de la boucle sans effectuer d'autres répétitions.

Illustration:



# Agenda



Expression booléenne, type logique



Structures de programmation, pseudo-code schématique



Structures de décision



Structures de répétition

# Sommaire

- 1    **Expressions booléennes, type logique**
- 2    **Structures de programmation, pseudo-code schématique**
- 3    **Structures de décision**
- 4    **Structures de répétition**