Université A. Mira Béjaia
Faculté de Sciences Exactes
Département Mathématiques
L1 Math (tronc commun)
2025/2026

Module : Algorithmique et Structures de Données 1 (ASD1)

Chapitre 3 : Structure de contrôle

Les structures de contrôle encore appelées **instructions structurées** permettent d'exprimer comment s'enchaînent les instructions d'un algorithme. Il existe **trois structures** : La **séquence, la sélection (structure conditionnelle) et l'itération (boucle).** Elles permettent d'exprimer tous les enchaînements possibles dans un algorithme.

1. La séquence

La séquence exprime un enchaînement séquentiel des instructions. Elle se présente sous forme d'une suite ordonnée d'instructions regroupée en un bloc.

Syntaxe : Instruction 1
Instruction 2
…
Instruction N

- Tout algorithme est défini par une séquence d'instructions délimitée par les mots clés **Début et Fin.**
- Les instructions d'une séquence sont exécutées séquentiellement (l'une après l'autre) dans **l'ordre ou elles se présentent**. Aucune instruction n'est ignorée.

2. Structure de contrôle conditionnelle

La sélection exprime un enchaînement **conditionnel sélectif**. Elle offre la possibilité de sélectionner pendant l'exécution la prochaine séquence (Bloc d'instructions) à exécuter. La sélection se fait sur la base **d'une condition.**

2.1. La conditionnelle simple (Si…..alors…..finsi ;)

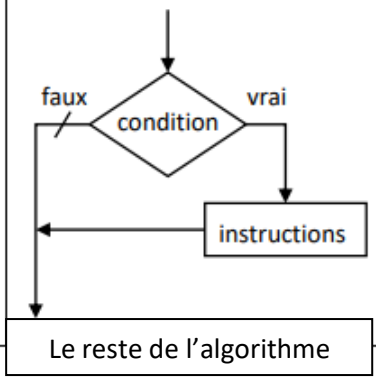
Une condition est évaluée pour déterminer si l'action ou le groupe d'actions peut être exécuté.

- Où ***condition*** est une expression **booléenne** (son évaluation donne soit **vrai ou faux**)

Fonctionnement :

- On commence par **évaluer** (calculer) **la condition**.
- Si la condition est **vérifiée (= Vrai)** alors le bloc d'instructions est exécuté. **Sinon (= Faux),** il est **ignoré**.

Syntaxe algorithmique	Syntaxe langage C	Organigramme

Si (condition) **alors**
bloc d'instructions;
finSi;
	`if(condition){`
	`bloc d'instructions ;`
	`}`
		

Exemple : écrire un algorithme permettant de calculer la valeur absolue d'un nombre lue au clavier.

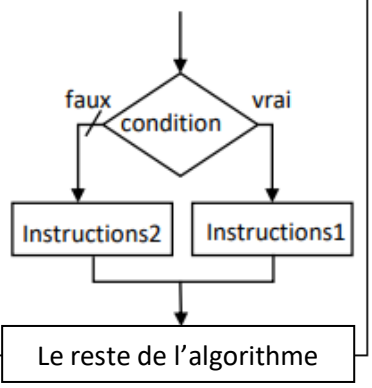
```
Algorithme Absolue;
Var v : reel;
Debut
   Lire(v);
      Si (v<0) alors
         v ← -1*v ;
      finSi ;
Ecrire('Absolue(v) = ',v);
Fin.
```

```
#include <stdio.h>
int main(){
int v;
     scanf("%d",&v);
     if(v<0){
          v=-v;
     }
     printf("Absolue(v) = %d",v);
     return 0;
}
```

2.2. La conditionnelle complète (Si….alors….Sinon….finsi ;)

Une condition est testée pour déterminer quelle action ou quel groupe d'actions doit être exécuté.

Syntaxe algorithmique	Syntaxe langage C	Organigramme

Si (condition) **alors**
bloc d'instructions 1;
sinon
bloc d'instructions 2;
finSi;
	`if(condition){`
	`bloc d'instructions1;`
	`else`
	`bloc d'instructions2;`
	`}`
		

Fonctionnement:

- La condition est évaluée.
- Si la condition **est vraie**, le bloc **d'instruction 1** est exécuté et l'exécution se poursuit après finSi. Le **bloc d'instruction 2 est ignoré**.
- Si la condition **est fausse**, le bloc **d'instruction 1 est ignoré** et l'exécution se poursuit **après sinon.**

Exemple 1 : écrire un algorithme permettant d'afficher le maximum de deux nombres lus au clavier.

```
Algorithme Max;
Var a,b: reel;
Debut
   Lire (a,b);
   Si (a>=b) alors
        Ecrire('la valeur max = ',a);
   Sinon
        Ecrire('la valeur max = ',b);
   FinSi ;
Fin.
```

Exemple 2: Ecrire un algorithme qui permet de lire un nombre entier N et d'afficher Positif (Si N ≥ 0) ou Négatif (N < 0).

```
Algorithme signe_nombre ;
Var N : entier ;
Debut
   Lire (N) ;
   Si (N≥ 0) alors
      Ecrire ('Nombre positif') ;
   Sinon
     Ecrire ('Nombre négatif') ;
   Finsi ;
Fin.
```

2.3. Structures conditionnelles imbriquées :

Dans une instruction conditionnelle qu'elle soit simple ou composée, les blocs d'instructions peuvent contenir des instructions conditionnelles. Dans ce cas, les instructions conditionnelles sont imbriquées.

Syntaxe algorithmique	Syntaxe langage C

Si (condition 1) alors
Si(condition 2) alors
bloc d'instructions 1;
sinon
bloc d'instructions 2;
finSi ;
sinon
Si(condition 3) alors
bloc d'instructions 3;
sinon
bloc d'instructions 4;
finSi ;
finSi;
	if(condition 1)
	if(condition2){
	bloc d'instructions 1;
	else
	bloc d'instructions 2;
	}
	else if(condition3){
	bloc d'instructions3;
	else
	bloc d'instructions4;
	}

Exemple: écrire un algorithme permettant d'afficher le maximum de trois nombres lus au clavier.

```
Algorithme Max;
Var a, b, c, max: reel;
Debut
     Lire(a,b,c);
        Si (a>b) alors
          Si(a>c) alors
            max ← a ;
            Sinon
            max ← c ;
          finSi ;
          sinon
          Si(b>c) alors
            max ← b ;
            Sinon
            max ← c ;
          finSi ;
        finSi ;
     ecrire(max) ;
Fin.
```

```
#include <stdio.h>
int main(){
float a,b,c,max;
scanf("%f",&a);
scanf("%f",&b);
scanf("%f",&c);

   if(a>b)
     if(a>c)
        max=a;
       else
        max=c ;
   else
     if(b>c)
        max=b;
       else
        max=c ;
   printf("max = %f",max);
     return 0;
}
```

2.4. Structures conditionnelles de choix multiples :

Parfois, Il est nécessaire **de choisir** un traitement parmi **plusieurs en fonction de la valeur** d'une variable **entière ou caractère**. On peut utiliser une succession d'instruction **Si Sinon, mais** il est préférable d'utiliser **un branchement direct** vers le **traitement voulu** à l'aide de la structure « **Selonque** ».

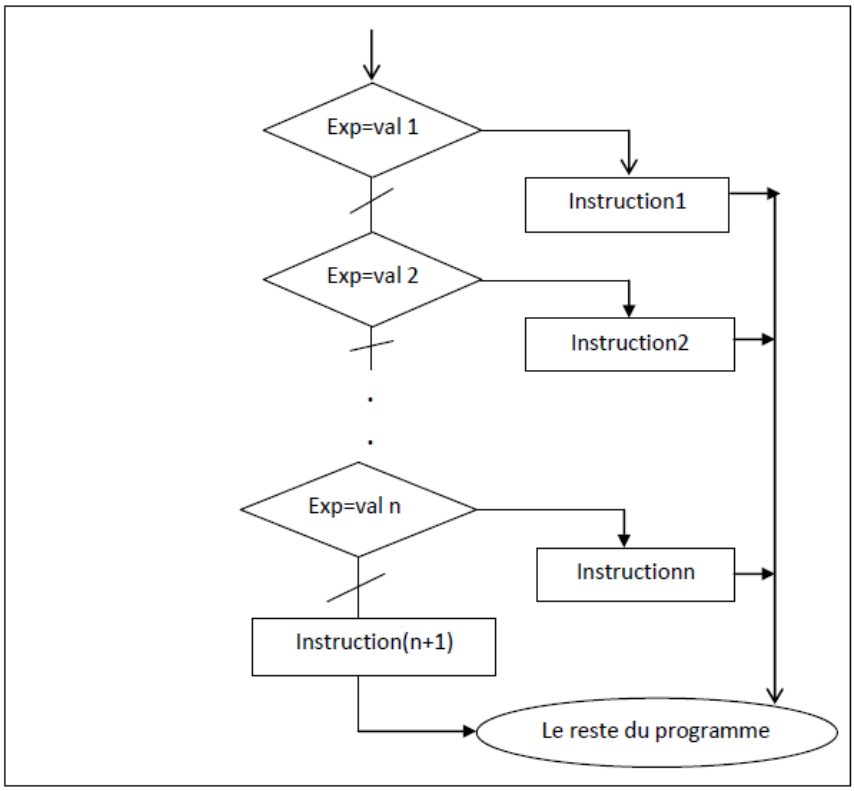
Syntaxe algorithmique	Syntaxe langage C

Selonque (variable) **faire**
V1 : bloc d'instructions 1 ;
V2 : bloc d'instructions 2 ;
V3 : bloc d'instructions 3 ;
.
.
.
Sinon
bloc d'instructions n ;
finSelonque ;
	switch(variable){
	case V1:bloc d'instructions 1;
	break;
	case V2:bloc d'instructions 2;
	break;
	case V3:bloc d'instructions 3;
	break;
	.
	.
	.
	default: bloc d'instructions n;
	}

❖ **Fonctionnement:**

- Le bloc d'instruction à exécuter est choisi selon la valeur de la variable de contrôle.

- On compare expression avec **valeur1**, si elles sont égales instruction1 est exécutée sinon on passe à **valeur2**, et ainsi de suite jusqu'à ce qu'on termine avec toutes les valeurs. Si **aucune valeur** ne correspond à expression, **la clause sinon** sera exécutée.
- La clause sinon n'est pas obligatoire, si elle n'est pas spécifiée et qu'aucune valeur ne correspond à expression, l'instruction suivante est exécutée.
- **L'organigramme correspondant**



Exemple : écrire un algorithme qui lit un entier au clavier et affiche le jour lui correspondant.

```
Algorithme jour;
Var n : entier ;
 Debut
  Lire (n) ;
  Selonque (n) faire
     1 : ecrire('Dimanche') ;
     2 : ecrire('Lundi') ;
     3 : ecrire('Mardi') ;
     4 : ecrire('Mercredi') ;
     5 : ecrire('Jeudi') ;
     6 : ecrire('Vendredi') ;
     7 : ecrire('Samedi') ;
   Sinon
     ecrire('ce n'est pas un jour') ;
  finSelonque ;
Fin.
```

3. Les instructions itératives (les boucles)

Les structures répétitives appelées souvent **les boucles** permettent de répéter l'exécution d'une instruction ou d'un bloc d'instructions **plusieurs fois**. Les boucles sont très utiles pour la programmation, ils permettent l'automatisation des calculs. Il y'a trois types de boucles: **pour, tant que, répéter.**

3.1. La boucle Pour

- Elle permet de répéter l'exécution d'un bloc d'instructions un certain nombre de fois connu à l'avance.
- Une **variable de contrôle** de type entier est utilisée pour **contrôler le nombre d'itérations** de la boucle. Le nombre d'itérations d'une boucle est **le nombre de fois** que les instructions de la boucle sont exécutées.
- Il est interdit de modifier manuellement la valeur du variable_contrôle dans la boucle.

❖ **La syntaxe**

Algorithmique	Langage C

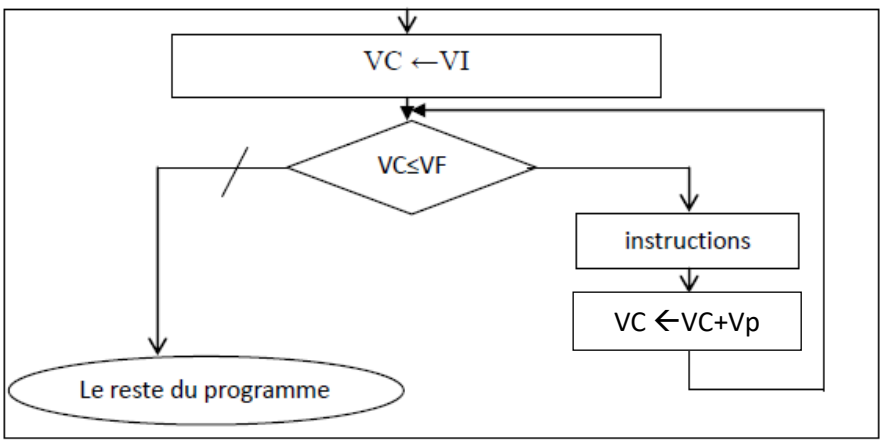
Pour VC **allant de** Vi **jusqua** Vf Pas=VP **faire**
Bloc instructions;
Finpour;
	For (exp1 ; exp2 ; exp3) {
	Bloc instructions
	}

• **VC : variable_controle (compteur)**, • **Vi** : valeur_initiale, • **Vf :** valeur_finale doivent être du même type.	**exp1** : l'expression d'initialisation ; **exp2** : une expression logique qui contient la **condition d'arrêt** de la boucle, si = faux, l'exécution de la boucle s'arrête**.** **exp3 :** elle est utilisée pour incrémenter la variable compteur (VC) de la boucle.

❖ **Fonctionnement**

- La variable compteur cpt est initialisée à la valeur Vi. Tant que la variable_contrôle appartient à l'intervalle [valeur_initiale, valeur_finale], le bloc d'instructions doivent être re-exécutées, et la variable VC prend la valeur **VC+Vp**.
- Le processus recommence ensuite pour exécuter la prochaine itération. Si **VC** >Vf, l'exécution de la boucle s'arrête et l'exécution de l'algorithme continue après finpour. Le champ **Pas** est facultatif, lorsque il est omis, sa valeur par défaut est 1.

❖ **L'organigramme correspondant**



Exemple : l'algorithme suivant permet d'afficher **les nombres entiers** compris entre la valeur **1** et la valeur **d'une variable n** lue au clavier.

```
Algorithme Affiche_Nombres ;
Var N : entier ;
Debut
  Lire (N) ;
  Pour i allant de 1 a n faire
     Ecrire(i);
  Finpour;
Fin.
```

- L'exécution de cet algorithme avec une valeur de **N = 10**, affiche à l'écran la suite : **1 2 3 4 5 6 7 8 9 10.**
- Dans cet algorithme le champ **Pas** est absent alors sa valeur par **défaut est 1.**

3.2. La boucle Tantque

La boucle **Tantque** permet de répéter l'exécution d'une ou de plusieurs instructions tant qu'une **condition est vérifiée** (= Vrai). Le contrôle de la boucle **Tantque** est assuré avec une **variable booléenne** ou **une expression logique.**

❖ **La syntaxe**

Algorithmique	Langage C

`Tantque (expression) faire`
`  Instructions ;`
`finTantque ;`
	`While (expression) {`
	`  Instructions ;`
	`}`

❖ **Fonctionnement**

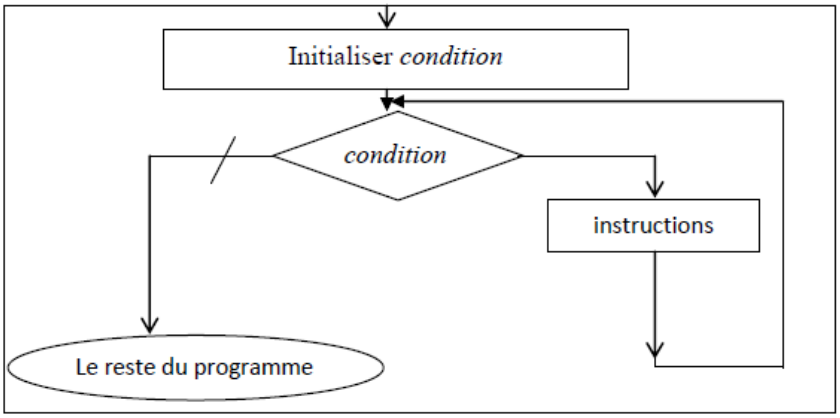
Le déroulement de la boucle **Tantque** provoque successivement et de manière répétitive :

- L'évaluation de la condition.
- L'exécution éventuelle (si la condition est satisfaite= Vrai) du corps de la boucle (le bloc d'instructions).
- Jusqu'à ce que la condition soit fausse.

❖ **Remarques importantes**

➢ La condition doit être modifiée par instructions, sinon on aura une boucle infinie.
➢ Le bloc d'instructions qui forme le corps de la boucle **Tantque** peut ne jamais être exécuté dans le cas où la condition **est fausse** dès le départ.
➢ La boucle est très utile lorsque le nombre d'itération n'est pas connu à l'avance.

❖ **Organigramme correspondant**



Exemple : l'algorithme suivant permet d'afficher **les nombres entiers** compris entre la valeur **1** et la valeur **d'une variable n** lue au clavier avec la boucle **Tantque.**

```
Algorithme Affiche_Nombres ;
Var N : entier ;
Debut
  Lire (N) ;
  Tantque (i<= n) faire
     Ecrire(i); i←i+1 ;
  Fintantque;
Fin.
```

3.3. La boucle Répéter

La boucle Répéter permet de **répéter l'exécution** d'une instruction ou bloc d'instructions jusqu'à ce qu'une **condition soit vérifiée** (= Vrai). La condition est formulée par une expression booléenne.

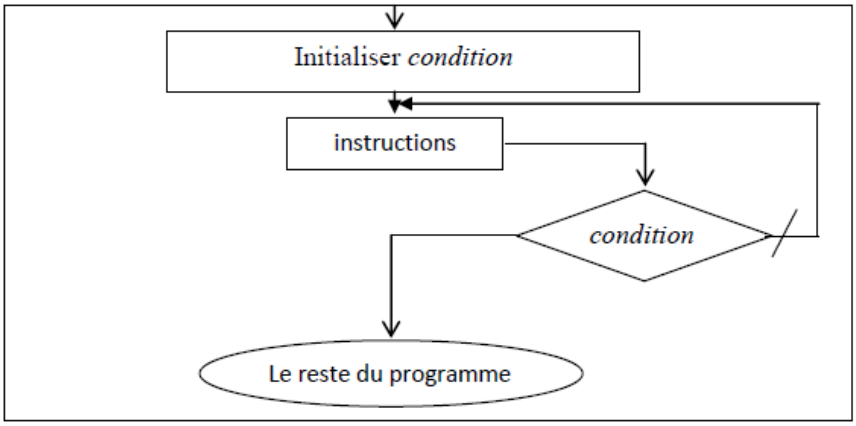
❖ **La syntaxe**

Algorithmique	Langage C
repeter Instructions ; Jusqua(expression);	do{ Instructions ; } While (expression);

❖ **Fonctionnement**

Contrairement à la boucle **tantque**, l'expression logique de la boucle **repeter** se trouve à la fin de la boucle. De ce fait, les instructions de la boucle sont exécutées au moins une fois. L'algorithme continu à exécuter les instructions de la boucle lorsque la valeur de (expression) **est fausse**. L'exécution de la boucle s'arrête dès que (expression) devient **vrai.**

❖ **Organigramme correspondant**



Exemple : l'algorithme suivant permet d'afficher **les nombres entiers** compris entre la valeur **1** et la valeur **d'une variable n** lue au clavier avec la boucle **Repeter**.

```
Algorithme Affiche_Nombres ;
Var N : entier ;
 Debut
   Lire (N) ;
   Repeter
      Ecrire(i); i←i+1 ;
   Jusqu'à (i> n);
 Fin.
```