1 Algorithmique : Bref Historique & Définitions

1.1 Bref historique

- Le Terme ***algorithme*** vient du nom du mathématicien arabe musulman (780-850) perse du 9ième siècle ***al-Khwarizmi*** (780-850).
- Le terme algorithme ne concerne pas que l'informatique, et la notion de l'algorithme a précédé celle de l'informatique.
- Les premiers algorithmes dont on a retrouvé décrivent des méthodes de calcul mathématique. Par exemple, l'algorithme d'Euclide qui permet de calculer le PGDC de deux nombres entiers a et b :
    1) Diviser a par b, on obtient r
    2) Remplacer a par b
    3) Remplacer b par r
    4) Continuer tant que c'est possible, sinon on obtient le PGDC.

Exemple introductif : Algorithme (Préparation béton).

1) Acheter la matière première (Ciment, concasseur, sable) ***les données***

2) Verser deux brouettes du concasseur sur une Brouette du sable.***(Instruction 1)***

3) Verser un sac du ciment sur le mélange et mélanger.***(Instruction 2)***

4) Verser 60 L d'eau sur le mélange puis mélanger. ***(Instruction 3)***

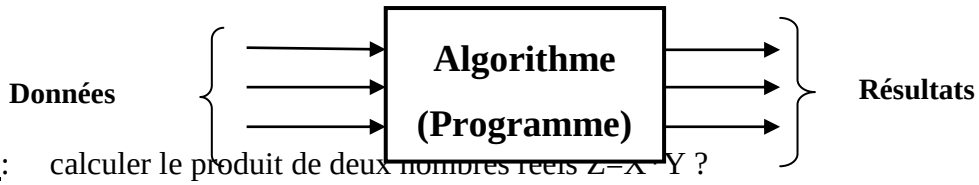
5) On obtient alors le béton***. (résultat)***

1.2 Définitions

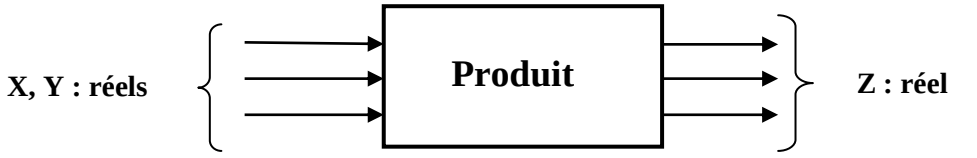
1.2.1 Algorithme

- Un ***algorithme*** est une suite finie et non ambiguë d'opérations ou *d'**instructions*** permettant de résoudre un problème.
- Un ***algorithme*** est une méthode, pour arriver dans un temps fini, à un ***résultat*** déterminé à partir d'une situation ***donnée***.

Le traitement de l'information par ordinateur consiste à faire élaborer, par cette machine, des informations appelées ***résultats***, à partir d'informations appelées ***données*** :



Exemple : calculer le produit de deux nombres réels Z=X*Y ?



1.2.2 Langage algorithmique

C'est le langage utilisé pour décrire et définir les algorithmes. En utilisant un ensemble de ***mots-clés*** et de structures permettant de décrire de manière complète, claire, l'ensemble des opérations à exécuter sur des données pour obtenir des résultats.

1.2.3 Mot-clé

Un mot clé est un identificateur qui a une signification particulière (algorithme, début, fin, si, alors……..)

1.2.4 Algorithmique

L'***algorithmique*** est la science qui étudie les algorithmes.

1.2.5 Programme

Un programme informatique est une séquence d'instructions écrites dans un langage compréhensible par l'ordinateur (langage de programmation).

En d'autres termes, un programme informatique est la traduction de l'algorithme dans un langage de programmation.

1.2.6 Langage de programmation

Ensemble de commandes et de mots clés nécessaires pour l'écriture d'un programme afin qu'il soit compréhensible par l'ordinateur

Examples: Basic, Fortran, C, C++, Pascal, Delphi, java ….

1.3 Structure et trace d'exécution d'un algorithme

1.3.1 Structure générale d'un algorithme

- Un algorithme est composé de 3 parties : l'entête, la déclaration et le corps de l'algorithme**.**

```
ALGORITHME  Nom-Algorithme
        Constantes
        Types
        Variables
        Fonctions
        Procédures
DEBUT
        Instruction  1 ;
        Instruction  2 ;
        …
        Instruction  n ;
 FIN.
```

L'en-tête : il permet tout simplement d'identifier (nommer) un algorithme.

Les déclarations : c'est la liste de tous les objets (Constantes, variables, types, fonctions et procédures….) utilisés et manipulés dans le corps de l'algorithme.

Le corps : Le corps contient la suite des instructions à exécuter (ensemble d'opérations à exécuter sur les données i.e variables).
Exemple : un algorithme qui calcule le produit de deux nombres entiers (Premier algorithme)

```
Algorithme Produit
    Nombre1, Nombre2, Produit : entier ;
Début
    Nombre1 ← 5 ;
    Nombre1 ← 4 ;
    Produit ← Nombre1*Nombre2 ;
Fin.
```

1.3.2 Trace d'exécution d'un algorithme :

- L'exécution d'un algorithme (programme) se fait instruction par instruction selon l'ordre indiqué par l'algorithme (programme).
- L'exécution commence à partir de la première instruction qui suit le mot clé **DEBUT** et se termine à la dernière instruction qui précède juste le mot clé **FIN**.
- Au début les valeurs des variables sont inconnues (?).
- Quant une variable reçoit une valeur cette valeur est sauvegardée jusqu'à ce qu'une autre instruction change cette valeur.

Exemple :

Etape	a	b
Instruction1	10	?
Instruction2	10	15
Instruction3	3	15

```
Algorithme Trace
    a : entier;
    b : entier;
Début
    a ← 10 ;
    b ← a+ 5 ;
    a ← 3 ;
Fin.
```

1.4 Résolution d'un problème sur ordinateur :

La réalisation d'un programme exécutable par un ordinateur, nécessite le suivi d'une démarche constituée de 4 phases, la place de l'algorithme (phase 2) est montrée dans la figure suivante :

1.4.1 Analyse du problème

L'analyse consiste à :

- ✓ Extraire les données initiales ***(entrées)***,
- ✓ Définir l'objectif ou les objectifs du problème ***(sorties ou résultat),***
- ✓ Déduire la méthode à suivre pour résoudre et atteindre ces objectifs.

1.4.2 Ecriture des algorithmes

- ✓ Ecriture de l'ensemble d'algorithmes correspondants

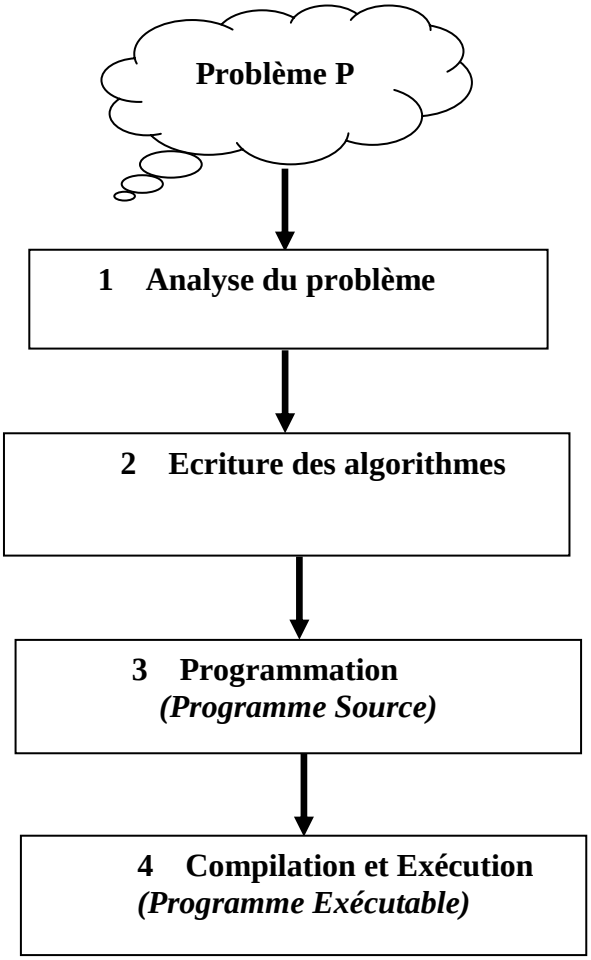
1.4.3 **Programmation**

Dans cette étape on va traduire l'algorithme en un langage de programmation (C,C++, Pascal, Fortran, Java …etc). On obtient alors le ***code source*** du Programme qui est stocké dans un ou plusieurs ***fichiers*** sur l'ordinateur.

1.4.4 **Compilation et Exécution**

Dans cette étape le code source du programme est traduit *en* ***langage machine*** en utilisant un Logiciel spécial appelé ***compilateur,*** on obtient *al*ors ***le code machine du Programme,*** ce code machine est ensuite exécuté directement par l'ordinateur.

Il existe plusieurs logiciels (d'application) qui intègrent un compilateur pour le langage C comme : ***Dev C++,***Turbo C, Microsoft Visuel C++ Express…



Exemple :(Problème)

Réaliser un programme qui calcule l'intensité du courant électrique« I » qui passe dans un circuit fermé alimenté par une source dont la différence de potentiel est « U=220 Volt »et qui contient une résistance « R=12 Ω ».

Solution :

1) **Analyse du problème :**

- ✓ Données d'entrées: U=220 Volt, R=12 Ω.
- ✓ Résultats ou Sorties (Objectifs) : intensité I calculée
- ✓ Méthode : **U=R*I,** d'où **I=**$\frac{U}{R}$

2) Ecriture des algorithmes :

Algorithme calcul _intensité
U,R,I : entiers ;
Début
U←220 ;
R←12;
I←$\frac{U}{R}$;
Fin

3) Programmation : (voir **TP)**

Programme en langage *pascal*	**Programme en langage *C***	**Programme en langage *C++***
Program calcul _intensité ; Uses crt ; Var U,R,I : integer ; **Begin** U **:=**220 ; R **:=**12; I **:=** U/R ; **End.**	#include <stdio.h> int U,R,I ; **void main()** **{** U **=** 220 ; R**=**12; I = U/R ; **}**	#include <iostream> using namespace std; int U,R,I ; **void main()** **{** U **=** 220 ; R**=**12; I = U/R ; **}**

4) Compiler le programme : (voir **TP)**

1.5 Règles de base de construction d'un bon algorithme

- Définir clairement et sans ambiguïtés le problème posé.
- Un algorithme doit être bien structuré.
- Le résultat doit être atteint en un nombre fini d'étapes. Il ne faut donc pas de boucles infinies.
- Il faut étudier tous les cas possibles de données, ...
- Le résultat doit répondre au problème demandé. Attention, un jeu d'essais ne prouve ni ne garantit JAMAIS qu'un programme soit correct. Il peut seulement prouver qu'il est faux.
- Un algorithme doit être commenté. Le commentaire est une phrase que l'on peut insérer à n'importe quel niveau de l'algorithme (programme). Il sert uniquement à introduire des explications destinées au lecteur de l'algorithme. il est totalement ignoré par l'exécutant. {commentaire}.
- Un algorithme doit être Modulaire, s'il s'agit d'un algorithme qui peut être décomposé.
- Un algorithme doit être Efficace, parmi quelques critères : temps d'exécution, quantité de l'espace mémoire…

1.6 Variables

1.6.1 Notion de variable

- Les variables d'un algorithme contiennent les informations nécessaires à son déroulement.
- Une ***variable*** dans un programme est un espace mémoire identifié par un nom, destiné à stocker une valeur, qui peut être modifiée durant les traitements (opérations).
- Chaque variable est caractérisée par un ***nom*** (identifiant) et un ***type*.**

1.6.2 L'identificateur (le nom)

pour identifier une variable on utilise les lettres de l'alphabet (a,b,c,A …) et les chiffres (0,1,2 , …) à condition que le ***premier caractère*** soit une lettre d'alphabet, l'identificateur peut aussi contenir un trait d'union"_".

Exemples :

X, y, nom, prix, X1: sont des identificateurs (noms) de variables.
1x : n'est pas un identificateur puisqu'il commence par un chiffre
nom*, Met%un1 : ne sont pas identificateurs de variables car les deux contiennent un caractère spécial.

1.6.3 Le type :

Le type correspond au genre d'information que l'on souhaite utiliser :

- ✓ ***Entier*** pour manipuler des nombres entiers (1, 115, -7,…),
- ✓ ***Réel*** pour manipuler des nombres réels (11. 3, 15.7, -4.3,……),
- ✓ ***Booléen*** pour manipuler des valeurs booléennes (*vrai* ou *faux),*
- ✓ ***Caractère*** pour manipuler des caractères alphabétiques et numériques (a, b, 3,…),
- ✓ ***Chaîne de caractère****s* pour manipuler des chaînes de caractères permettant de représenter des mots ou des phrases (mila, omar,….).

Remarque :

- L'identificateur doit être différent de tous les mots clés.
- Pour la lisibilité du code choisir des noms significatifs : ***exp*TotalVentes2004, Prix_TTC, Prix_HT, somme, …**

1.6.4 Déclaration d'une Variable :

Toutes les variables doivent être **déclarées** avant d'être utilisées.
Syntaxe :Nom de la variable : Type de la variable ;
Exemple :

```
Algorithme exp_decl
        a : entier;
        c: caractère;
        Age : entier;
        prénom: chaine de caractères;
        x,y,z: réel;
Début
        { Suite d'instructions }
Fin.
```

Remarque : On peut déclarer plusieurs variables de même type sur une seule ligne en les séparent par des virgule (par exemple : a,b,c :entier ;).

1.7 Types des variables

1.7.1 Le type entier (int)

Le type entier est muni des opérateurs suivants :

- ✓ Les opérateurs arithmétiques classiques : + (addition), - (soustraction), * (produit).
- ✓ la division entière, notée ***div*** , telle que ***n div p*** donne la partie entière du quotient de la division entière de n par p .
- ✓ le modulo, noté ***mod***, telle que ***n mod p*** donne le reste de la division entière de n par p.
- ✓ les opérateurs de comparaison classiques : **<, >, =, ...**

Exemples :

7 div 2 = 3 7 mod 2 = 1 sqrt (25) = 5 abs (- 17) = 17

2.9.2. Le type réel (float et double)

Les opérations valides sur les réels sont :

- ✓ les opérations arithmétiques classiques : + (addition), - (soustraction), * (produit), / (division).
- ✓ les opérateurs de comparaison classiques : <, >, =, ...
- ✓ La fonction qui fournit la racine carrée ***(sqrt).***

2.9.3. Le type booléen :

Il s'agit d'un type dont les seules valeurs sont ***vra i***ou ***faux.*** Les opérateurs logiques (opération) sur ce type sont : non, ou, et. Ces opérateur sont définis par les tables suivantes appelées tables de vérité :

- ✓ **Opérateur logique (NON):**

Non	
vrai	faux
faux	vrai

- ✓ **Opérateur logique (ET):**

Et	*vrai*	*faux*
vrai	vrai	faux
faux	faux	faux

- ✓ **Opérateur logique (OU):**

Ou	*vrai*	*faux*
vrai	vrai	vrai

faux	vrai	faux

2.9.4. Le type caractère

C'est un type constitué des caractères alphabétiques et numériques. Une variable de ce type ne peut contenir qu'un seul caractère.

Les opérations prédéfinies sur les caractères sont :

- ✓ ***Succ(c)*** : elle fournit le caractère qui suit immédiatement le caractère c.
- ✓ ***Pred (c)*** **:** fournit le caractère qui précède immédiatement le caractère c.

Exp : Succ ('B')='C', Pred('z')='y'.

2.9.5. Le type chaîne de caractères :

Une chaîne est une suite de caractères. Une chaîne est encadrée par deux Guillemets. **Exp :**"informatique" " cour" "mila" "-1,6" …..

Les opérations prédéfinies sur les chaînes sont :

- ✓ ***les comparaisons :*** <, >, =, ... selon l'ordre lexicographique (ASCII).
- ✓ ***La Concaténation représenter par un*** **+** : elle fournit la chaîne obtenue par concaténation des deux chaînes.

Remarque :

- ➢ Les chaînes sont ordonnées selon l'ordre lexicographique.

Exp : 'art' < 'cours', 'cour' < 'cours', 'courage' < 'cours'.

2.10. Les constantes :

Une constante est un objet algorithmique qui prend une seule valeur durant toute l'exécution.

Exemple :

```
Algorithme exp_cons
        Const  Pi ← 3,14 ;
        A : entier;
         Y : réel;
Début

         X ← 5 ;
        Y←X+Pi ;
        Pi ← X+1 ;   (faux)
Fin.
```

Donc Pi aura toujours la valeur 3,14.

Remarques :

- ➢ Pour déclarer une variable, on définit son ***nom*** et son ***type.***
- ➢ Pour déclarer une constante, on doit utilise le mot clé ***CONST***. Et on définit son ***nom*** et sa ***valeur***.
- ➢ On commence toujours par la déclaration des constantes avant les variables.

2.11. Expressions

2.11.1.Qu'est ce qu'une expression

- Une expression représente une combinaison ***d'opérandes*** et ***d'opérations*** effectuées sur ces opérandes.
- Les opérandes sont :
  - ✓ Les Variables
  - ✓ Les Valeurs
- Les opérateurs (opérations) sont :
  - ✓ Les opérateurs algébriques : +,-,*, /, div, mod.
  - ✓ Les opérateurs logiques : et, ou, non.
  - ✓ Les opérateurs relationnels :<,<=,>,>=,=,<>.
- A toute expression est associé un ***type*** qui est le type de la valeur de cette expression.

Exp : a+8 est une expression qui représente une opération d'addition portant sur les opérandes a et 8.

Remarque :

Un opérateur qui s'applique sur deux opérandes est dit opérateur ***binaire*** (Exp : +) alors qu'un opérateur qui s'applique sur un seul opérande est dit ***unaire*** (Exp : Non).

2.11.2.Règles d'évaluation d'une expression :

- Le calcul de ***la valeur*** d'une expression qui comporte plus d'un opérateur dépend du sens que l'on donne à cette expression.

2 **Exp :** 5-4 div 2 est une expression ambiguë.

- Les parenthèses permettent de résoudre ce problème :
  - ✓ (5-4) div 2 =0.
  - ✓ 5- (4 div 2) =3.
- En cas d'absence de parenthèses, et pour éviter toute ambiguïté, des règles d'évaluation ont été établies. Il existe un ***ordre de priorité*** entre les opérateurs de façon décroissante comme suit :
  1) Opérateurs unaire : non.
  2) Opérateurs multiplicatifs : *, /, div, mod, et.
  3) Opérateurs additifs : +,-, ou.
  4) Opérateurs relationnels : <, <=,>,>=, <>.

  Exp : Donc le sens attribué à 5-4 div 2 est bien 5-(4 div 2) =5-2=3.
- Si une expression comporte des opérateurs de même priorité alors : Les opérateurs de même priorité sont associatifs à gauche.

  Exp : 5-3-2 est interprétée comme (5-3)-2=2-2=0.

Remarque :

En algorithmique, pour éviter toute ambiguïté, il faut toujours introduire les parenthèses.

2.12. Instructions

2.12.1. Définition d'une instruction

- Une instruction représente une ou plusieurs ***actions*** (opérations) portant sur une ou plusieurs ***variables***.
- Il existe deux types d'instruction élémentaires :
  - ✓ ***Instruction d'affectation***
  - ✓ ***Instruction de lecture /écriture***.

2.12.2. Instruction d'affectation :

- L'affectation est une instruction qui stocke ***la valeur*** d'une expression dans ***une variable***.

3 Syntaxe d'affectation :

4 **X←exp ;** se lit : x reçoit exp.
5 Ou :
6 **X** : variable,
7 **← :**L'opérateur d'affectation,
8 **exp** : expression.

Ce qui revient à remplacer la valeur initiale de **x** par la valeur de l'expression

Exemple :

```
Algorithme exp_Aff
      a, b : entier;
Début
    a ← 12 ;
    b ←a+ 4 ;
Fin.
```

Execution: a=12 b=16

2.12.3. Instruction de lecture / écriture :

a) L'instruction de lecture :

- Cette instruction permet la lecture de la ***valeur*** d'une ***variable*** à partir du ***clavier***.

9 Syntaxe :

10 Lire (variable) ;

- L'exécution de cette instruction consiste à affecter une valeur à la variable en prenant cette valeur sur le périphérique d'entrée (clavier).

11 **Exemple :** Supposons X une variable de type entier alors:
12 Lire(X); Affectera une valeur de type entier tapée par le clavier à la variable X.

b) L'instruction d'écriture :

- Cette instruction permet l'affichage sur écran.
- Il y a deux types d'affichage :
  - ✓ Soit on affiche des valeurs de variables :

    13 **Syntaxe :** **Ecrire (variable) ;**
  - ✓ Soit on affiche des textes:

14 **Syntaxe :** **Ecrire (' texte') ;**

15 Exemples :

- Ecrire (2*x+5) : permet d'afficher la valeur de l'expression 2*x+5. Si x vaut 10 alors cette instruction affiche 25.
- Ecrire ('' Le résultat est : '') : permet d'afficher le texte :Le résultat est :
- Ecrire ('' Le résultat est : '', X) : si x vaut 15 cette instruction affiche : Le résultat est : 15.

2.13. Les commentaires :

- pour écrire l'algorithme sous une forme claire et logique il est possible de rajouter des commentaires aux actions (instructions) algorithmiques.
- Par convention, un commentaire :
  - ✓ Commence par slash (/) suivie d'une étoile * et se termine par une étoile * suivie d'un slash : **/* plusieurs lignes */**
  - ✓ commence par double slash pour un commentaire qui s'étale sur une seule ligne : **// Une seule ligne**
- Un commentaire n'est pas une instruction. Il n'a aucun effet sur l'exécution de l'algorithme.

Exemple :

Écrire un algorithme qui fait la somme de deux nombres entiers ?

```
Algorithme exp_comt
        a, b, somme : entiers ;
Début
    Lire (a, b) ;          /* cette instruction permet de lire les valeurs
                           des variables  a et b*/
    Somme ← a+b ;    // calcule de la somme
    Ecrire (somme) ;   // écrire le résultat
Fin.
```