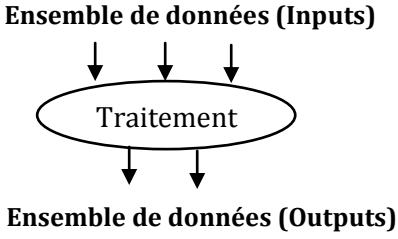
CHAPITRE II : NOTIONS D'ALGORITHME ET DE PROGRAMME

II.1. Définition d'un Algorithme

Un Algorithme est une séquence d'instructions ordonnées, qui permet de résoudre un problème. Le terme "*Algorithme* " vient de l'arabe الخُوَارِزْمِيّ, nom du mathématicien perse Al-Khwarizmi.
Un algorithme prend, en entrée, un ensemble de données (Inputs) et délivre (produit, renvoie) un ensemble de données en sortie (Outputs).



II.2. Les étapes de résolution d'un problème

La résolution d'un problème, en informatique, passe par différentes étapes. Ces dernières sont schématisées dans la Figure 1 :

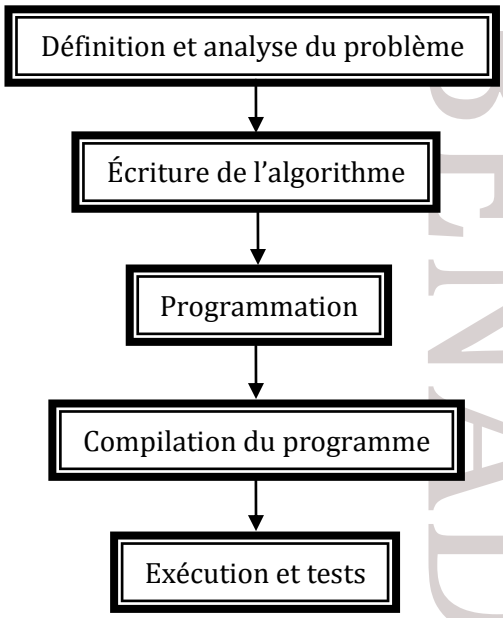


Figure 1 Démarche de résolution d'un problème

* **Définition et analyse du problème :** L'objectif de cette étape est de bien comprendre l'énoncé du problème et de déterminer :
  - Les résultats attendus (sorties),
  - Le traitement à effectuer (méthodes et formules de calculs pour atteindre le résultat),

- Les données nécessaires au traitement (entrées).

* **Écriture de l'algorithme :** Une fois l'analyse terminée, il faut écrire les instructions dans leur ordre logique d'exécution et obtenir ainsi un algorithme. Ce dernier peut être écrit dans n'importe quelle langue.

* **Programmation :** C'est la traduction de l'algorithme en programme en utilisant un langage de programmation.

  - **Langage de programmation :** C'est l'ensemble de la syntaxe (commandes et instructions) utilisé par les développeurs afin d'écrire un **programme.** Il existe plusieurs langages de programmation, comme C/C++, Pascal, Java, Fortran, Matlab, etc.

* **Compilation du programme :** Désigne le procédé de traduction d'un programme, écrit et lisible par un humain (code source), en un programme exécutable par un ordinateur (code binaire).

* **Exécution et tests :** Une fois compilé, un programme doit être testé pour s'assurer de son fonctionnement et qu'il répond aux besoins exprimés par l'utilisateur. Un programme est testé par des valeurs différentes de données (jeu de test).

II.3. Structure d'un Algorithme :

Un algorithme se compose de trois parties principales : L'en-tête, la partie déclarative et le corps de l'algorithme. Ces parties doivent respecter une syntaxe bien déterminée, définie comme suit :

```
L'En-tête        Algorithme  <identificateur_algo> ;
La partie        Constantes  < identificateur_constante>  = valeur ;
déclarative      Variables   < identificateur_variable> : Type;
                 Début
                 <Instruction 1> ;
Le corps de      <Instruction 2> ;
l'algorithme     ..
                 <Instruction n> ;
                 Fin.
```

Et voici la structure d'un programme écrit en Pascal :

```
Program  <identificateur_algo> ;
Const  < identificateur_constante>  = valeur ;
Var  < identificateur_variable> : Type;
Begin
<Instruction 1> ;
<Instruction 2> ;
..
<Instruction n> ;
End.
```

II.3.1 L'En-tête :

L'en-tête sert à donner un nom à l'algorithme en utilisant un identificateur. Ce dernier est précédé par le mot clé "**Algorithme**". Alors qu'est-ce qu'un identificateur ?

- **Identificateur :** Un identificateur est une chaîne de caractère qui permet de donner un nom unique à un programme (algorithme), une constante, une variable, une procédure ou une fonction. Cette chaîne doit commencer soit par un caractère alphabétique ou par un tiret du 8 (_) et ne peut contenir que des caractères alphanumériques. Aussi, les mots réservés (mots-clés) d'un langage de programmation ne peuvent être utilisés comme identificateurs. Voici quelques mots réservés au langage Pascal: Begin, end, program, var, const, real, integer, char, if, then, else, while, for, do, repeat.

  - **Exemple:**

    B1 ✓ _B ✓ B-1 ✗ Contient un tiret du 6 a54b ✓ B_1 ✓
    Ab y ✗ Contient un caractère blanc (espace) String ✗ Est un mot réservé

II.3.2 La partie déclarative :

La partie déclarative sert à déclarer les différentes données que l'algorithme utilise (Constantes, variables,.. etc.). Une donnée non déclarée et utilisée par l'algorithme engendre une erreur lors de la compilation. Alors qu'est-ce qu'une variable et qu'est-ce qu'une constante ?

- **Constante :** Une constante est une entité dont la valeur ne change pas lors de l'exécution de l'algorithme.

  - **Exemples :**

    Const min = 20 ; {Constante de type entier.}

    Const car = 'm' ; {Constante de type caractère.}

    Const a13 = 'exemple' ; {Constante de type chaîne de caractères.}

    Const b = True ; {Constante de type booléen.}

    Const note = 18.75 ; {Constante de type réel.}

✓ **Remarques:**

- Si on veut déclarer la constante π, par exemple, nous devons le faire comme suit **Const** PI = 3.14 ; Nous devons remplacer le symbole π par le mot PI (ou par un autre identificateur valide : P, Pii, Valeur_Pi, a, x, .. etc.) car π est un caractère grec qu'un identificateur ne peut pas contenir.
- En algorithmique, la virgule (,) des nombres réels est remplacée par un point (.)

- **Variable :** Une variable est une entité dont la valeur, peut changer, lors de l'exécution de l'algorithme. Une variable a un type, ce dernier définit l'ensemble des valeurs que la variable peut prendre.

Le tableau 1 résume les 5 types de base (prédéfinis) utilisés en algorithmique :

Tableau 1

Algorithme	Programme (Pascal)	Valeurs
Entier	Integer	Représente l'ensemble des entiers relatifs Z, tels que : 0, 45, -10,…
Réel	Real	Représente l'ensemble des réels R, tels que : 0.45, -3 , -11.27,…
Booléen	Bool	Ce type prend deux valeurs : vrai (true) ou faux (false)
Caractère	Char	Représente l'ensemble des caractères imprimables : a...z, A...Z, 0...9, /@# ? ...
Chaine de caractères	String	Représente une séquence d'un ou de plusieurs caractères :

- **Exemple :**

  Var taille : real ; {variable de type real.}

  Var c : char ; {variable de type caractère.}

  Var AnneeDeNaissance : integer ; {variable de type entier.}

  Var prenom : string ; {variable de type chaine de caractères.}

  Var b1, b2, b3 : bool ; {variables de type booléen.}

✓ **Remarque :** Les variables de même type peuvent être déclarées en même temps sur la même ligne.

II.4. Types d'instructions :

Une instruction spécifie une opération ou un enchaînement d'opérations à exécuter sur des objets (constante, variable, .. etc.).
Les instructions se situent entre les mots-clés Début (Begin) et Fin (End.).
Les instructions sont séparées par des ; et sont exécutées séquentiellement, c'est-à-dire l'une après l'autre, depuis le Begin jusqu'au End. final.

II.4.1. La lecture :

La lecture est une instruction qui permet d'introduire (saisir) une valeur à l'Algorithme à travers le clavier.
Il n'est pas possible de lire une constante ou une valeur.

Syntaxe :

Algorithme	Programme (Pascal)
Lire (id_var) ;	**Read** (id_var) ; **Readln** (id_var) ; {Lecture avec retour à la ligne.}
Lire (id_var1, id_var2, .., id_varN) ; {Lecture de plusieurs variables en même temps.}	**Read** (id_var1, id_var2, id_var3, .., id_varN) ;

II.4.2. L'écriture :

L'écriture est une instruction qui permet d'afficher, à l'écran, des données. Ces dernières peuvent être un message, une valeur, la valeur d'une variable, une constante et même le résultat d'un calcul.

Syntaxe :

Algorithme	Programme (Pascal)
Écrire (id_var) ;	**Write** (id_var) ; **Writeln** (id_var) ; {Écriture avec retour à la ligne.}
Écrire (' Ceci est un message ! ') ; {Affichage du message : Ceci est un message ! à l'écran.}	**Write** (' Ceci est un message ! ') ;

Remarque : En Pascal, un message est TOUJOURS délimité par deux apostrophes.

Exemples :

Écrire (' Bonjour ') ; {Affiche le message : Bonjour}
Write (' Bonjour ') ;

Écrire (a,b,c) ; {Affiche les valeurs des variables a, b et c côte à côte.}
Write (a,b,c) ; {Il est préférable d'éviter cette écriture car elle peut induire en erreur. Par exemple si a=5, b=3, c=1, l'affichage est 531 et on risque de le lire 531 (cinq cent trente et un).}

Écrire (5+2) ; {Affiche le résultat du calcul c.à.d. 7.}
Write (5+2) ;

Écrire (a+b*c) ; {Affiche le résultat de l'évaluation de l'expression a+b*c. }
Write (a+b*c) ; {Par exemple si a=5, b=3, c=1, l'affichage est 8.}

Écrire (3<7) ; {Affiche le résultat de l'évaluation de l'expression 3<7 c.à.d. true.}
Write (3<7) ;

Voici l'exemple d'un algorithme et de son programme Pascal qui lit et affiche un entier a :

Algorithme lit_ecrit; **Variables** a : entier; **Début** **Écrire** ('Entrez un entier : '); **Lire** (a); { introduction de la valeur de a } **Écrire** ('valeur de a = ', a); **Fin.**	**Program** lit_ecrit; **Var** a : integer; **Begin** **Write** ('Entrez un entier : '); **Readln** (a); **Writeln** ('valeur de a = ', a); **End.**

✓ **Remarque :**

- L'instruction **Écrire**('valeur de a = ', a); est le regroupement des deux instructions **Écrire**('valeur de a = ') et **Écrire**(a), il a suffi juste de séparer le message de la variable a par une virgule de concaténation (,).

- En Pascal, il est possible de commenter un programme, il suffit d'écrire les commentaires entre accolades { } ou de mettre deux slaches // avant le commentaire.
  Par exemple : {Ceci est un commentaire} ou // Ceci est un commentaire.
  Le commentaire n'est pas pris en compte à la compilation. Il sert à rendre le programme plus clair à la lecture, à noter des remarques, etc.

II.4.3. L'affectation :

L'affectation est une instruction qui permet de modifier la valeur d'une variable.

Syntaxe :

Algorithme	**Programme (Pascal)**
a ← b ;	a := b;
{a : **DOIT** être une variable. b peut être une valeur, une constante, une variable ou une expression.}	

Remarque : Les deux côtés d'une affectation doivent être du même type sauf pour le type entier qui peut être stocké dans un réel car l'ensemble des réels inclut l'ensemble des entiers.

Exemples :

a ← 3; {Si a, par exemple, avait la valeur 5, elle devient 3.}
a := 3 ;

c ← a+7 ; {Si a =4 alos la valeur de c devient 11.}
c := a+7 ;

Inf ← a>b ; {Si a = 6 et b = 9 alos la valeur de Inf devient False.}
Inf := a>b ;

Les trois types d'instructions que nous venons de voir (Lecture, écriture et affectation) représentent les instructions qui s'exécutent en **séquentiel** c.à.d. que la fin de l'exécution d'une instruction entraîne l'exécution de l'instruction suivante.

Cependant, il existe un autre type d'exécution dans lequel on est amené soit à choisir entre deux ou plusieurs chemins d'exécution (un choix entre deux ou plusieurs options), ou bien à répéter l'exécution d'un ensemble d'instructions. On appelle les instructions qui permettent ce genre d'exécution **les structures de contrôle**. Ces structures sont de deux types : Structures de contrôles conditionnelles et structures de contrôle répétitives (itératives).

II.4.4. Les structures de contrôle :

II.4.4.1. Les structures de contrôle conditionnelles : Ces structures sont utilisées pour décider ou pas de l'exécution d'un ou de plusieurs instructions en testant (vérifiant) une ou plusieurs condition. On en distingue deux types de tests : le test alternatif simple et double.

a. Le test alternatif simple : Ce test contient un seul bloc d'instructions. Selon une ou plusieurs conditions (expression logique). Si la condition est vraie, on exécute le bloc, sinon on ne l'exécute pas (c.à.d. soit on exécute l'instruction suivante dans le programme ou on sort du programme).

Syntaxe :

Algorithme	Programme (Pascal)

Si <condition> **alors**
<Instruction(s)> **;**
FinSi ;
	If <condition> **then**
	begin
	  <Instruction(s)> ;
	end ;

✓ **Remarque :** En Pascal, le bloc d'instructions à exécuter après un if (juste après then) **DOIT** être délimité par un *begin* et un *end* lorsque le nombre d'instructions est supérieur ou égal à 2.

- **Exemple :** Un algorithme et son programme qui déterminent si un nombre entier est pair.

```
Algorithme Pair;
Variables  a : entier;

Début
Écrire ('Donner la valeur de a: ');
Lire (a);
Si (a mod 2 = 0 ) alors
   écrire (a , ' est pair ');
FinSi ;
Fin.
```

```
Program Pair;
Var    a : integer;

Begin
Writeln ('Donner la valeur de a: ');
Readln (a);
If (a mod 2 = 0 ) then  writeln (a , ' est pair ');
   //On n'a pas mis de begin end après then car
    //il y a une seule instruction à exécuter
End.
```

b. Le test alternatif double : Ce test contient deux blocs d'instructions. Le premier bloc est exécuté lorsque la condition et vérifiée (vraie) et le second lorsque la condition n'est pas vérifiée (fausse).

Syntaxe :

Algorithme	Programme (Pascal)

Si <condition> **alors**
  <Instruction(s) 1>
Sinon <Instruction(s) 2> **;**
FinSi ;
	If <condition> **then**
	begin
	  <Instruction(s) 1> ;
	end
	Else
	begin
	  <Instruction(s) 2> ;
	end ;

✓ **Remarque :** En Pascal, L'instruction qui précède else ne doit pas contenir un ;

- **Exemple :** Un algorithme et son programme qui déterminent si un nombre entier est pair ou impair.

```
Algorithme PairImpair;
Variables  a : entier;

Début
Écrire ('Donner la valeur de a: ');
Lire (a);
Si (a mod 2 = 0 ) alors
   écrire (a , ' est pair ')
Sinon      écrire (a , ' est impair ');
FinSi ;
Fin.
```

```
Program PairImpair;
Var   a : integer;

Begin
Writeln ('Donner la valeur de a: ');
Readln (a);
If (a mod 2 = 0 ) then  writeln (a , ' est pair ')
Else   writeln (a , ' est impair ');
End. //Là aussi, on n'a pas mis de begin end après
 //if et else car les deux contiennent une seule
 //instruction.
```

II.4.4.2. Les structures de contrôle répétitives : Ces structures permettent de répéter un traitement un nombre fini de fois. Elles sont appelées " *boucles* " et sont de trois types : la boucle pour, la boucle tant-que et la boucle répéter.

a. **La boucle Pour "For":** Cette boucle permet de répéter l'exécution des instructions un nombre de fois prédéterminé c.à.d. le nombre de répétitions (itérations) est connu à l'avance.

Syntaxe :

Algorithme

```
Pour <compteur> ← <Vi> à <Vf> faire
    <Instruction(s)> ;
FinPour ;
```

Programme (Pascal)

```
For <compteur> := <vi> to <vf> do
begin
    <Instruction(s)> ;
end ;
```

- ***compteur*** est le compteur de la boucle, ***Vi*** et ***Vf*** sont les bornes inférieure et supérieure (valeurs initiale et finale).
- ***Vi*** et ***Vf*** sont des expressions ordinales (qui expriment l'ordre), du même type que la variable ***compteur***.
- ***Vi*** et ***Vf*** sont d'abord évaluées, puis ***compteur*** prend la valeur ***Vi***. (*) Si ***compteur*** <= ***Vf***, alors les instructions sont exécutées, puis ***compteur*** est incrémenté de 1, et on recommence depuis (*).

  - **Exemple :** Un algorithme et son programme qui déterminent si les nombres entiers entre 0 et n sont pairs ou impairs.

```
Algorithme PairImpairPour;
Variables  i,n : entier;

Début
Écrire ('Donner la valeur de n: ');
Lire (n);
Pour i ← 0 à n faire
   Si (i mod 2 = 0 ) alors
      écrire (i , ' est pair ')
   Sinon     écrire (i , ' est impair ');
   FinSi ;
FinPour ;
Fin.
```

```
Program PairImpairPour;
Var    i,n : integer;

Begin
Writeln ('Donner la valeur de n: ');
Readln (n);
For i := 0 to n do
 Begin
   If (i mod 2 = 0 ) then  writeln (i , ' est pair ')
   Else    writeln (i , ' est impair ');
 End ;
End.
```

✓ **Remarque :** Le langage Pascal n'est pas sensible à la casse, ses mots clés peuvent être écrits en minuscule, en majuscule ou un mélange des deux, ça ne change rien.