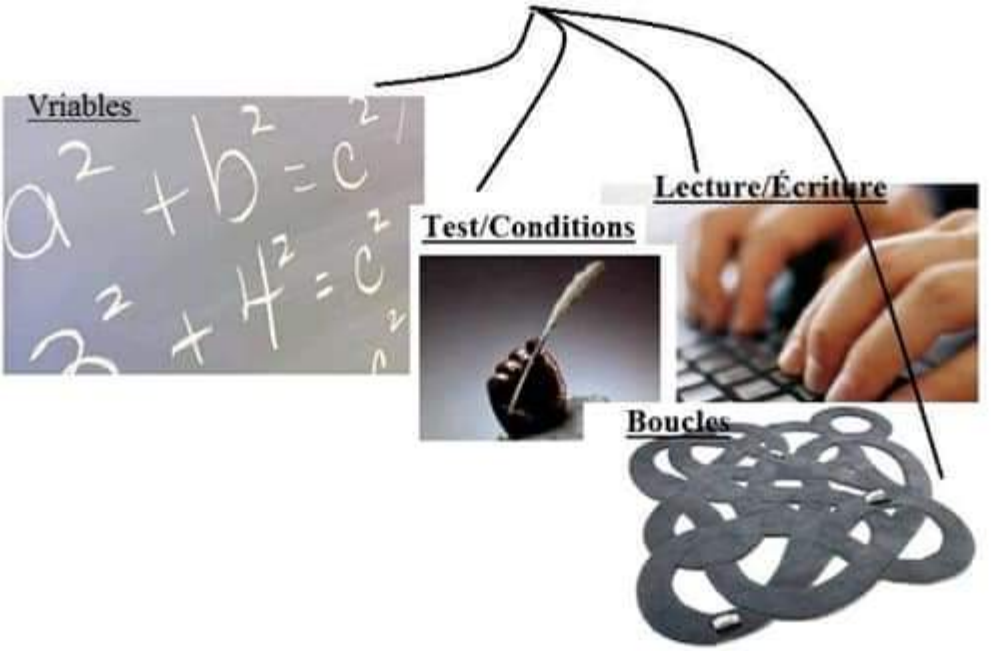
Chapitre III

Notions de base en algorithmique



Introduction

Tout problème doit être résolu d'abord sous forme d'algorithme, puis converti en programme. En effet, un algorithme est indépendant de tout langage de programmation. Il exprime les instructions résolvant un problème donné indépendamment des particularités de tel ou tel langage de programmation.

La programmation est une démarche qui se déroule en deux phases :

- Phase d'analyse du problème, c.à.d. la recherche d'algorithme ;
- Phase de programmation proprement dite, qui consiste à exprimer le résultat de la première phase dans un langage donné.

Donc, la conception d'un algorithme est une étape indispensable dans tout développement informatique. Ainsi que la réécriture d'un algorithme dans un langage de programmation est l'étape finale du développement.

1- Notion d'algorithme :

D'une façon générale, un algorithme est la description de la méthode de résolution d'un problème donnée en utilisant des instructions élémentaires. Ces instructions deviennent compréhensibles par un ordinateur après la traduction de l'algorithme en programme.

Dans la vie courante un algorithme peut prendre la forme d'une recette de cuisine, d'un itinéraire routier, d'un mode d'emploi ou une notice de montage.

Une recette de cuisine par exemple est un algorithme qui explique comment réaliser un plat à partir des ingrédients. De même un itinéraire routier explique comment rejoindre une position finale à partir d'une position initiale en un certain nombre d'étapes.

Exemple :

Pour préparer la pâte d'une tarte les actions élémentaires pour réaliser ce travail sont :

Début
Incorporer le beurre dans la farine
Pétrir le mélange jusqu'à ce qui.il soit homogène
Ajouter le lait
Mélanger
Le laisser reposer une demi-heure
Faire cuire la pâte
Fin

Donc, plus précisément, un algorithme sert à transmettre un savoir faire. Il décrit les étapes à suivre pour réaliser un travail. Tout comme le savoir faire du cuisinier se transmet sous la forme d'une recette, celui d'un informaticien se transmet sous la forme d'un algorithme.

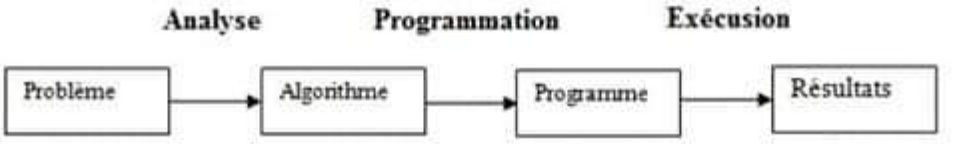
Un algorithme est appelé aussi **« pseudo-code ».**

2- Algorithme et programme :

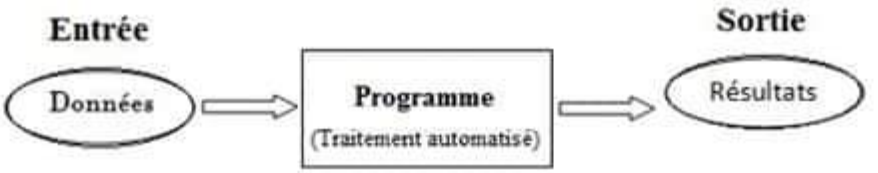
Un programme est un enchaînement d'instructions, écrit dans un langage de programmation, permettant de traiter un problème. Il représente la traduction d'un algorithme à l'aide d'un langage de programmation.

Exemple : parmi les langages de programmation on peut citer Pascal, C, C++, Visual Basic, JAVA, C#, etc.

Le cycle de développement d'un programme (ou d'une application) peut se résumer en deux phases :



Un traitement destiné à être réalisé par un ordinateur dit traitement automatisé. Ce traitement consiste à effectuer des opérations sur des informations que l'on peut qualifier de données d'entrée, après le traitement, d'autres informations appelées résultats (ou données de sortie) sont générées.



Généralement, lorsqu'on cherche un algorithme permettant d'automatiser un traitement donné, on doit se poser trois questions :

- Qu'est ce qu'on doit obtenir comme résultat ?
- Quelles sont les données dont on a besoin ?
- Comment faire (traitement proprement dit) ?

Exemple :

Considérons le traitement qui consiste à calculer la moyenne d'un groupe d'étudiants pour un module donné. Dans ce cas :

- Pour effectuer le calcul de la moyenne, on a besoin de la note de chaque étudiant dans le module en question ;
- Ensuite, on effectue le calcul comme suit :
  - ✓ On calcule la somme des notes,
  - ✓ On divise par le nombre d'étudiants,
- Enfin, on affiche le résultat obtenu (la moyenne).

3- Structure d'un algorithme

L'algorithme est composé de trois parties principales :

- L'entête : sert à donner un nom à l'algorithme. Elle est précédée par le mot Algorithme.
- La partie déclarative : dans cette partie on déclare les différents objets que l'algorithme nécessite (variable et constantes, etc.) .
- Le corps de l'algorithme : cette partie contient les instructions a réaliser pour traiter le problème. L'ensemble de ces instructions doit être délimitée par les deux mots clés Début et Fin.

Entête		*Algorithme* NomAlgorithme
Partie déclarative		*Constante* Identificateur = valeur *Variable* Identificateur : type
Corps de l'algorithme		Début Instruction 1 Instruction 2 ..... Instruction n Fin

3-1- Notions de base

Quatre notions de base en algorithmique et en programmation vont être étudiées, à savoir la notion de variables et de constantes, la notion d'affectation et les deux notions d'écriture et de lecture.

1- Les variables et les constantes

Avant d'entamer ces deux notions, il serait nécessaire de comprendre l'organisation de la mémoire dans un ordinateur. Physiquement, la mémoire vive de l'ordinateur est formée d'éléments dont chacun ne peut prendre que deux états distincts. Ce sont les bits d'information. Ces bits sont manipulés par groupes de 8 bits (octets) ou (mots de 16, 32, 64 bits).

La mémoire est donc formée de mots (cases mémoire) et pour que l'unité centrale puisse y placer une information et la retrouver, chaque mot est repéré par une adresse. En programmation, les cases mémoire sont représentées par des noms. Le programmeur ne connait pas donc l'adresse d'une case mais plutôt son nom.

Les données ainsi que les résultats des calculs intermédiaires et finaux sont rangés dans des cases mémoire qui correspondent à des variables. Ainsi une variable est rangée dans un emplacement mémoire nommé de taille fixe prenant au cours du déroulement de l'algorithme, un nombre indéfini de différentes valeurs.

a) Variable

Elle peut stocker des nombres, des caractères, des chaînes de caractères,... dont la valeur peut être modifiée au cours de l'exécution de l'algorithme.

b) Constante

Elle représente des nombres, des caractères, des chaînes de caractères,... dont la valeur ne peut pas être modifiée au cours de l'exécution de l'algorithme.

Les variables et les constantes sont définis par :

- L'identificateur : c'est le nom de la variable ou de la constante, il est composé de lettres et de chiffres.
- Le type : il définit la nature de la variable ou de la constante (entier, réel).

c) Types de base

Le type d'une variable est l'ensemble des valeurs qu'elle peut prendre. Généralement, il existe cinq types de base :

1- Type entier : est un type numérique qui représente l'ensemble des entiers naturels et relatifs, tels que : 0, 55, -1, etc.

Mot clé en algorithmique : **entier.**

2- Type réel : est un type numérique qui représente les nombres réels, tels que : -0.5, 1.67, $2.8e^{-10}$, etc.

Mot clé en algorithmique : **réel.**

3- Type caractère : représente tous les caractères alphanumériques tels que :

'5', 'B', 'a', '&', '@', ' ', etc.

Mot clé en algorithmique : **car.**

4- Type chaînes de caractères : concerne des chaines de caractères tels que les mots ou les phrases : "Mohamed", " Mohamed Ibn Moussa EL KHAWRIZMI", etc.

Mot clé en algorithmique : **chaîne.**

5- Type booléen : ce type ne peut prendre que deux états : vrai ou faux

Mot clé en algorithmique : **booléen.**

Exemple de déclaration :

✓ Dans un algorithme la partie déclaration consiste à énumérer toutes les variables dont on aura besoin. Chaque déclaration doit comporter le nom de la variable (identificateur) et son type.

Syntaxe : Variable identificateur : type ;

```
Variable  A : entier ;
          moyenne, note1, note2 : réel ;
          nom : chaîne ;
          lettre : car ;
Constant  n = 5 ;
          arobase= '@' ;
          e = "425" ;
```

✓ L'identificateur doit obligatoirement commencer par une lettre suivie d'une suite de lettre et de chiffres et il ne doit pas contenir d'espace.

2- L'affectation

C'est une instruction simple qui permet d'affecter une valeur à une variable

Syntaxe : variable ← expression ;

Une instruction d'affectation se fait toujours en deux temps : évaluation de l'expression située à droite, et affectation du résultat à la variable située à gauche.

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

Exemples :

a ← 15 ; b ← a ;
somme ← a + b ;

❖ Les opérateurs :

Un opérateur est un signe qui peut relier deux valeurs pour produire un résultat. Un opérateur dépend du type des valeurs qu'il relie.

a- Opérateurs arithmétiques (numériques) :

+	Addition
-	Soustraction
*	Multiplication
/	Division
mod	Modulo
^	Puissance

b- Opérateurs de comparaison

>	Supérieur
<	Inférieur
>=	Supérieur ou Egal
=<	Inférieur ou Egal
=	Egal
≠	Différent

c- Opérateurs logiques (booléens) :

et	Fonction et
ou	Fonction ou
non	Fonction non
non et	Fonction non et
non ou	Fonction non ou

d- Opérateur de concaténation : &

Cet opérateur permet de « concaténer » ou relier deux chaines de caractères.

Exemple : "bonjour " & "Monsieur" = "Monsieur bonjour".

Type	Opérations possibles	Symbole ou mot clé correspondant
Entier	Addition Soustraction Multiplication Division Division entière Modulo (le reste de la division entière) x exposant y Comparaisons	+ - * / (DIV en VB et % en C) (MOD en VB) ^ : en vb et pow(x,y) en C <, =, >, <=, >=, ≠
En algorithmique nous symbolisons la division entière par DIV et le reste de la division entière par MOD		
Réel	Addition Soustraction Multiplication Division Exposant Comparaisons	+ - * / ^ <, =, >, <=, >=, ≠
Caractère	Comparaisons	<, =, >, <=, >=, ≠
Chaîne	Concaténation Comparaison	(+ , & : en VB) <, =, >, <=, >=, ≠
Booléen	Logiques	ET, OU, NON et OU_{ex}

Remarque :

En algorithmique, le signe de l'affectation est le signe ←. Mais en pratique, la plupart des langages de programmation emploient le signe égal.

3- Les notions de lecture et d'écriture

Supposons qu'on veut calculer le carré d'un nombre (3 par exemple), l'algorithme le plus simple serait de ce genre :

Algorithme carré ;
Variable a : entier ;
Début
a ← 3 ^ 2 ;
Fin

Mais, le carré est calculé et affecté à la variable a, mais l'utilisateur ne peut jamais connaitre ce résultat (puisque tout se fait dans la mémoire). D'autre part, si on veut calculer le carré d'un autre nombre, est-ce que serait intéressant de modifier l'algorithme? Pour cela, il existe deux instructions permettant à la machine de communiquer avec l'utilisateur :

1- L'instruction de lecture

Permet de recevoir une valeur (à un clavier ou un fichier) et de l'attribuer à une variable.

Syntaxe : Lire (identificateur) ;

Exemple :

Lire (a);
Lire (a, b, c);

2- L'instruction d'écriture

Permet d'afficher une valeur d'une variable sur l'écran.

Syntaxe : Écrire (expression);

Expression peut être une valeur, un résultat, un message, le contenu d'une variable, etc.

Exemple 1 :

Écrire (A) ;

Cette instruction permet d'afficher à l'écran la valeur de la variable A.

A ← 2 ;

Écrire ("La valeur de A est : ", A) ;

Cette instruction permet d'afficher à l'écran : La valeur de A est : 2.

Exemple 2 :

Algorithme produit ;
Variables x, y, z : entier
Début
Écrire ("Donnez la valeur de x :") ;
Lire (x) ;
Écrire ("Donnez la valeur de y :") ;
Lire (y) ;
z ← x * y ;
Écrire (z) ;
Fin

Remarque :

Avant de lire une variable, il est recommandé d'écrire des libellés a l'écran afin de prévenir L'utilisateur de ce qu'il doit frapper, sinon il passe son temps à se demande ce que l'ordinateur de lui.

Exercice : Écrire un algorithme qui permet d'échanger les valeurs de deux variables.

Solution :

Algorithme permutation ;
Variable a,b,c : réel ;
Début
Écrire ("Donnez valeur de a :") ;
Lire (a) ;
Écrire ("Donnez valeur de b :") ;
Lire (b) ;
c ← a ;
a ← b ;
b ← c ;
Écrire (c) ;
Fin

4- Les structures de contrôles

Il a été démontré que pour représenter n'importe quel algorithme, il faut disposer des trois possibilités suivantes:

- La structure de séquence qui indique que les opérations doivent être exécutées les unes après les autres.
- la structure de répétition qui indique qu'un ensemble d'instructions doit être exécuté plusieurs fois.
- la structure de choix qui indique qu'un ensemble d'instructions doit être exécutée suivant les circonstances.

Un algorithme (ou programme) c'est la combinaison de quatre instructions élémentaires qu'on peut les classé en deux catégories :

- Les instructions de base : qui permettent la manipulation de variables telles que l'affectation, la lecture et l'écriture.

- Les instructions de structuration : dans cette catégorie on utilise des structures de contrôle qui précisent l'enchainement chronologique des instructions de base. Ces structures sont les structures conditionnelles et les structures répétitives.

4-1 Structure de contrôle conditionnelle

Dans les notions précédentes, on a vu des algorithmes dans lesquels les instructions s'exécutent dans l'ordre de leur écriture (exécution séquentielle). Mais, la puissance d'un algorithme (programme) provient du fait qu'il peut effectuer des choix dans la façon d'exécuter les instructions.

Par exemple, dans un algorithme de calcul d'impôts, le montant à payer diffère selon le revenu déclaré. L'algorithme doit donc être capable d'appliquer à chaque tranche de revenu, un taux différent. Il effectuera par conséquent des choix sur les opérations à exécuter en fonction du revenu.

Donc, il s'agit du deuxième type d'instructions appelées conditionnelles, ou alternatives. Pour exprimer ce type d'instructions on utilise les structures conditionnelles ou tout simplement les tests.

Il existe deux formes possibles pour un test : la forme réduite et la forme complète.

1- Structure réduite :

Dans cette structure, seule la situation correspondant à la validation de la condition entraine l'exécution des instructions, l'autre situation conduisant systématiquement à la sortie de la structure.

Syntaxe :

Si Condition **Alors**

Instruction(s) ;

2- Structure complète :

Dans cette structure, l'exécution d'un des deux traitements distincts ne dépend que du résultat du test effectué sur la condition. Si cette dernière est vérifiée, seul le premier traitement est exécuté ; si elle n'est pas vérifiée, seul le deuxième traitement est exécuté.

Syntaxe :

Si Condition **Alors**
Instruction(s) 1 ;
Sinon
Instruction(s) 2 ;

Exemples :

- ✓ Tester si un nombre entier est pair ou impair :

r ← n mod 2 ;
Si (r = 1) **Alors**
Écrire (n, "est un nombre impair") ;

Sinon

Écrire (n, "est un nombre pair") ;

✓ Tester si un nombre n est positif ou négatif :

Lire (n) ;

Si (n≥0) **Alors**

Écrire (n, "est un nombre positif") ;

Sinon

Écrire (n, "est un nombre négatif") ;

3- Tests imbriqués

Un test avec Si ouvre donc deux voies, correspondant à deux traitements différents. Mais dans certains cas, ces deux voies ne suffisent pas pour toutes les solutions possibles.

Par exemple, un programme devant donner l'état de l'eau selon sa température doit pouvoir choisir entre trois réponses possibles (solide, liquide ou gazeux).

Une première solution serait la suivante :

Début

Écrire ("Entrez la température de l'eau :") ;

Lire (Temp) ;

Si (Temp =< 0) **Alors**

Écrire ("Etat solide") ;

Si (Temp > 0 Et Temp < 100) **Alors**

Écrire ("État liquide") ;

Si (Temp >= 100) **Alors**

Écrire ("État gazeux") ;

Fin

Une autre solution consiste à imbriquer les tests de la manière suivante :

```
Algorithme Temps;
Variable Temps : entier ;
Début
Écrire ("Entrez la température de l'eau :") ;
Lire (Temps) ;
Si (Temps =< 0) Alors
Écrire ("État solide") ;
Sinon Si (Temps < 100) Alors
        Écrire ("État liquide") ;
        Sinon
        Écrire ("État gazeux") ;
 Fin
```

Donc, au lieu de devoir taper trois conditions (dont une est composée) on peut utiliser deux tests imbriqués (nous n'avons plus que deux conditions simples).

5- Structure de contrôle répétitive

On appelle boucle ou structure répétitive (dite aussi itérative) tout ensemble d'instructions qui peut être exécuté plusieurs fois. Par exemple, un programme de calcul de la paye répètera pour chaque employé les mêmes instructions pour établir une fiche de paye.

Généralement, on peut considérer deux cas pour les boucles. Dans le premier cas, le nombre de répétitions n'est pas connu ou il est variable, il existe deux structures pour ce cas : la boucle « Répéter... jusqu'à » et la boucle « Tant que ... faire ». Dans le deuxième cas, si le nombre de répétitions est connu, on peut utiliser la boucle « Pour ».

1- La boucle « Répéter... Jusqu'à »

Dans cette structure, le traitement est exécuté une première fois puis sa répétition se poursuit jusqu'à ce que la condition soit vérifiée.

Syntaxe :

Répéter
Action ;
Jusqu'à (condition) ;

L'action dans cette boucle est toujours exécutée au moins une fois.

Exemple :

Variables n, p : entier ;
Début
Répéter
Écrire ("Donner un nombre :") ;
Lire (n) ;
p ← n*n ;
Écrire (p) ;
Jusqu'à (n=0) ;
Écrire ("Fin de l'algorithme") ;
Fin

Les instructions délimitées par les deux mots répéter et jusqu'à constitue le bloc de la boucle qu'il faut répéter jusqu'à ce que la condition n=0 soit vérifiée. Donc le nombre de répétitions de cette boucle dépend des données fournies par l'utilisateur.

Remarque :

Dans la structure «répéter... jusqu'à», la condition telle qu'elle est exprimée ci-dessus, constitue une condition d'arrêt de la boucle ; mais réellement, cela diffère selon le langage de programmation utilisé. Par exemple, en langage Pascal, la condition de la boucle « répéter... jusqu'à » est une condition d'arrêt. Alors qu'en langage C, cette condition est exprimée en tant qu'une condition de continuation.

2- La boucle «Tant que… Faire»

Dans cette structure, on commence par tester une condition, si elle est vérifiée, le traitement est exécuté.

Syntaxe :

Tant que (condition) **Faire**
Action ;

L'action dans cette boucle peut ne jamais être exécutée.

Exemple :

Variables n, p : entier ;
Début
Écrire ("Donner un nombre :") ;
Lire (n) ;
Tant que ($n \neq 0$) **Faire**
Début
Écrire ("Donner un nombre :") ;
Lire (n) ;
$p \leftarrow n*n$;
Écrire (p) ;
Fin tant que
Écrire ("Fin de l'algorithme") ;
Fin

3- La boucle « Pour »

Cette structure est utilisée lorsqu'on sait exactement combien de fois on doit répéter un traitement. Donc, la boucle s'arrête si le nombre souhaité d'itérations est atteint.

Cette structure utilise une variable de contrôle caractérisée par sa valeur initiale et sa valeur finale.

Syntaxe :

Pour (variable ← valeur initial, valeur final) **faire**

Action;

Exemple :

Soit l'algorithme suivant qui compte jusqu'à 100.

Variables n, p : entier ;
Début
Pour (i ← 1, 100) **Faire**
Écrire (i);
Fin

L'indice i est une variable dont la valeur est augmentée de 1 à chaque passage dans la boucle. Il sert donc à compter le nombre de fois que la boucle est exécutée. Dans la structure « pour » la variable indiciaire (l'indice) joue le rôle du compteur.

On peut représenter l'utilisation d'un tel compteur dans les boucles «répéter…jusqu'à» et «tant que … faire» respectivement comme suit :

i ← 0 ;	i ← 0 ;
Répéter	**Tant que** (condition) **Faire**
Instruction(s) ;	**Début**
…	Instruction(s) ;
i ← i +1 ;	…
Jusqu'à (condition) ;	i ← i +1 ;
	Fin Tant que

On remarque que la variable «i», utilisé ci-dessus comme compteur, a été initialisée à zéro (0) avant le début de la boucle. Donc, il faut toujours initialiser le compteur avant de commencer.

L'instruction d'affectation «i ← i +1» signifie : augmenter la valeur de «i» de un (1). Elle peut être placée n'importe où, l'essentiel qu'elle soit à l'intérieur de la boucle.

Exemple : Une structure qui affiche les dix premier nombres pairs.

```
Pour (i ← 1, 10) Faire
Écrire (i*2);
```

```
i ← 1 ;
Tant que (i≤10) Faire
Début
Écrire (i*2);
i ← i +1 ;
Fin Tant que
```

4- Les boucles imbriquées

On parle de boucles imbriquées lorsqu'une boucle contient elle-même une autre boucle. Les deux boucles peuvent être les mêmes ou différentes.

Exemple :

Les deux boucles imbriquées qui affichent à l'écran le triangle suivant :

```
Pour (i ← 1, 4) Faire
Pour (j de 1 à i) Faire
Écrire ("*") ;
```

```
*
**
***
****
```

Dans cet exemple, à chaque itération de la boucle 1, la boucle 2 s'exécute jusqu'à la fin, et ainsi de suite jusqu'à la fin des deux boucles.