Chapitre 5 : Algorithmes et Programmes

1. CONCEPT D'UN ALGORITHME

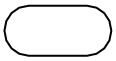
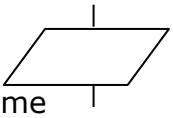
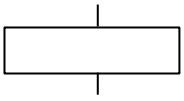
Le mot *algorithme* vient du nom du célèbre mathématicien arabe *Al Khawarizmi* (**ABU JA'FAR MOHAMMED BEN MUSSA AL-KHWARISMI**) origine de l'ancienne ville de **KHAWARISM** (aujourd'hui **KHIVA**).

1.1 DEFINITION D'UN ALGORITHME

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

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 :

SYMBOL	DESIGNATION
Lovale 	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

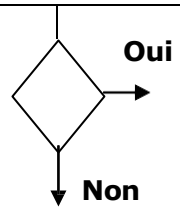
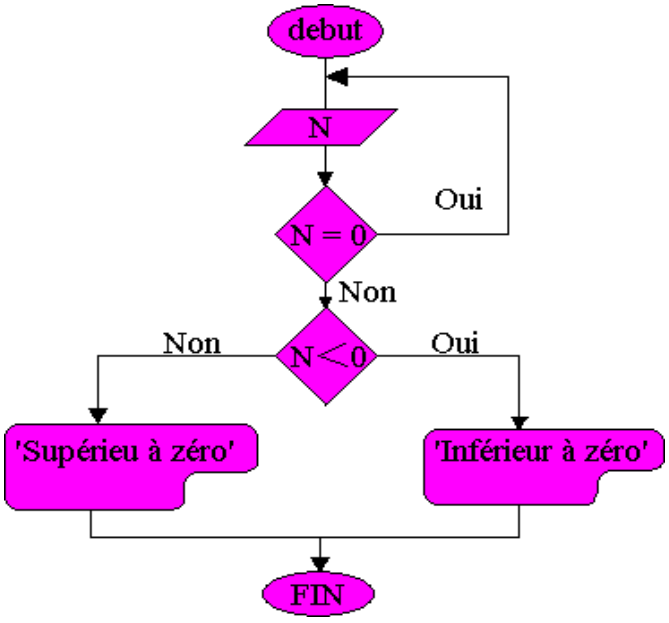
Le losange : 	Il est utilisé pour la vérification d'une condition (un test). Instruction conditionnelle. la condition est évaluée pour pouvoir prendre le chemin correspondant
Cercle de conjonction : 	Il est utilisé pour la liaison de plusieurs critères

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 à 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 →

La déclaration des constantes *et des variables* →

Le corps →

```
ALGORITHME  nom de l'algorithme
Constantes : liste des constantes
Variables : liste des variables
Début
      Instruction 1
      Instruction 2
      …
      Instruction n
Fin
```

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

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

Le corps : contient les instructions de l'algorithme.

4. 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 .. etc

4.1 CONSTANTES ET VARIABLES

4.1.1 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, elles peuvent être de différentes natures : entier, réel, booléen, caractère, …etc.

```
Constantes :
   id = valeur
```

id : 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 :
      n = 10
      note = 19.5
      présent = vrai
      c = 'A'
      prénom = "Mohammed"»
```

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) ...

```
Variables :
      id : type
```

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

id : 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 :

A. **Les types standards** : ils sont déjà définis dans le compilateur du langage, contient :

- **Entier** : représentant un nombre entier quelconque exemple : (1, 5, -9000, 1256, 98, -45)
- **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** : 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 la chaîne de caractères sera notée entre guillemets doubles (exemple **"contenu de la chaine"**).

- **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 :

```
Variables :
    A, B : entier
    X, Y : réel
    Pair : booléen
```

B. Les types non standards : C'est à l'utilisateur de le définir. Exemple : tableau de tableaux, tableau d'enregistrement….

5. LES OPÉRATEURS

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

A. ARITHMETIQUES :

+: Addition

- : Soustraction

***** : Multiplication

/ : Division entière

B. RELATIONNELS :

< : inférieur à

> : Supérieur à

<= : inférieur ou égale

>= : Supérieur ou égale

<> : Différent de

C. LOGIQUES :

AND : le et logique

OR : le ou logique

XOR : le ou exclusif

NOT : le non logique

D. **SUR LES CHAINES DE CARACTÈRES** : & (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".

6. 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.

6.1 INSTRUCTION DE LECTURE (ENTRÉE)

L'instruction de prise de données sur le périphérique d'entrée.

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.

Structure générale :

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

Exemple :

```
Écrire ('entrer les notes');
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 <- 3 (* A vaut 3 , B n'a pas encore de valeur *)*

2. *B <- A+4 (* B vaut 7 , A vaut toujours 3 *)*

3. *A <- B*2 (* A vaut 14 , B vaut 7 *)*

4. *B <- 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
       Lire (A, B)
       T <- A
       A <- B
       B <- T
       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*

7. LES ST RUCTURES DE CONTRÔLE

7.1 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 :

7.1.1 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 :

```
Si (Condition) Alors :
     Instruction1
     Instruction2
        …
     Instruction N
Fin si
```

Exemple :

```
Lire (A, B)
Si  (A > B) alors :
     A <- A+2
     B <- 3
Fin si
Ecrire (A, B)
```

7.1.2 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 :

```
Lire (A, B)
Si (B <> 0) Alors :
     A <- 3
     C <- B*4
  Sinon
     C <- A+1
Fin si
Écrire (A, B, C)
```

7.1.3 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 = ValeurN : 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('n doit être compris entre 1 et 7')
Fin selon
```

7.2 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 :

7.2.1 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 ('Bonjour')
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
```

7.2.2 La boucle « Tant que »

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

```
Tant que (Condition) faire :
    Instruction 1
    Instruction 2
     …
    Instruction N
Fin Tant que
```

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

Exemple1 : Comment afficher le message «Bonjour » 20 fois

```
i<-1
Tant que (i <=20) faire
    Ecrire ("Bonjour")
    i<- i+1
Fin Tant que
```

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 <- 0
  i <- 1
  Tant que  (i <=N) faire
       Som <- Som+i
       i <- i+1
  Fin Tant que
  Ecrire(Som)
Fin
```

9.2.1 LA BOUCLE « RÉPÉTER »

La boucle « Répéter» permet de répéter l'exécution d'une suite 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 :

```
Répéter
      Instruction 1
      Instruction 2
            …
      Instruction N
Jusqu'à (Condition)
```

Exemple 1 : Comment afficher le message «Bonjour» 20 fois ?

```
i<- 1
Répéter
   |    Ecrire ("Bonjour")
   |    i<- i+1
Jusqu'à (i > 20)
```

Exemple 2 : Ecrire un algorithme qui calcule la somme des N premiers nombres entiers positifs.

```
Algorithme sommeN3
Variables : N, Som, i : entier
Début
      Lire (N)
      Som <- 0
      i <- 1
      Répéter
            Som <- Som+i
            i <- i+1
      Jusqu'à (i>20)
      Ecrire(Som)
Fin
```