

# Chapitre 1

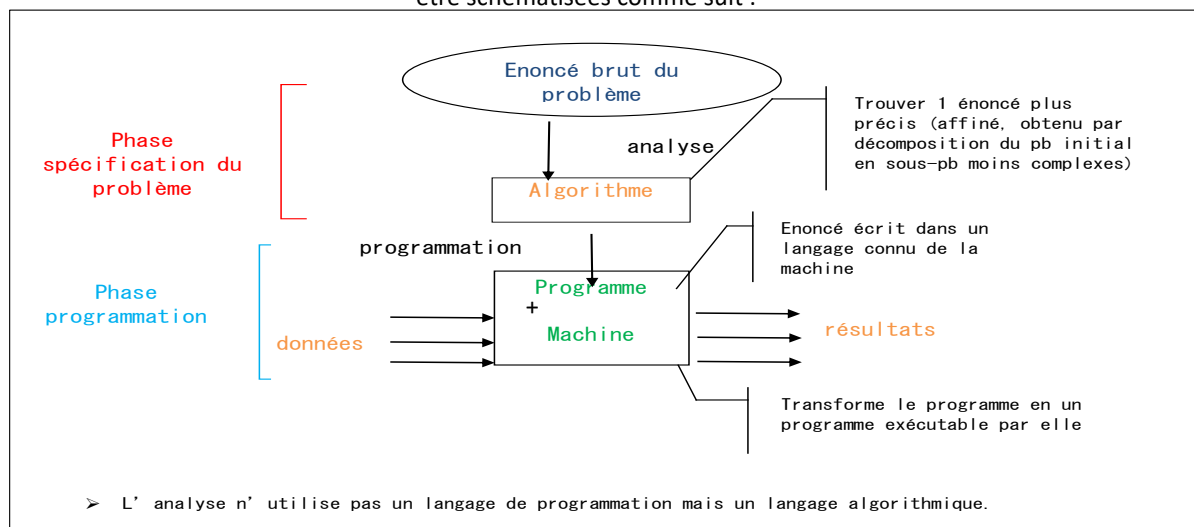
## Du problème au Programme en passant par l'Algorithme

L.ZERTAL

1

### Chapitre 1 Du Problème au Programme en passant par l'Algorithme

Soit un problème concret à résoudre dont les étapes peuvent être schématisées comme suit :



L.ZERTAL

2

## Chapitre 1 Du Problème au Programme en passant par l'Algorithme

### ➤ Analyse guidée par le(s) résultat(s)

**Résultat(s)  $\Rightarrow$  Donnée(s)**

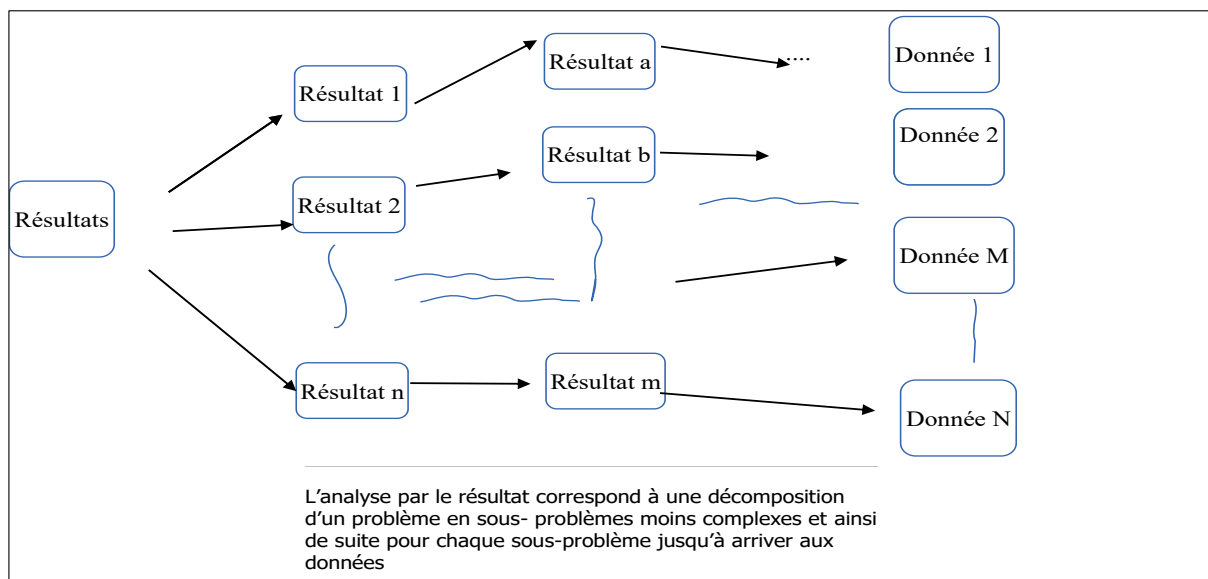
Décomposition

On part du ou des résultats à produire (correspondant au problème à résoudre) que l'on décompose en sous-résultats ou résultats intermédiaires jusqu'à arriver aux données du problème.

L.ZERTAL

3

## Chapitre 1 Du Problème au Programme en passant par l'Algorithme



L.ZERTAL

4

## Chapitre 1 Du Problème au Programme en passant par l'Algorithme

### ➤ Analyse guidée par la(es) donnée(s)

**Donnée(s)  $\Rightarrow$  Résultat(s)**

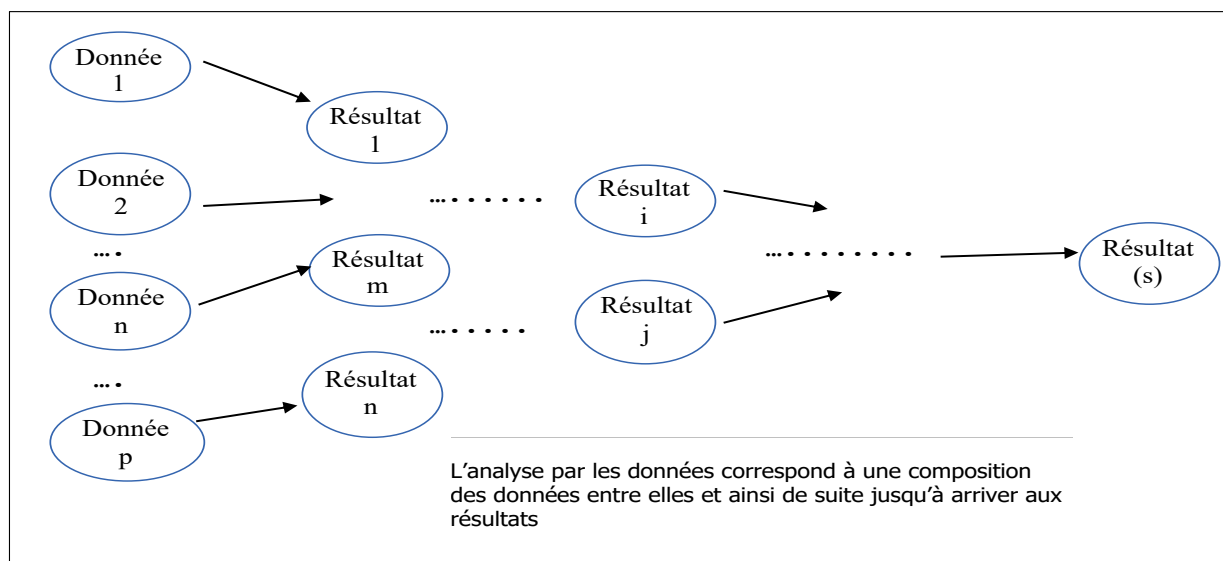
Composition

On part des données du problème que l'on compose entre elles pour obtenir des résultats intermédiaires jusqu'à arriver au résultat final.

L.ZERTAL

5

## Chapitre 1 Du Problème au Programme en passant par l'Algorithme



L.ZERTAL

6

## Chapitre 1 Du Problème au Programme en passant par l'Algorithme

### I.2) la notion d'algorithme

**Définition** (d'après Donald Erwin Knuth, un des pionniers de l'algorithmique) :

Il a donné une liste de cinq propriétés qui sont largement reconnues comme les pré-requis d'un algorithme:

- ✓ **Finitude**: "Un algorithme doit toujours se terminer après un nombre fini d'étapes."
- ✓ **Définition précise**: "Chaque étape d'un algorithme doit être définie précisément; les actions à transposer doivent être spécifiées rigoureusement et sans ambiguïté pour chaque cas."
- ✓ **Entrées**: "...des quantités (informations) qui lui sont données avant qu'un algorithme ne commence. Ces entrées sont prises dans un ensemble d'objets spécifié."
- ✓ **Sorties**: "...des quantités (informations) qui ont une relation spécifiée avec les entrées."
- ✓ **Rendement**: "... toutes les opérations que l'algorithme doit accomplir doivent être suffisamment basiques pour pouvoir être en principe réalisées dans une durée finie par un homme utilisant du papier et un crayon."

## Chapitre 1 Du Problème au Programme en passant par l'Algorithme

En d'autres mots :

Un algorithme un ensemble de règles (ou instructions ou actions) ayant les caractéristiques suivantes :

- Il doit être **fini** et se **terminer** après un nombre fini d'opérations
  - Il doit être **défini** et **précis** : chaque instruction est définie sans ambiguïté
  - Le champ d'application des données doit être précisé sans ambiguïté
  - Il doit posséder un **résultat**
  - Il doit être **effectif** : toutes les opérations doivent pouvoir être effectuée par un opérateur.
- Conclusion : un algorithme est le résultat d'une **Analyse** qui, après **traduction** dans un **Langage de Programmation**, peut être exécuté par une **Machine**.

## Chapitre 1 Du Problème au Programme en passant par l'Algorithme

### II Algorithme et méthode

#### II.1) Syntaxe (forme de l'algorithme)

Un algorithme se présente en 3 parties :

- **Instructions** : définitions des différentes étapes de la résolution
- **Ordonnement** : indique l'ordre selon lequel doivent se dérouler les différentes étapes
- **Lexique** : identifie les objets manipulés

Remarque : l'ordonnement est **obligatoire** dans une analyse par le **résultat** (et **facultatif** dans une analyse par les **données**)

Deux types d'analyse => deux types d'algorithmes.

## Chapitre 1 Du Problème au Programme en passant par l'Algorithme

#### II.2) Exemple : facture téléphonique (très simplifiée)

On veut établir une facture téléphonique

**Résultat** : Montant à payer

**Données** : Montant de l'abonnement, Prix d'une unité, Nombre d'unités consommées

## Chapitre 1 Du Problème au Programme en passant par l'Algorithme

### II.2.1) Analyse guidée par le(s) résultat(s)

Définitions (nom de l'algorithme)	Ord	Lexique
<b>Résultat</b> : Montant à payer	4	montant à payer (réel) : montant de la facture
Montant à payer = Prix conso + Montant Abonnement	3	prix conso (réel) : prix de la consommation
Prix conso = Nb unités * Prix unité	2	prix unité (réel) : prix d'une unité
<b>Données</b> : Montant abonnement, Prix unité, Nb d'unités	1	nb unités (entier) : nombre d'unités consommées
		montant abonnement (réel) : montant de l'abonnement

L'ordonnancement indique l'ordre dans lequel les actions doivent être faites.

Un problème est résolu au fur et à mesure qu'il se présente.

Une donnée n'est pas un problème.

## Chapitre 1 Du Problème au Programme en passant par l'Algorithme

### II.2.2) Analyse guidée par les données

Définitions (nom de l'algorithme)	Ord	Lexique
<b>Données</b> : Montant abonnement, Prix unité, Nb d'unités consommées		
Prix conso = Nb unités * Prix unité		
Montant à payer = Prix conso + Montant Abonnement		
<b>Résultat</b> : Montant à payer		

Les actions à effectuer sont déjà ordonnées.

**Attention : Il ne faut pas confondre ordre de décomposition avec ordre d'exécution.**

## Chapitre 1 Du Problème au Programme en passant par l'Algorithme

### II.3) Notion de variable

**Définition** : une variable est un nom qui permet d'identifier un objet manipulé dans un algorithme. Concrètement, elle permet d'identifier un emplacement mémoire de stockage.

L'objet possède une valeur qui peut être modifiée dans l'algorithme.

La valeur de l'objet est la valeur (contenu) de la variable.

## Chapitre 1 Du Problème au Programme en passant par l'Algorithme

### II.4) Une première instruction : l'affectation

**Définition** : l'affectation est l'instruction qui permet d'associer une valeur à une variable.

- Cette valeur peut être une valeur **constante** ou une valeur **calculée**.
- Symbole utilisé pour représenter cette instruction :  $\leftarrow$  (flèche de droite à gauche)

**Nom Variable  $\leftarrow$  valeur**

**Sens** : la variable prend la valeur (ou aura pour valeur); on affecte une valeur à la variable

- Le nom d'une variable est une chaîne alphanumérique (**lettre, chiffre, \_**), sans espaces ni lettres accentuées.

**Exemple** : a, b, prix\_conso, px\_unite, AB\_3, titi45, ....

prix\_conso  $\leftarrow$  px\_unit \* nbunite

a  $\leftarrow$  8 : on affecte à la variable a la valeur constante 8 : a *prend pour* valeur 8

a  $\leftarrow$  b : (variable réceptrice  $\leftarrow$  variable émettrice) : a *prend pour valeur* la valeur de b

**Une variable peut changer de valeur.**

Dans un lexique, définir a comme variable ne signifie pas que a contient une valeur définie.

## Chapitre 1 Du Problème au Programme en passant par l'Algorithme

Notion de constante : On peut associer un nom à une valeur constante.

Exemple :

- ❑ on associe à la constante **3,14** (*le rapport entre la circonférence d'un cercle et son diamètre*) le nom **pi**
- ❑ si **a** désigne la valeur **8**, on ne peut plus lui associer d'autres valeurs

Remarque : Le choix des identifiants de variables/constantes doit être suffisamment parlant pour indiquer la nature de l'objet.

## Chapitre 1 Du Problème au Programme en passant par l'Algorithme

### II.5) Notion de type

- La forme du contenu d'une variable est définie par un **type**.
- A toute variable est associé un type. Idem pour la notion de constante.

Les types permettent de définir :

- L'ensemble des **valeurs** que pourra prendre la variable (domaine de définition)
- L'ensemble des **opérations** applicables à la variable
- La **forme** de représentation (ou codage) en mémoire de l'ordinateur



## Chapitre 1 Du Problème au Programme en passant par l'Algorithme

### Les types simples :

- ✓ Entier, Réel (numériques)
- ✓ Caractère, Chaîne (de caractères)

#### Exemple :


a ← 5                    | | a (entier)  
b ← 'e'                 | | b (caractère)

#### Remarque :

on différencie le nom d'une variable d'une valeur de type caractère en mettant la valeur entre simples guillemets ''.

Un autre intérêt de l'utilisation de types est de pouvoir contrôler certaines opérations.

Exemple : si a est définie comme une variable de type *entier* :

a ← 5,4    | | a (entier)        #ERREUR !#   
(on affecte à a le réel 5,4)

Et

a ← 5       | |                #Correct#  
(on affecte à a l'entier 5)

## Chapitre 1 Du Problème au Programme en passant par l'Algorithme

### Définitions dans le lexique :

*nom\_variable*(type) : commentaire explicatif

*nom\_constant*(cste/type=valeur) : commentaire explicatif

#### Exemples :

pi(cste/réel=3,14) : la constante de cercle

nb\_an(entier) : nombre d'années

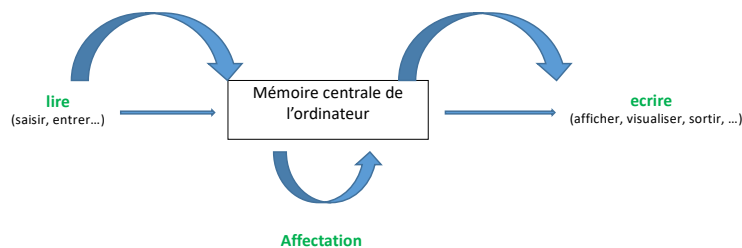
## Chapitre 1 Du Problème au Programme en passant par l'Algorithme

### II.6) Opérations de communication (de l'algorithme) avec l'extérieur utilisant des variables.

#### But :

- permettre de récupérer les données (par exemple à partir d'un clavier) ⇒ **lire**.
- permettre d'afficher la valeur d'un résultat (par exemple sur un écran) ⇒ **écrire**.

## Chapitre 1 Du Problème au Programme en passant par l'Algorithme



- ❑ **lire, affectation** : actions qui modifient le contenu de la variable
- ❑ **écrire** : ne modifie pas le contenu de la variable

## Chapitre 1 Du Problème au Programme en passant par l'Algorithme

**Syntaxe** :    lire (liste\_arguments)        écrire (liste\_arguments)

**Exemples** :        lire(a)  
                         écrire(a)  
                         lire(a,b)  
                         écrire('e', total)  
                         écrire(a,b,c)  
                         écrire (5)  
                         écrire("bonjour", somme)

## Chapitre 1 Du Problème au Programme en passant par l'Algorithme

### Exemple d'algorithme écrit selon les concepts

#### Avec une analyse guidée par le résultat

algo Facture_Telephone	Ord	Lexique
écrire ("Montant facture = ", montant)	7	mt_abt (reel) : montant de l'abonnement
montant ← px_conso + mt_abt	6	montant (reel) : somme facturée
px_conso ← px_unite * nb_unites	5	px_unite (reel) : prix d'une unité téléphonique
écrire ("Saisir le montant de l'abonnement : ")	1	nb_unites (entier) : nombre d'unités consommées
lire (mt_abt)	2	px_conso (reel) : prix de la consommation
écrire ("Saisir le prix d'une unité et le nombre d'unités :")	3	
lire (px_unite, nb_unites)	4	

## Chapitre 1 Du Problème au Programme en passant par l'Algorithme

### III Définitions conditionnelles (ou primitives conditionnelles)

**Exemple** : calcul de la paie (simplifié)

**Données** : nom, nb heures, salaire horaire, taux de retenue, montant plafond, prime

**Résultats** : nom, salaire brut, salaire net

## Chapitre 1 Du Problème au Programme en passant par l'Algorithme

### Analyse par le résultat

algo Calcul_Paie	Ord	Lexique
<b>ecrire</b> (nom)	7	nom (chaîne) : nom employé
<b>ecrire</b> (salbrut,salnet)	8	salbrut (réel) : salaire brut
<b>lire</b> (nom)	1	salnet (réel) : salaire net
salbrut $\leftarrow$ nbh*salhor	4	nbh (réel) : nombre d'heures travaillées
<b>lire</b> (nbh, salhor)	2	salhor (réel) : salaire horaire
salnet $\leftarrow$ salbrut – rss + prime	6	rss,prime (réel) : respectivement retenue sociale et prime
<b>si</b> salbrut $\leq$ plafond <b>alors</b>	5	plafond (cste/réel=1000) : montant plafonné de la sécurité sociale
rss $\leftarrow$ salbrut * taux1		taux1 (cste/réel= 1,5) : 1er taux
<b>sinon</b>		taux2 (cste/réel= 2,5) : 2ème taux
rss $\leftarrow$ (salbrut-plafond)*taux2+ plafond*taux1		
<b>fsi</b>		
<b>lire</b> (prime)	3	

## Chapitre 1 Du Problème au Programme en passant par l'Algorithme

### Syntaxe :

#### ➤ conditionnelle simple

```
si condition vérifiée alors  
|  
| instructions (lire, écrire, ← , si)  
|  
fsi
```

## Chapitre 1 Du Problème au Programme en passant par l'Algorithme

### Syntaxe :

#### ➤ conditionnelle avec une alternative

```
si condition vérifiée alors  
|  
| instructions1 (ensemble d'instructions)  
|  
sinon  
|  
| instructions2 (autre ensemble d'instructions)  
|  
fsi
```

## Chapitre 1 Du Problème au Programme en passant par l'Algorithme

### Syntaxe :

#### ➤ conditionnelle avec alternatives multiples

```
si condition1 vérifiée alors  
| instructions1  
sinonsi condition2 vérifiée alors  
| instructions2  
sinonsi condition3 vérifiée alors  
| instructions3  
| ...  
sinonsi conditionN vérifiée alors  
| instructionsN  
sinon  
| instructions  
fsi
```

## Chapitre 1 Du Problème au Programme en passant par l'Algorithme

### Syntaxe :

#### ➤ conditionnelle avec alternatives multiples (autre écriture)

```
si condition1 vérifiée alors  
| instructions1  
sinon  
| si condition2 vérifiée alors  
| | instructions2  
| sinon  
| | si condition3 vérifiée alors  
| | | instructions3  
| | | .....  
| | fsi  
| fsi  
fsi
```

## Chapitre 1 Du Problème au Programme en passant par l'Algorithme

### IV le choix multiple

Lorsqu'une variable peut prendre différentes valeurs et que chaque valeur détermine un traitement différent, on utilise l'instruction de **choix multiple** pour définir le traitement à faire.

#### Exemple :

Soit la variable  $a$  (*entier*)

- si  $a = 2$  : on exécute instructions1
- si  $a \in \{3,4\}$  : on exécute instructions2 ( $a=3$  ou  $a=4$ )
- si  $a \in [6..20]$  : on exécute instructions3 ( $a$  prend sa valeur dans l'intervalle)

## Chapitre 1 Du Problème au Programme en passant par l'Algorithme

On l'exprime comme suit :

**cas** a **parmi**

- | 2 : écrire('ok')
- | 3,4 :  $b \leftarrow 8$
- | 6..20 : lire( $b$ )
- | **autres** : écrire( " pas ok ")

**fincas** (ou **fcas**)

équivalent à :

```
si ( $a = 2$ ) alors  
| ...  
sinonsi ... alors  
| ...  
sinonsi ... alors  
| ...  
sinon  
| ...  
fsi
```

## Chapitre 1 Du Problème au Programme en passant par l'Algorithme

### Syntaxe :

**cas** variable **parmi**

	valeur_1 : instructions1	(peut être optionnel)
	valeur_2, valeur_3, .... : instructions2	(peut être optionnel)
	valeur_4..valeur_5 : instructions3	(peut être optionnel)
	<b>autres</b> : instructions	(peut être optionnel)

**fincas** (ou **fcas**)

Remarque : la variable doit être de type scalaire (cf chapitre suivant)