

Chapitre 1 : Notions d’algorithmme et de programme

1. CONCEPT D’UN ALGORITHMME

Le mot *algorithme* vient du nom du célèbre *mathématicien arabe Al Khawarizmi (ABU JA'FAR MOHAMMED BEN MUSSA AL-KHWARISMI)* origine de l’ancienne ville de KHAWARISM aujourd’hui « KHIVA » située en ex-U.R.S.S. « algorithmique » a été conçu en 1958 par des chercheurs universitaires.

1.1 DEFINITION D’UN ALGORITHMME

Un algorithme est une suite d’instructions ayant pour but de résoudre un problème donné.

1.2 CRITERES D’UN BON ALGORITHMME

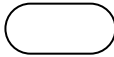
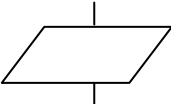
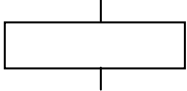
La *bonne connaissance* de l’algorithmique permet d’écrire des algorithmes *exacts et efficaces*, or c’est en écrivant des algorithmes *corrects*, qu’on devient un bon *Programmeur*.

Un algorithme correct doit donc subir à certains critères qui sont :

- être fini (achevé après un nombre fini d’actions élémentaires), être précis (la machine n’a pas à choisir)
- , être effectif (On pourrait le traiter “ à la main ”) et mentionner les entrées (saisie de données) et les sorties (affichage des résultats)

2. REPRÉSENTATION EN ORGANIGRAMME

Un organigramme est une représentation graphique d’un algorithme, il permet de schématiser graphiquement la solution d’un problème. Un organigramme permet de *mieux visualiser* la démarche de *résolution* d’un problème, il est construit à partir d’un formalisme comprenant *cinq simples* symboles normalisés qui sont reliés entre eux par des lignes de liaisons., ces symboles sont :

<i>SYMBOLE</i>	<i>DESIGNATION</i>
L’ovale : 	Il exprime le début ou la fin de l’organigramme.
Le parallélogramme : 	Il est utilisé pour les opérations d’entrée / sortie (Instruction ou groupe d’instructions pour lire les données ou écrire les résultats)
Le rectangle : 	Il est utilisé pour les opérations ou groupe d’opérations de traitement (calcul) sur des données.

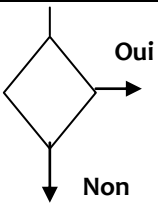

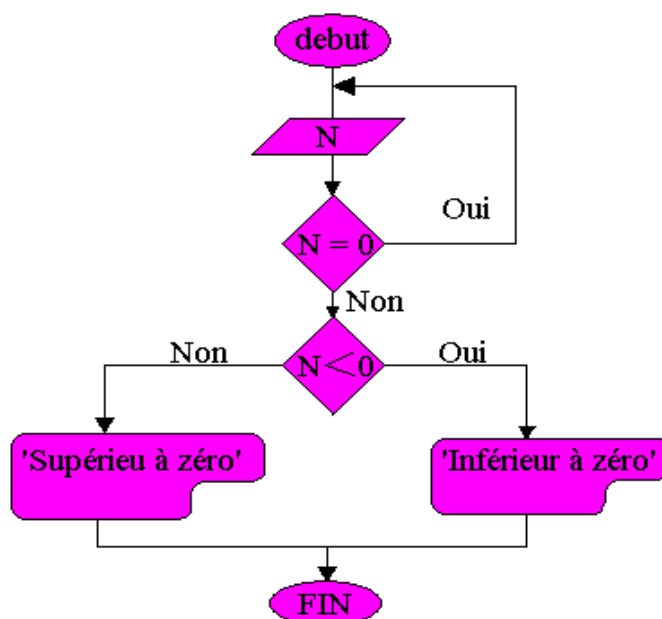
Le losange : 	Il est utilisé pour la vérification d'une condition (un test). Instruction conditionnelle. la condition est évaluée pour pouvoir prendre le chemin correspondant
Cercle de conjonction : 	Il est utilisé pour la liaison de plusieurs critères

Figure 1 : les différents symboles utilisés en organigramme

Exemple : Ecrire un organigramme qui lit un nombre N non nul et affiche le message: inférieure à "0" ou supérieur à "0" suivant sa valeur.

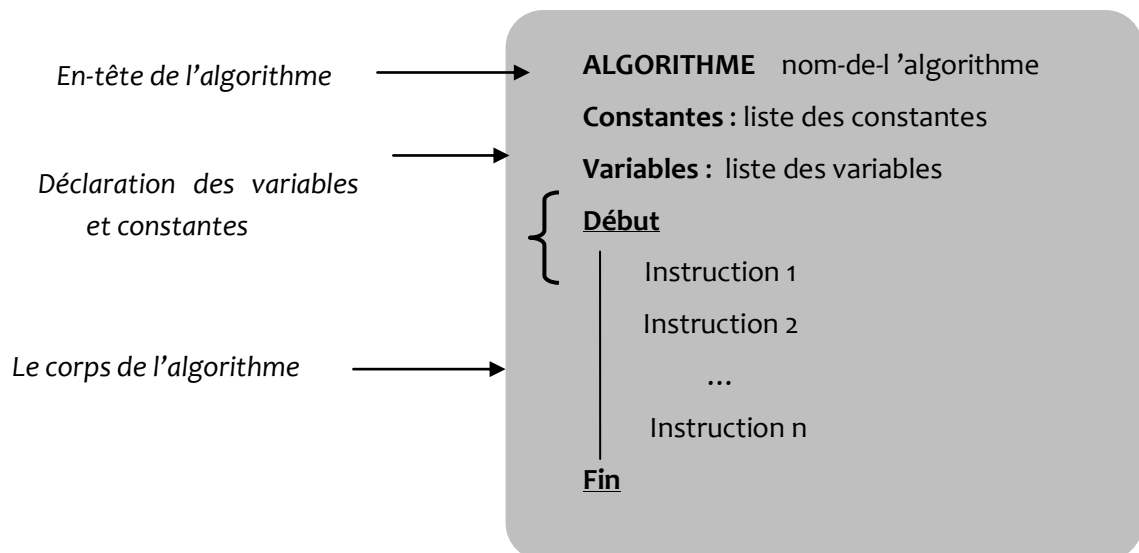


3. STRUCTURE D'UN PROGRAMME

3.1 A QUOI RESSEMBLE UN PROGRAMME INFORMATIQUE?

L'allure d'un programme dépend du type de langage utilisé pour faire le programmé. Le programme est constitué d'une suite d'instructions que la machine doit exécuter. Celle-ci exécute les instructions au fur et à mesure qu'elle lit le fichier (donc de haut en bas) jusqu'à ce qu'elle rencontre une instruction (appelée parfois instruction de branchement) qui lui indique d'aller un endroit précis du programme jusqu'à ce qu'elle arrive à la fin du programme et celui-ci s'arrête.

3.2 STRUCTURE GENERALE D'UN ALGORITHME



L'en-tête : permet tout simplement d'identifier l'algorithme.

Déclaration : liste de toutes les constantes et variables utilisées dans l'algorithme.

Le corps : cette partie contient les ordres (instructions) ou les tâches de l'algorithme.

4. LA DÉMARCHE ET ANALYSE D'UN PROBLÈME

Un algorithme s'écrit le plus souvent en pseudo-langage de programmation afin de faciliter ultérieurement sa traduction dans un langage de programmation.

Un programme se construit selon plusieurs étapes, la première consiste en l'analyse du problème posé. la seconde est l'établissement d'un algorithme et la troisième étape est la traduction de l'algorithme en programme, en utilisant un langage choisi. En fait L'algorithme est la résolution brute d'un problème informatique, il est indépendant du langage de programmation, Par exemple, on utilisera le même algorithme pour une implantation en Java, ou bien en C++ ou autre langage...

La rédaction d'un algorithme est un exercice de réflexion qui se fait sur papier. L'élaboration d'un algorithme précède donc l'étape de la programmation, C'est une démarche de résolution de problème exigeante,

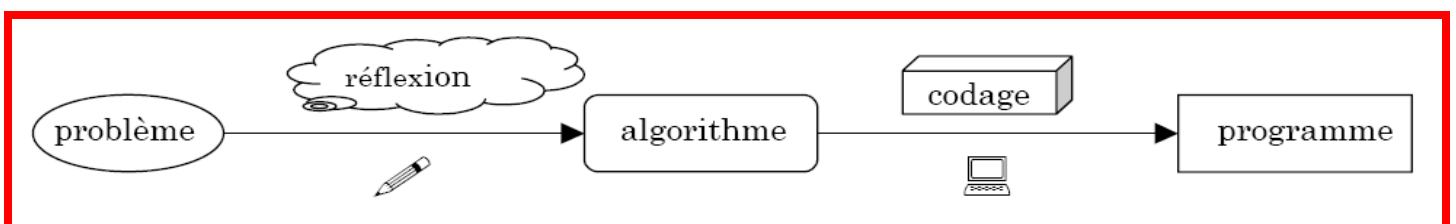


Figure 2 : Démarche et analyse d'un problème

Il faut remarquer que pour trouver une solution informatique à un problème, on doit passer par deux étapes :

- 1- phase de réflexion dite aussi analyse qui permet de lister les opérations à effectuer dans l'ordre, sur un ensemble de données d'entrées, pour arriver à un résultat. Cette phase sera soldée par un algorithme plus ou moins détaillé.

2- Phase de codage ou traduction de l'algorithme en un programme dans un langage de programmation bien choisi. Le résultat de cette opération pourra être exécuté sur un ordinateur.

Remarque :

- Un algorithme n'est donc exécutable directement par aucune machine. Mais il a l'avantage d'être traduit facilement dans tous les langages de programmation.
- Un algorithme exprime la structure logique d'un programme informatique et de ce fait est indépendant du langage de programmation utilisé. Par contre, la traduction de l'algorithme dans un langage particulier dépend du langage choisi.

5. STRUCTURE DES DONNEES

En informatique, une structure de données est une manière d'organiser les données pour les traiter plus facilement. Différentes structures de données existent pour des données différentes : constantes, variables, enregistrements, structures composées finies , tableaux (sur [1..n]), listes, arbres, graphes

5.1 CONSTANTES ET VARIABLES

5.1.1 DÉCLARATION DE CONSTANTES ET VALEUR

Les constantes sont des entités **permettant de réserver de l'espace** mémoire pour stocker des données dont leur valeur ne peuvent pas être modifiées au cours de l'exécution de l'algorithme (càd fixée pour tout l'algorithme), elles peuvent être de différentes natures : entière, réelle, booléenne, caractère, ...etc.

Comment déclarer les constantes ?

Constantes :
Ident = Valeur

Ident : c'est l'identificateur, c'est-à-dire le nom de la constante, il est composé de lettres et de chiffres.

Valeur : c'est la valeur de la constante, cette valeur restera inchangée pendant l'exécution de l'algorithme.

Exemple :

Constantes :

D= 55

QTE1= 45 .05

ElemTRV= Vrai

Ch= 'K'

5 .1.2 DÉCLARATION DE VARIABLES ET TYPE

Une variable est une place mémoire où est stockée une donnée sous forme d'octets. L'identificateur de cette variable permet d'avoir accès à ces données sans être obligé de travailler sur les octets. Les variables peuvent être

de différents types, par exemple : entier (INTEGER), réel (REAL), Booléen (BOOLEAN), caractère (CHAR), chaîne de caractères (STRING) ...

Comment déclarer les variables ?

Variables :
Ident : Type

Déclarer une variable, c'est réserver une certaine place mémoire adaptée au type de la variable et lui associer un identificateur.

Ident : c'est l'identificateur, c'est-à-dire le nom de la variable, il est composé de lettres et de chiffres.

Type : Un type (de variable) détermine l'ensemble de valeurs possibles de la variable déclarées pour designer la nature du contenu de la variable et les opérations pouvant être effectuées sur celle-ci. Lorsqu'une variable est déclarée (association d'un identifiant et d'un type) la place mémoire correspondant au type est associée à l'identifiant de la variable. Donc le type déterminera la nature de la variable.

Les types sont de deux sortes :

I. **LES TYPES STANDARDS** : ils sont déjà définis dans le compilateur du langage, contient :

- **Type entier** : représentant un nombre entier quelconque exemple : (1, 5, -9000, 1256, 98, -45)
- **Type réel** : représentant un nombre réel quelconque exemple : (1.5, 5.0, -90.125, 1.256, 9.8, -45.0)
- **Caractère** : représentant un caractère seul exemple : ('a', 'b', ' ', '7', '/', '^', 'R', ':')
- **Chaîne de caractères ou String** : représentant un texte de zéro, un ou plusieurs caractères. Le nombre maximal de caractères pouvant être stockés dans une seule variable string dépend du langage utilisé. Un caractère sera noté avec une apostrophe simple (exemple 'c') et le string sera notée entre guillemets doubles (exemple "contenu de la chaine").
- **Type booléen** : représentant une valeur logique binaire oui ou non, ouvert ou fermé, vrai ou faux. On peut représenter ces notions abstraites de VRAI et de FAUX par tout ce qu'on veut : de l'anglais (TRUE et FALSE) ou des nombres (0 et 1). Peu importe. Ce type booléen est très économique en termes de place mémoire occupée, puisque pour stocker une telle information binaire, un seul bit suffit.

Exemple de type standard :

Variables :

A, B : entier

Val1, Val2 : réel

Test : booléen

Remarque :

- Ne pas faire la confusion entre une variable Chaîne contenant des nombres (par exemple « 185 ») et une variable Entier ! Sur une variable de type Chaîne, il n'est pas possible de faire des calculs, il faut alors considérer le nombre comme un mot composé des caractères '1', '8', et '5'. Pour éviter de confondre un nombre d'une chaîne, il faut *TOUJOURS* noter une chaîne entre guillemets.
- Il est obligatoire de préciser dès le départ ce que la variable contiendra. Cela permet à l'ordinateur de prévoir la taille de la variable : un entier aura besoin d'une petite place, tandis qu'une chaîne de caractères aura besoin d'une grande place.

II. **Les types non standards** : ils ne sont pas définis. C'est à l'utilisateur de le définir.

Exemple : tableau de tableaux, tableau d'enregistrement....

6. LES OPÉRATEURS

Les opérateurs dépendent du type de l'opération, ils peuvent être :

A.DES OPERATEURS ARITHMETIQUES: Ce sont les quatre opérations arithmétiques :

- + : Pour l'addition
- - : Pour la soustraction
- * : Pour la multiplication
- DIV : pour la division entière, exemple $5 \text{ DIV } 2 = 2$;
- MOD : Pour le modulo (le reste de la division entière), exemple $5 \text{ MOD } 2 = 1$.

B.DES OPERATEURS RELATIONNELS qui sont :

- < inférieur à
- > Supérieur à
- <= inférieur ou égale
- >= Supérieur ou égale
- <> Différent de

C.DES OPERATEURS LOGIQUES:

- AND le et logique
- OR le ou logique
- XOR le ou exclusif
- NOT le non logique

D.DES OPERATEURS SUR LES CHAINES: & (concaténation) ou Opérateur alphanumérique (&) :

Cet opérateur permet de concaténer, autrement dit de coller l'une à l'autre, deux chaînes de caractères.

Remarques

- ❖ On ne peut pas additionner un entier et un caractère.
- ❖ La signification d'un opérateur peut changer en fonction du type des opérandes, par exemple :

L'opérateur + avec des entiers effectue l'addition, 3+6 vaut 9 • avec des chaînes de caractères il effectue la concaténation "bonjour" + " tout le monde" vaut "bonjour tout le monde".

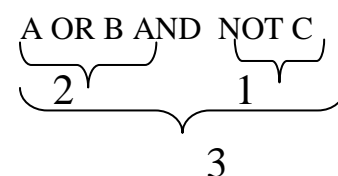
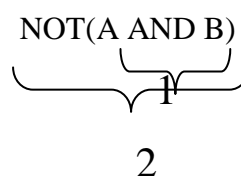
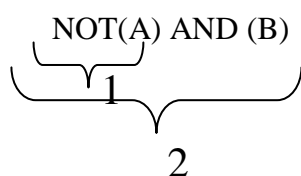
7.1 LES PRIORITÉS DANS LES OPÉRATIONS :

Une expression est évaluée de *gauche à droite* mais en tenant compte des priorités des opérateurs.

• Pour l'évaluation des expressions logiques on a l'ordre suivant :

- 0° : () = parenthèse et fonctions prédéfinis.
- 1° : NOT
- 2° : AND
- 3° : OR

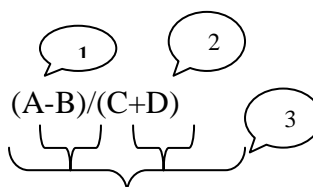
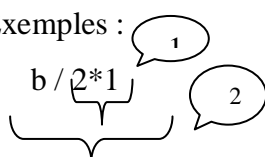
Exemples:



• Pour l'évaluation des expressions arithmétiques :

- 0° : () les parenthèses et les fonctions prédéfinis.
- 1° : * , / , DIV , MOD
- 2° : - , +
- 3° : affectation

Exemples :



Remarque : On utilise les parenthèses, avec les mêmes règles qu'en **mathématiques**. La multiplication et la division ont priorité sur l'addition et la soustraction. Les parenthèses sont utiles pour modifier cette priorité naturelle. Exemple $6 : 12 * 3 + 5$ et $12 * (3 + 5)$ n'ont pas le même résultat.

7. LES OPÉRATIONS D'ENTREE/SORTIE

Un algorithme peut avoir des interactions avec l'utilisateur, Il peut afficher un résultat comme il peut demander à l'utilisateur de saisir une information afin de la stocker dans une variable.

7.1 INSTRUCTION DE LECTURE (ENTRÉE)

L'instruction de prise de données sur le périphérique d'entrée (en général le clavier)

Structure générale :

Lire (variable)
Lire (variable1, variable2, ...)

Exemple :

Lire(s) ; Lire (a, b, c) ; Lire (X, Y)

7.2 INSTRUCTION D'ÉCRITURE (SORTIE)

L'instruction de restitution de résultats sur le périphérique de sortie (en général l'écran)

Structure générale :

Écrire (variable)
Écrire ('message')
Écrire ('message', variable)

Exemple :

Écrire (s) ; Écrire (' entrer les valeurs') ; Écrire ('la somme=', som)

7.3 L'INSTRUCTION D'AFFECTATION

L'instruction d'affectation, comme son nom l'indique, permet d'affecter une valeur à une variable.

Structure générale :

Ident ← expression

L'expression est une suite d'opérations sur des constantes ou variables déjà déclarées.

Remarque : les variables et constantes utilisées dans l'expression plus l'identificateur doivent avoir le même type.

Exemple 1 : Soient *A* et *B* deux variables de type entier.

- ① $A \leftarrow 3$ (* *A* vaut 3, *B* n'a pas encore de valeur *)
- ② $B \leftarrow A+4$ (* *B* vaut 7, *A* vaut toujours 3 *)
- ③ $A \leftarrow B*2$ (* *A* vaut 14, *B* vaut 7 *)
- ④ $B \leftarrow B+1$ (* *A* vaut 14, *B* vaut 8 *)

- **Montrer le tracé d'exécution**

Etape	A	B	Ecran
1	3		/
2	3	7	/
3	14	7	/
4	14	8	/

Exemple 2 :

Comment échanger les valeurs de deux variables *A* et *B* ?

Algorithme permutation

Variables :

A, B, T : entier

Début

- ① Lire (A, B)
- ② $T \leftarrow A$
- ③ $A \leftarrow B$
- ④ $B \leftarrow T$
- ⑤ Ecrire (A, B)

Fin.

- **Montrer le tracé d'exécution pour les valeurs ($A=3, B=1$)**

Etape	A	B	T	Ecran
1	3	1	/	/
2	3	1	3	/
3	1	1	3	/
4	1	3	3	/
5	1	3	3	1 - 3 - 3

SEANCE 2

Chapitre 1 : Notions d'algorithme et d Programme(SUITE)

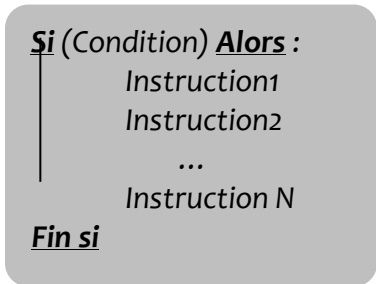
8. LES STRUCTURES DE CONTRÔLE

8.1 LES STRUCTURES DE CONTRÔLE CONDITIONNELLE

Les instructions conditionnelles choisissent ou annulent l'exécution d'une suite d'ordres selon une condition bien définie. On en distingue trois types :

8.1.1 L'INSTRUCTION CONDITIONNELLE SIMPLE

Ce type d'instructions incorpore un bloc d'instructions dont son exécution dépend de la condition qui lui a été associé. Cette structure a la forme suivante :



Exemple :

① Lire (A, B)

Si (A > B) **alors** :

② { A ← A+2
 B ← 3

Fin si

③ Ecrire (A, B)

- Montrer le tracé d'exécution pour les valeurs :

a. (A= 3, B =2)

b. (A=2, B =2)

A :

Etape	A	B	Ecran
1	3	2	/
2	5	3	/
3	5	3	5 - 3

B :

Etape	A	B	Ecran
1	2	2	/
3	2	2	2 - 2

8.1.2 L'INSTRUCTION CONDITIONNELLE ALTERNATIVE

Ce type d'instructions incorpore deux blocs d'instructions dont un et un seul bloc sera exécuté, de manière alternative, en fonction de la condition. Cette structure a la forme suivante :

```

Si (Condition) Alors :
    Instruction1
    ...
    Instruction N
Sinon
    Instruction1
    ...
    Instruction M
Fin si
    
```

Exemple :

① Lire (A, B)

Si (B ≠ 0) **Alors** :

② { A ← 3
 C ← B*4
Sinon

C ← A+1 ③

Fin si

écrire (A, B, C) ④

- Montrer le tracé d'exécution pour les valeurs :

a. (A= 5, B =2) et (A=4, B =0)

A :

Etape	A	B	C	Ecran
1	5	2	/	/
2	3	2	8	/
4	3	2	8	3 - 2 - 8

B :

Etape	A	B	C	Ecran
1	4	0	/	/
3	4	0	1	/
4	4	0	1	3 - 0 - 1

SEANCE 2

8.1.3 L'INSTRUCTION CONDITIONNELLE DE CHOIX

L'instruction conditionnelle de choix comporte plusieurs blocs d'instructions dont un et un seul bloc sera exécuté selon la valeur de la variable employée comme condition. Cette instruction a la structure suivante :

Selon le cas (Variable) :

Variable = Valeur₁ : Instruction (s)

Variable = Valeur₂ : Instruction (s)

...

Variable = Valeur_N : Instruction (s)

Sinon

Instruction(s)

Fin selon

Exemple : Ecrire un algorithme qui lit une valeur puis il affiche le jour qui correspond à cette valeur.

Algorithme jour

Variables :

N : entier

Début

Lire (n)

Selon le cas (n) :

n = 1 : écrire ('samedi')

n = 2 : écrire ('dimanche')

n = 3 : écrire ('lundi')

n = 4 : écrire ('mardi')

n = 5 : écrire ('mercredi')

n = 6 : écrire ('jeudi')

n = 7 : écrire ('vendredi')

Sinon écrire ('le nombre doit être compris entre 1 et 7')

Fin selon

Fin.

SEANCE 3

Chapitre 1 : Notions d'algorithme et d Programme(SUITE)

8.2 LES STRUCTURES DE CONTRÔLES RÉPÉTITIVES

Les instructions répétitives, appelées aussi les instructions itératives ou encore les boucles, permettent de répéter plusieurs fois l'exécution d'une même suite d'instructions. On en distingue trois types :

8.2.1 LA BOUCLE « POUR »

La boucle « Pour » permet de répéter l'exécution d'une suite instructions un nombre de fois connu d'avance. Pour cela, il faut préciser la valeur initiale et la valeur finale et éventuellement le pas (lorsqu'il est différent de 1). . Cette boucle a la structure suivante :

```
Pour i allant de VI à VF (pas= valeur) faire :  
|  
|   Instruction1  
|   Instruction2  
|   ...  
|   Instruction N  
Fin pour
```

VI : valeur initiale.

VF : valeur finale.

Exemple 1 : Comment afficher le message «Université Mentouri' » 20 fois ?

```
Pour i allant de 1 à 20 faire  
|   Ecrire ('Université Mentouri')  
Fin pour
```

Exemple2 : Ecrire un algorithme qui calcule la somme des N premiers nombres entiers positifs.

Algorithme sommeN1

Variables :

N, Som, i : entier

Début

```
|   Lire (N)  
|   Som ← 0  
|   Pour i allant de 1 à N faire  
|   |   Som ← Som+i  
|   Fin pour  
|   Ecrire(Som)
```

Fin

SEANCE 3

Remarque : l'incrémentation de « i » se fait de manière automatique.
Le pas, lorsqu'il n'est pas spécifié, est égal à 1.

8.2.2 La boucle « Tant que »

La boucle « Tant que » permet de répéter l'exécution d'une suite d'instructions tant qu'une certaine condition est remplie. Cette boucle a la structure suivante :

Tant que (Condition) **faire :**

Instruction1

Instruction2

...

Instruction N

Fin TQ

La suite d'instructions de la boucle « Tant que » sera ré-exécutée tant que la condition restera vraie.

Exemple1 : Comment afficher le message « Université Mentouri » 20 fois ?

$i \leftarrow 1$

Tant que ($i \leq 20$) **faire**

Ecrire ('Université Mentouri')

$i \leftarrow i+1$

Fin TQ

Exemple2 : Ecrire un algorithme qui calcule la somme des N premiers nombres entiers positifs.

Algorithme sommeN2

Variables :

N, Som, i : entier

Début

Lire (N)

Som $\leftarrow 0$

$i \leftarrow 1$

Tant que ($i \leq N$) **faire**

Som \leftarrow Som+i

$i \leftarrow i+1$

Fin TQ

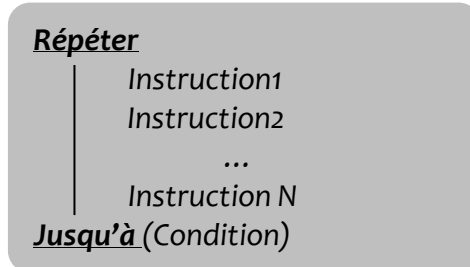
Ecrire(Som)

Fin

SEANCE 3

9.2.1 LA BOUCLE « RÉPÉTER »

La boucle « Répéter » permet de répéter l'exécution d'une suite d'instructions tant qu'une certaine condition n'est pas remplie, c'est-à-dire, on sort de la boucle « Répéter » quand la condition sera satisfaite. Cette boucle a la structure suivante :



Exemple1

Comment afficher le message « Université Constantine1' » 20 fois ?

$i \leftarrow 1$

Répéter

| Ecrire ('Université Constantine1')

| $i \leftarrow i+1$

Jusqu'à ($i > 20$)

Exemple2 : Ecrire un algorithme qui calcule la somme des N premiers nombres entiers positifs.

Algorithme sommeN3

Variables :

N, Som, i : entier

Début

| Lire (N)

| $Som \leftarrow 0$

| $i \leftarrow 1$

Répéter

| $Som \leftarrow Som+i$

| $i \leftarrow i+1$

Jusqu'à ($i > 20$)

| Ecrire(Som)

Fin

SEANCE 4

9. EXERCICES

9.1 LES STRUCTURES CONDITIONNELLES

Exercice1 : écrire un algorithme qui calcule la valeur absolue d'un nombre réel.

Exercice2 : écrire un algorithme qui détermine si un nombre lu est pair ou impair.

Exercice3 : écrire un algorithme qui résout une équation de 1^{er} degré.

- Construire l'organigramme correspondant à cet algorithme.

Exercice4 : écrire un algorithme qui résout une équation de 2^{ème} degré.

9.2 SOLUTIONS

Exercice1 :

Algorithme ValAbs_1

Variables :

X : réel

Début

Lire (X)

Si $X \geq 0$ alors

Ecrire(X)

Sinon

Ecrire (-X)

Fin si

Fin

Algorithme ValAbs_2

Variables :

X : réel

Début

Lire (X)

Si $X < 0$ alors

$X \leftarrow -X$

Fin si

Ecrire (X)

Fin

Exercice2 :

Algorithme pairImpair_1

Variables :

N, Q, R : entier

Début

① Lire (N)

② $Q \leftarrow N \text{ div } 2$

③ $R \leftarrow N - Q * 2$

Si $(R = 0)$ alors

④ Ecrire (N, 'est pair')

Sinon

Ecrire (N, 'est impair') ⑤

Fin si

Fin

Algorithme pairImpair_2

Variables :

N : entier

Début

Lire (N)

Si $(N \bmod 2 = 0)$ alors

Ecrire (N, 'est pair')

Sinon

Ecrire (N, 'est impair')

Fin si

Fin

- Montrer le tracé d'exécution pour les valeurs :

- a. $N=5$ ET b. $N=8$

SEANCE 4

A :

Etape	N	Q	R	Ecran
1	5	/	/	/
2	5	2	/	/
3	5	2	1	/
5	5	2	1	5 est impair

B :

Etape	N	Q	R	Ecran
1	8	/	/	/
2	8	4	/	/
3	8	4	0	/
4	8	4	0	8 est pair

Exercice3 :

Algorithme equaD1

Variables :

A, B, X: réel

Début

Lire(A)

Lire(B)

Si A=0 **alors**

Si B=0 **alors**

 Ecrire ('infinité de solutions')

Sinon

 Ecrire ('Pas de solution')

Fin si

Sinon

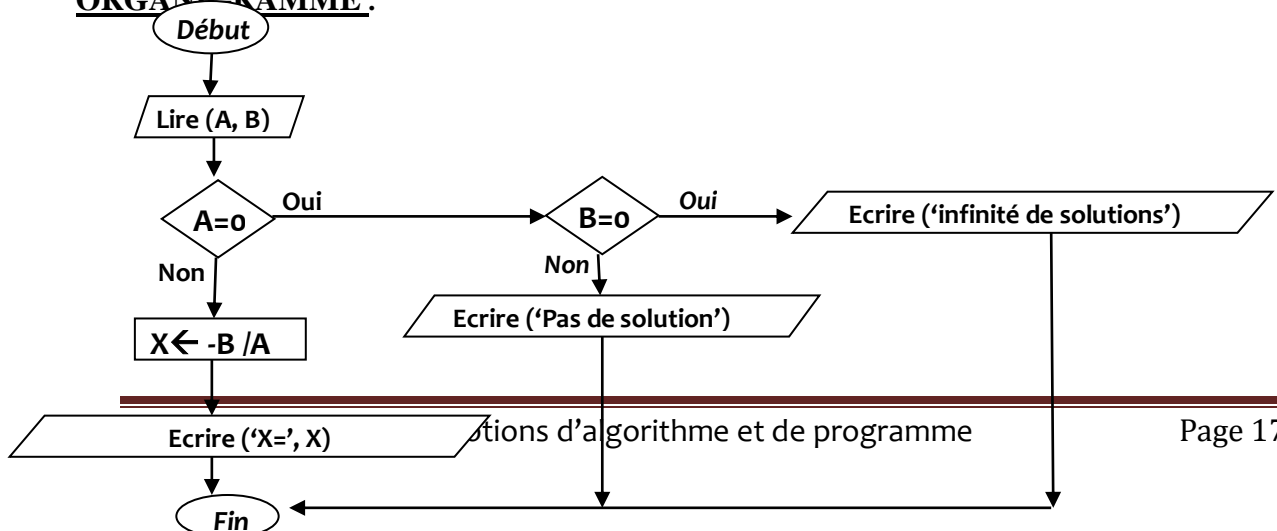
$X \leftarrow -B/A$

 Ecrire ('X=', X)

Fin si

Fin

ORGANIGRAMME :



SEANCE 4

Exercice4 :

Algorithme equaD2

Variables :

A, B, C, X, X1, X2, Delta: réel

Début

Lire(A)

Lire(B)

Lire(C)

Si A=0 alors

Si B=0 alors

Si C=0 alors

Ecrire ('infinité de solutions')

Sinon

Ecrire ('Pas de solution')

Fin si

Sinon

X ← -C/B

Ecrire ('X=',X)

Fin si

Sinon

Delta ← B*B-4*A*C

Si Delta < 0 alors

Ecrire ('Pas de solution')

Sinon

Si Delta=0 alors

X ← -B/(2*A)

Ecrire ('solution double=',X)

Sinon

X1 ← -(B-SQRT(Delta))/(2*A)

X2 ← -(B+SQRT(Delta))/(2*A)

Ecrire ('deux sol dif=',X1, X2)

Fin si (* fin si de delta=0*)

FIN SI (* fin si de delta <0*)

Fin si (* fin si de A=0*)

Fin (* fin de l'algorithme*)

9.3 LES STRUCTURES RÉPÉTITIVES :

Exercice1 : écrire un algorithme qui affiche les nombres impairs divisibles par 3 et inférieurs à « N »

SEANCE 4

Algorithme Impair3_A

Variables :

N, i : entier

Début

Lire (N)

Pour i allant de 1 à N faire

Si (N mod 2 ≠ 0) et (N mod 3=0) alors

Écrire (i)

Fin si

Fin pour

Fin

Algorithme Impair3_B

Variables :

N, i : entier

Début

Lire (N)

Pour i allant de 1 à N (pas=2) faire

Si (N mod 2 ≠ 0) alors

Écrire (i)

Fin si

Fin pour

Fin

Algorithme Impair3_C

Variables :

N, i : entier

Début

Lire (N)

Pour i allant de 1 à N (pas =6) faire

Écrire (i)

Fin pour

Fin

Exercice 2 : écrire un algorithme qui calcule le factoriel d'un nombre lu
« N »

Algorithme factoriel

Variables :

N, Fact, i : entier

Début

Lire (N)

Fact ← 1

Pour i allant de 1 à N faire

Fact ← Fact*i

Fin pour

Écrire(Fact)

Fin

SEANCE 4

Exercice 3 : écrire un algorithme qui affiche tous les diviseurs d'un nombre lu « N »

Algorithme Diviseurs

Variables :

N, i : entier

Début

Lire (N)

Pour i allant de 1 à N faire

Si (N mod i=0) alors

Écrire (i)

Fin si

Fin pour

Fin

Exercice 4 : écrire un algorithme qui calcule le produit de deux nombres entiers en utilisant seulement la somme.

Algorithme produitAB

Variables :

A, B, P, i : entier

Début

Ecrire ('Entrer deux nombres
Entiers positifs',) ; Lire (A, B)

$P \leftarrow 0$

Pour i allant de 1 à B faire

$P \leftarrow P + A$

Fin pour

Écrire(P)

Fin

Exercice 5 : écrire un algorithme qui affiche la moyenne de 20 nombres entiers.

SEANCE 4

Algorithme Moyenne

Variables :

N, Som, i : entier

Moy : réel

Début

Som \leftarrow 0

Pour i allant de 1 à 20 faire

 Lire (N)

 Som \leftarrow Som+N

Fin pour

Moy \leftarrow Som/20

Ecrire(Moy)

Fin

Exercice 6 : écrire un algorithme qui affiche les nombres impairs inférieurs à N.

Algorithme Affichage-A

Variables :

N, i : entier

Début

i \leftarrow 1 | Lire (N)

Tant que (i \leq N) faire

 Ecrire (i)

 i \leftarrow i+2

Fin TQ

Fin

Algorithme Affichage-B

Variables :

N, i : entier

Début

 Lire (N)

Pour i allant de 1 à N (pas =2) faire

 Ecrire (i)

Fin pour

Fin