ALGO

Ch 3 – Les structures de contrôle

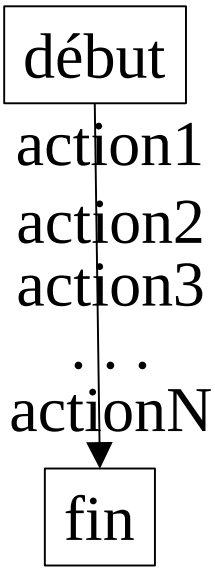
I. Introduction

A. Déroulement linéaire d'un algorithme : une séquence d'actions

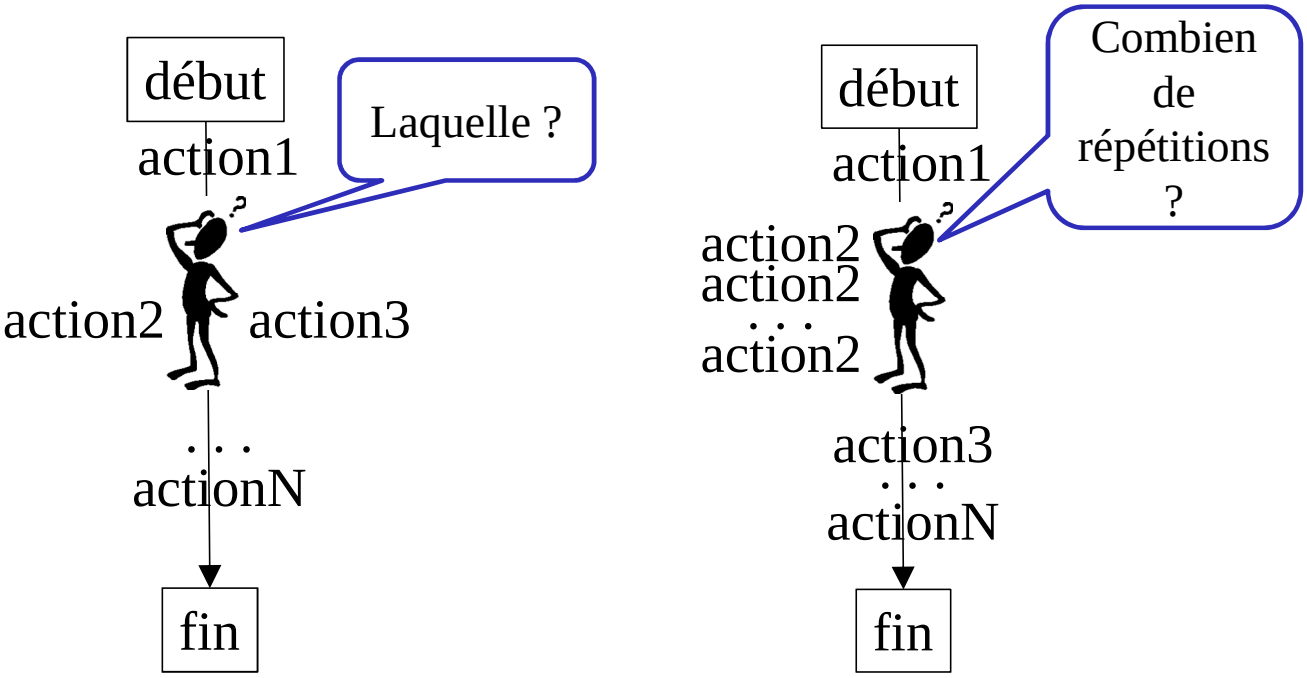
Nous avons étudié jusqu'à présent des algorithmes constitués par une succession non interrompue d'actions :

- Affectations, calculs
- Lectures et écritures d'informations.

Ces actions forment un bloc dont le déroulement est linéaire, ininterrompu.



Il arrive cependant que certains problèmes exigent l'interruption du déroulement séquentiel pour une prise de décision, ou bien pour une exécution répétée de certaines actions.



B. Rupture de l'exécution linéaire : choix et répétitions

Les **STRUCTURES DE CONTRÔLES** permettent la modification du déroulement d'un l'algorithme ; elles constituent une rupture dans l'exécution en séquence des actions en proposant

- un choix dans l'exécution de blocs d'actions : **STRUCTURES DE CONTRÔLES CONDITIONNELLES**,
- ou une répétition dans l'exécution d'un bloc d'actions : **STRUCTURES DE CONTRÔLES REPETITIVES**.

Ces **STRUCTURES SONT BASEES SUR L'EVALUATION D'EXPRESSIONS LOGIQUES**, plus simplement appelées **CONDITIONS**, ou **TESTS**.

II. Choisir : structures de contrôle conditionnelles

A. Structure conditionnelle simple

La **STRUCTURE CONDITIONNELLE SIMPLE** permet l'exécution d'un bloc d'instructions seulement **SI UNE CONDITION EST VRAIE** (= l'évaluation d'une expression logique a pour valeur VRAI), sinon aucune instruction n'est exécutée.

Syntaxe retenue :

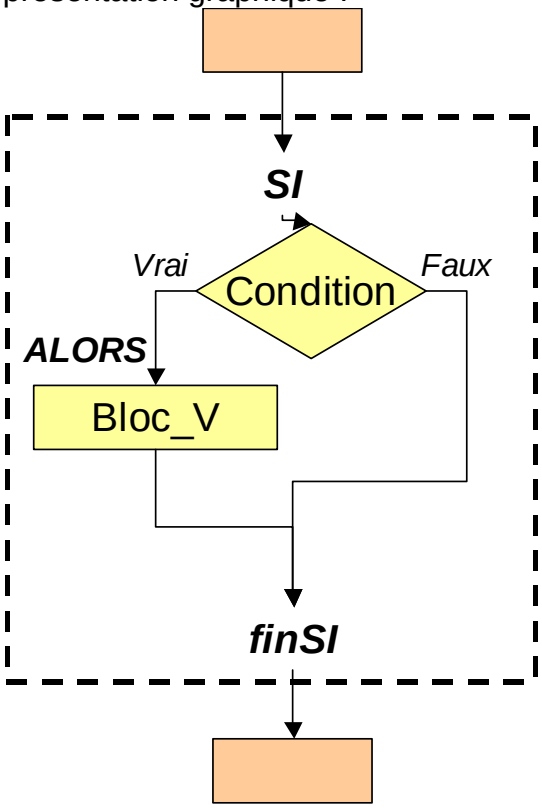
```
SI (expression_logique)
    ALORS
    |   ... bloc d'actions ...
finSI
```

- **expression_logique**

- représente l'expression logique à évaluer : c'est la condition qui va autoriser ou non l'exécution du bloc d'instruction qui suit le ALORS

- **bloc d'actions** :
  - bloc d'instructions à exécuter si 'expression_logique' a pour valeur VRAI

Représentation graphique :



Si la condition est VRAIE (=l'évaluation de 'Expr.Log.' est égal à VRAI), alors le bloc d'actions Bloc_V est exécuté, puis on poursuit l'exécution après la fin de la structure de contrôle (marquée par finSI).

Si la condition N'EST PAS VRAIE, on poursuit l'exécution après la fin de la structure de contrôle (marquée par finSI).

```
SI (age >= AGE_MAJORITE)
     ALORS
      |    AFFICHER("vous etes majeur !")
finSI
```

= « à condition que Vage soit plus grand ou égal à AGE_MAJORITE (18) alors j'affiche la chaîne de caractères : vous êtes majeur ! »

B. Structure conditionnelle alternative

La **STRUCTURE CONDITIONNELLE ALTERNATIVE** permet l'exécution d'un bloc d'actions si une expression logique a pour valeur VRAI et d'un autre bloc d'actions dans le cas contraire.
Elle offre le choix entre 2 possibilités d'action.

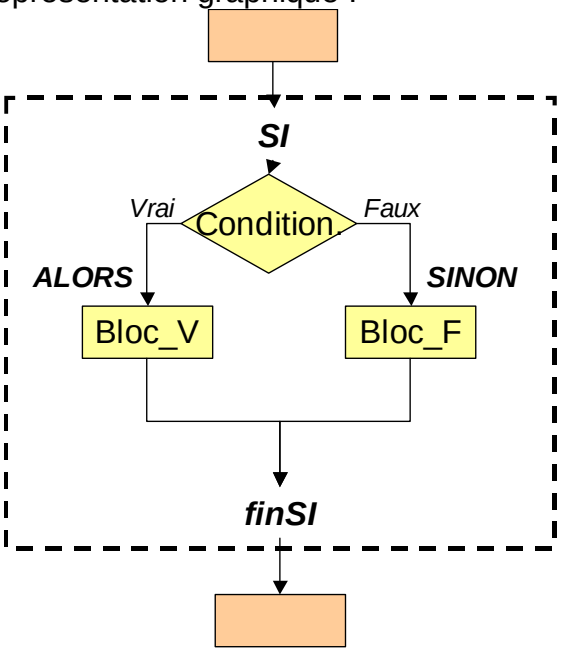
Syntaxe retenue :

```
SI (expression_logique)
     ALORS
      |   ... bloc d'actions 1...
     SINON
      |   ... bloc d'actions 2...
finSI
```

- **expression_logique**
  - représente l'expression logique à évaluer

- **bloc d'actions 1** :
  - représentent le bloc qui sera exécuté si 'expression_logique' a pour valeur VRAI
- **bloc d'actions 2** :
  - représentent le bloc qui sera exécuté si 'expression_logique' n'a pas pour valeur VRAI (sinon)

Représentation graphique :



Si la condition est VRAIE, ALORS le bloc Bloc_V est exécuté
SINON (la condition N'EST PAS VRAIE), le bloc d'instructions Bloc_F est exécuté

Exemple :

```
SI (age >= AGE_MAJORITE)
     ALORS
     |    AFFICHER("vous etes majeur !")
     SINON
     |    AFFICHER("vous etes mineur !")
finSI
```

C. Structure conditionnelle à choix multiples

La **STRUCTURE CONDITIONNELLE A CHOIX MULTIPLE** permet de choisir entre plus de 2 possibilités de blocs d'actions à exécuter en fonction de valeurs différentes d'une variable.

Syntaxe retenue :

```
SELON (nom_variable)
     CAS valeur1 :
          ... bloc d'actions 1...
     CAS valeur2 :
          ... bloc d'actions 2...
     CAS valeurN :
          ... bloc d'actions N...
     SINON
          ... bloc d'actions sinon...
finSELON
```

- **nom_variable**

- représente une variable

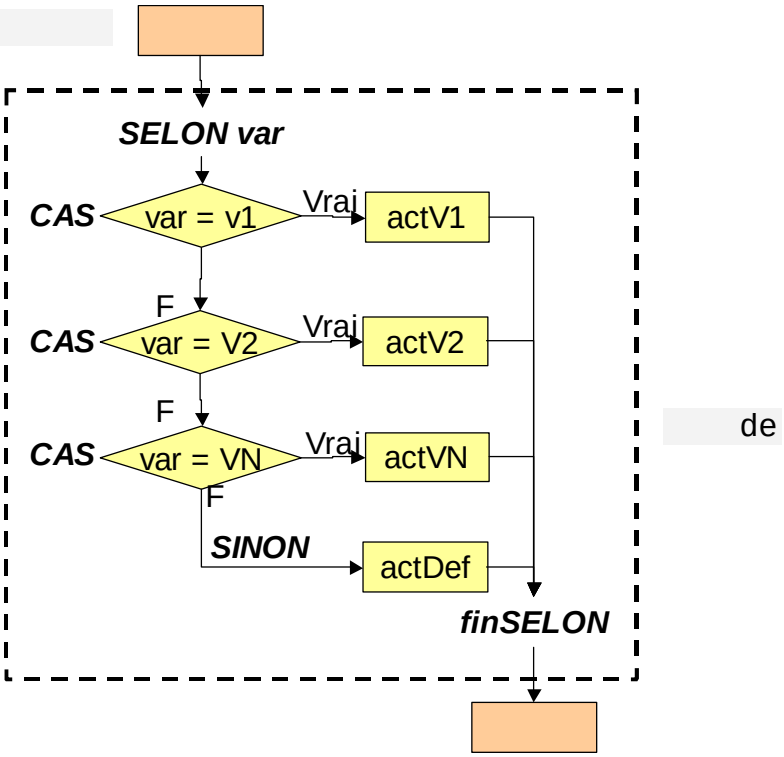
- **valeur1, valeur2, valeurN** :
  - représente chacune des valeurs à tester (**égalité**)
- **bloc d'actions 1, 2, N,** sinon :
  - représentent les blocs à exécuter dans chacun des cas : UN SEUL BLOC SERA EXECUTE

Exemple :

```
SELON (mois)
     CAS 1 :
          AFFICHER("Janvier ")
     CAS 2 :
          AFFICHER("Février")

     . . . autres cas . . .

     CAS 12 :
          AFFICHER("Décembre")
     SINON
          AFFICHER("le   numéro
mois est errone")
finSELON
```



de

III.Répéter : structures de contrôles répétitives, ou boucles

A. Généralités

Les **STRUCTURES REPETITIVES** permettent l'exécution répétée d'un bloc d'instructions. Le nombre de répétitions est contrôlé par une **CONDITION DE POURSUITE OU D'ARRET DE LA REPETITION.**

Synonymes : boucles, itérations (structures itératives)
La condition d'arrêt peut être exprimée de 2 manières logiques :

- Logique de continuité : tant que « quoi » la boucle continue-t-elle ?
- Logique d'arrêt : Quand la boucle s'arrête-elle ?

ATTENTION :
LA CONDITION D'ARRET (ou de poursuite) D'UNE BOUCLE DOIT ETRE PARFAITEMENT IDENTIFIEE.

B. Boucle TANT QUE - FAIRE

La boucle **TANT QUE** permet d'exécuter un bloc d'instructions **SI une expression logique est VRAIE et TANT QU'elle est vraie**. Le bloc d'actions s'exécutera 0 à N fois.

Syntaxe :

```
TANT QUE (expression_logique) FAIRE
    . . . Bloc d'instructions à répéter
finTANTQUE
```

- **Expression_logique** (=test, condition)
  - l'expression à évaluer : si VRAI, exécution du bloc d'instructions, et répétition de l'exécution TANT QUE cette expression logique a pour valeur VRAI
- **bloc d'instructions**:
  - bloc d'instructions à exécuter

```
AFFICHER("Entrez un nombre supérieur à 10 :")
SAISIR(Vnombre)
TANT QUE (Vnombre <= 10) FAIRE
    | AFFICHER("Entrez un nombre supérieur à 10 :")
    | SAISIR(Vnombre)
finTANTQUE
```

C'est ici la condition (Vnombre <= 10) qui va servir à déterminer la condition de poursuite de la boucle.

ATTENTION :
Les données permettant d'évaluer l'expression logique doivent avoir été initialisés auparavant ET devront être à nouveau initialisés dans le bloc d'instructions (sinon on obtient une boucle infinie !)

C. Boucle FAIRE - TANT QUE

La boucle **FAIRE … TANT QUE** permet d'exécuter un bloc d'instructions **AU MOINS UNE FOIS ET TANT QU'UNE CONDITION EST VRAIE**.
Le bloc d'actions s'exécutera de 1 à N fois.

Syntaxe :

```
FAIRE
    … Bloc d'action(s) à répéter
TANT QUE (expression_logique)
```

- **Expression_logique**
  - l'expression à évaluer : exécution du bloc d'instructions 1 fois, et répétition éventuelle TANT QUE cette expression logique a pour valeur VRAI
- **bloc d'instructions**:
  - bloc d'instructions à exécuter

```
FAIRE
    | AFFICHER("Entrez un nombre supérieur à 10 :")
```

```
     SAISIR(Vnombre)
TANT QUE (Vnombre <= 10)
```

C'est ici la variable (Vnombre<=10) qui va servir à déterminer la condition de poursuite de la boucle plus d'une fois.

ATTENTION :
Les données permettant d'évaluer l'expression logique doivent être initialisés dans le bloc d'instructions (sinon on obtient une boucle infinie !)

D. Boucle POUR - FAIRE

La boucle **POUR** permet de **REPETER** l'exécution d'un bloc d'instructions **UN NOMBRE DE FOIS DETERMINE**.
Le décompte du nombre de fois est réalisé en faisant varier une **variable de boucle** : cette dernière permet de compter le nombre de boucles et sert à déterminer la condition d'arrêt.
LE NOMBRE D'ITERATIONS EST CONNU.

Syntaxe :

```
POUR var_boucle DE val_deb A val_fin PAS DE val_pas FAIRE
    . . . Bloc d'instructions à répéter
finPOUR
```

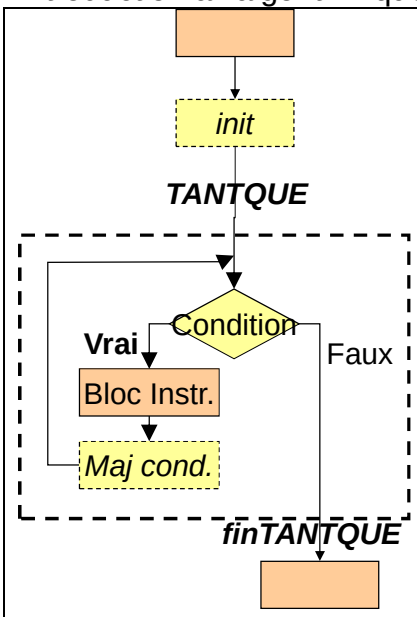
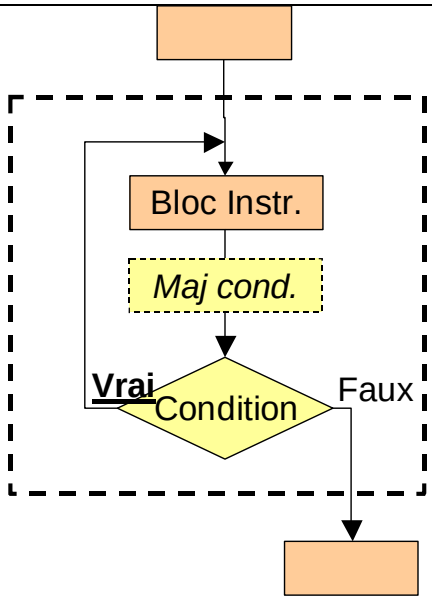
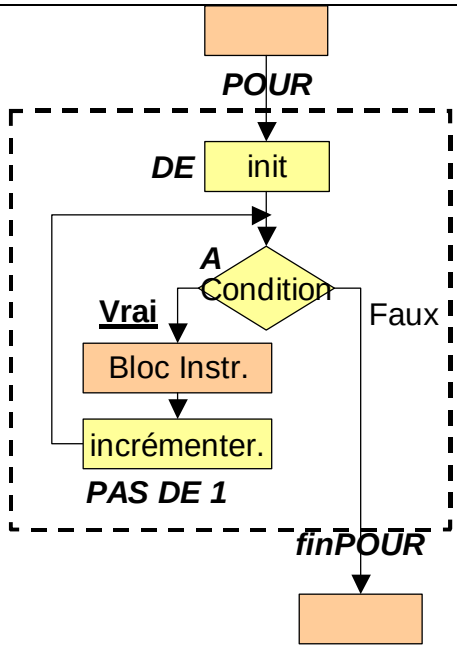
- **var_boucle**
  - Représente le nom de la variable numérique (ENTIER) qui va servir de compteur de boucle (variable de contrôle de boucle) (anglais loop counter)
- **val_deb** : valeur initiale de 'var_boucle' (anglais initial value)
- **val_fin** : valeur finale de 'var_boucle' : la boucle va s'arrêter quand 'var_boucle > val_fin' ou va continuer tant que 'var_boucle <= val_fin' (anglais loop-continuation condition) ; cette valeur détermine si oui ou non une boucle doit continuer ou s'arrêter
- **val_pas** : c'est le pas d'incrémentation (ou de décrémentation s'il est négatif)de 'var_boucle' à chaque itération (anglais increment / decrement) ; cette valeur modifie la variable de contrôle de boucle à chaque passage = à chaque itération de la boucle (anglais in each iteration of the loop)

```
POUR i DE 1 A 10 PAS DE 1 FAIRE
    |   AFFICHER("Vous etes à la boucle de numero :")
    |   AFFICHER(i)
finPOUR
```

C'est ici la variable i qui va servir à déterminer la condition de poursuite de la boucle (la condition i <= 10) (= tant que i est inférieur ou égal à 10)

E. Comparaison des boucles

TANT QUE – FAIRE	FAIRE - TANT QUE	POUR - FAIRE

		
Init : représente l'initialisation d'une variable utilisée dans l'évaluation de la condition (cela doit être réalisé impérativement une fois le premier test)	*Maj cond* : représente la modification d'une valeur utilisée dans l'évaluation de la condition	*Init* : représente l'initialisation de la variable de boucle avec la valeur initiale *Incrémenter* : représente l'incrémentation de la variable de boucle avec le pas
0 à N itérations, N étant déterminé par la modification de la valeur d'une variable au sein de la boucle	1 à N itérations, N étant déterminé par la modification de la valeur d'une variable au sein de la boucle	0 à N itérations, N étant déterminé grâce à la variable de boucle, connue à l'avance (avant de « boucler »

F. Boucle FAIRE … JUSQU'A

Dans les 2 formes de boucles précédentes (TANT QUE - FAIRE, FAIRE - TANT QUE), LA REPETITION EST POURSUIVIE SI UNE EXPRESSION LOGIQUE EST VRAIE, ET TANT QU'ELLE EST VRAIE.

Dans la forme REPETER - JUSQU'A, LA REPETITION EST ARRETEE dès QU'UNE EXPRESSION LOGIQUE EST VRAIE.

La boucle **REPETER … JUSQU'A** permet d'exécuter un bloc d'instructions AU MOINS UNE FOIS et **JUSQU'A** CE QU'UNE EXPRESSION LOGIQUE SOIT VRAIE.
Le bloc d'actions s'exécutera 1 à N fois.

Syntaxe :

```
REPETER
    . . . Bloc d'action(s) à répéter
JUSQU'A (expression_logique)
```

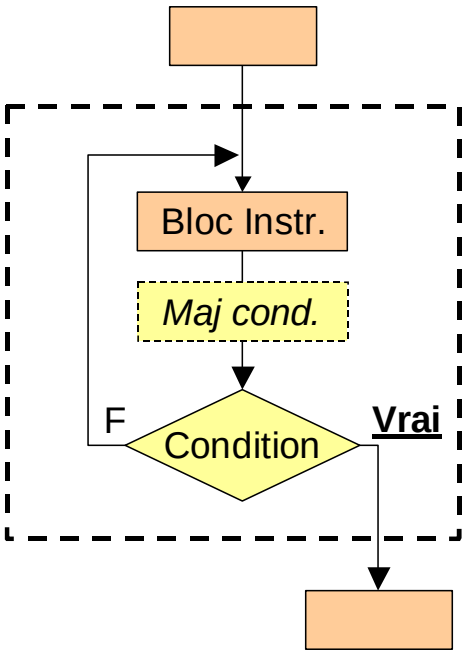
- **Expression_logique**
  - représente l'expression à évaluer : **si cette expression a pour valeur VRAI, ARRET DE l'exécution du bloc d'instructions**
- **bloc d'instructions**:
  - bloc d'instructions à exécuter

```
FAIRE
    |   AFFICHER("Entrez un nombre supérieur à 10 :")
    |   SAISIR(Vnombre)
JUSQU'A (Vnombre > 10)
```

C'est ici la variable Vnombre (Vnombre > 10) qui va servir à déterminer l'arrêt de la répétition.

ATTENTION :

Les données permettant d'évaluer la condition doivent être modifiées dans la boucle (bloc Maj cond.) sinon on obtient une boucle infinie !



G. Equivalences des structures répétitives (itératives)

Problème :
On souhaite construire un algorithme qui affiche les nombres de 1 à Vnombre, Vnombre étant saisi par l'utilisateur.

```
SAISIR(Vnombre)
```

POUR (= structure la plus adaptée dans ce cas)

```
POUR i DE 1 A Vnombre PAS DE 1 FAIRE
   |   AFFICHER(i)
finPOUR
```

TANT QUE – FAIRE

```
i ← 1
TANT QUE (i <= Vnombre) FAIRE
   | AFFICHER(i)
   | i ← i + 1
finTANTQUE
```

(i <= Vnombre) : si c'est VRAI la 1ère fois (et tant que c'est VRAI), je répète l'exécution

FAIRE – TANT QUE

```
i ← 1
FAIRE
   | AFFICHER(i)
   | i ← i + 1
TANT QUE (i <= Vnombre)
```

(i <= Vnombre) : j'exécute le bloc 1 fois, puis, tant que c'est vrai, je répète l'exécution

*******ATTENTION : Si l'utilisateur saisit la valeur '0', on passe une fois dans la boucle : pour palier à ce défaut, il faut ajouter une structure conditionnelle :***

```
SI (Vnombre >= 1) ALORS
   |  i ← 1
   |  FAIRE
   |     | AFFICHER(i)
   |     | i ← i + 1
   |  TANT QUE (i <= Vnombre)
finSI
```

FAIRE – JUSQU'A / REPETER – JUSQU'A

```
i ← 1
FAIRE
   | AFFICHER(i)
   | i ← i + 1
JUSQU'A (i > Vnombre)
```

(i > Vnombre) : j'exécute le bloc 1 fois, puis tant que c'est vrai, je répète

****Si l'utilisateur saisit la valeur '0', on passe une fois dans la boucle ; (idem. ci-dessus)

Pour passer d'une boucle TANT QUE à une boucle JUSQU'A, il faut inverser le test.

I. Structures imbriquées

Les blocs exécutés au sein des structures de contrôle peuvent comporter eux-mêmes des structures de contrôle imbriquées (et ainsi de suite).
Attention cependant au nombre de niveau d'imbrications, préjudiciable à l'intelligibilité des algorithmes.

A. Structure conditionnelles imbriquées

Exemple syntaxe 1 :

```
SI (expression_logique 1)
    ALORS
    |   SI (expression_logique 2)
    |       ALORS
    |       |... bloc d'instructions (1 ET 2)
    |       SINON
    |       |... bloc d'instructions (1 ET NON 2)
    |    finSI
    SINON
    |   SI (expression_logique 3)
    |       ALORS
    |       |... bloc d'instructions (NON 1 ET 3)
    |       SINON
    |       |... bloc d'instructions (NON 1 ET NON 3)
    |    finSI
finSI
```

Exemple syntaxe 2 :

```
SI (expression_logique 1)
        ALORS
        ... bloc d'instructions 1
    SINON SI (expression_logique 2)
        ALORS
          ... bloc d'instructions 2
    SINON SI (expression_logique 3)
        ALORS
          ... bloc d'instructions 3
    SINON
          ... bloc d'instructions N
finSI
```