

Algorithmique

1

Université Internationale
de Casablanca

A.AHMADI

2018/2019

I- Introduction

2

- **Algorithmique** : néologisme introduit en 1957. Il prend sa racine de l'ancien mot « Algorisme » qui définit le processus de faire l'arithmétique en utilisant les chiffres arabes ;
- Le mot "**Algorisme**" provient du nom du célèbre mathématicien Perse **Al-Khawarismi**, auteur du livre "Hissab Al jabr wa Al Mokabala" ;
- **Algorithme** : description du schéma de réalisation d'un événement à l'aide d'un répertoire fini d'actions élémentaires, réalisables à durée limitée dans le temps ;
- Un algorithme est destiné à un processeur qui peut l'exécuter, c'est à dire, réaliser l'enchaînement des opérations que l'algorithme lui décrit ;
- Pour qu'un algorithme soit exécutable par un processeur, il doit être écrit dans un Langage compréhensible par ce processeur ;
- Si le processeur est un ordinateur, l'algorithme s'appelle un programme ;
- La notion d'algorithme n'est pas spécifique à l'informatique (voir exemples ci-dessus).

- Dans les années 50, le mot algorithme était associé à "**l'algorithme d'Euclide**", c'est à dire, le processus de trouver le plus grand diviseur commun de deux entiers :

principe : $a = b * q + r$, $0 < r < b$ (Division Euclidienne de a par b).

On a $PGCD(a, b) = PGCD(b, r)$. Donc, on cherche le dernier reste non nul. On affine cette méthode comme suit :

Etape 1 : trouver le reste r de la division euclidienne de a par b ;

Etape 2 : **Si** $r = 0$, **FIN** de l'algorithme et $PGCD(a, b) = b$.

Sinon Faire la permutation suivante : $a \leftarrow b$ et $b \leftarrow r$ et retourner à l'étape 1.

Etape 3 : **FIN** de l'algorithme.

L'étape 3 sert à dire explicitement qu'on a bien terminé l'écriture de l'algorithme.

1- Définition du problème

Il s'agit de bien définir le problème, de manière à éviter toute ambiguïté, et souligner toutes ses spécifications et les conditions nécessaires à sa réalisation. Il s'agit de répondre aux questions suivantes :

- Quelles sont les données du problème? comment devront être introduites?
- Quelles sont les erreurs susceptibles de se produire ? et quelle est l'action à prendre dans chaque cas ?
- Quels sont les résultats à fournir ? et comment les présenter ?
- Quels sont les traitements (et les techniques) à effectuer?

2- Structure générale

Dans cette étape, on pose les grandes lignes de l'algorithme et on procède à une décomposition du problème en une suite de sous-problèmes plus simples à résoudre. C'est le principe de "**diviser pour régner**".

3- Développement

Il s'agit d'élaborer l'algorithme correspondant à chaque sous-problème de l'étape précédente et l'algorithme global du problème.

4- Validation de l'algorithme

Dans cette étape, on vérifie si l'algorithme répond bien à toutes les spécifications du problème définies à l'étape 1. Une erreur ou un cas non traité pourra remettre en cause la solution proposée.

5- Codage

On traduit l'algorithme, validé, en un programme dans un langage de programmation adéquat, disponible et maîtrisé, pour le rendre exécutable par la machine

6- Mise au point

On corrige les erreurs commises à l'étape précédente et signalées par le compilateur .

Le compilateur est un programme qui, ayant reçu un autre programme écrit dans un langage évolué, l'analyse(détermine les erreurs de syntaxe, mais pas de conception et de logique) et le traduit en un langage machine; il traduit donc le source en un exécutable.

1- Variable

- C'est un objet qui peut prendre une ou plusieurs valeurs durant son existence dans un programme donné. Elle est caractérisée par les trois éléments suivants :
 - Son nom (identification ou désignation de l'emplacement mémoire de la variable) qui la différencie des autres et permet l'accès à sa valeur.
 - Son type, qui spécifie le domaine de valeurs que peut prendre la variable.
 - sa valeur (c'est le contenu actuel de l'emplacement mémoire de la variable).
- Le type de variable spécifie implicitement le genre d'opérations permises. Par exemple sur les entiers on utilise l'addition, la multiplication, etc.

2-Affectation

- C'est l'attribution d'une valeur à une variable. Notation : \leftarrow ou $=$.
- On affecte une variable par une expression du même type. Il faut noter qu'une expression du style : $x = expression$, ne se lit que dans un seul sens : la variable x prend la valeur de l'expression à droite

Exemples d'affectation :

- $X \leftarrow -2.56$ (ou $X = -2.56$), signifie que la variable X reçoit la valeur -2.56 (sa valeur actuelle).
Son type est réel ;
 - $y = VRAI$. Son type est booléen {vrai, faux} ;
 - $Z = 'Bonjour'$. Son type est chaîne de caractères.
- Il existe trois formes d'affectation :
 - variable \leftarrow constante. (voir les 3 exemples précités) ;
 - variable \leftarrow variable (ex. $T = Z$ et donc la valeur de la variable T est 'Bonjour') ;
 - Variable \leftarrow expression composée, arithmétique ou logique (ex. $M = (x^2 + 1)/2$).

NB : Faire attention à la concordance des types de variables pendant l'affectation. Supposons que x est de type *réel* et que n est de type *entier*. L'affectation $x = n$ est valide alors que $n = x$ ne l'est pas.

3-Lecture

- C'est une opération qui consiste à attribuer une valeur de l'extérieur, par une unité d'entrée (ex. clavier, capteur de grandeurs physique, etc.), à une variable ;

Notation : $LIRE(x)$ ou $LIRE(x, y, \dots)$.

- A la différence de l'affectation qui se passe au niveau de la mémoire (interne), la lecture utilise un périphérique d'entrée (externe) ;

4- Ecriture

- Son rôle est de faire sortir l'information à l'extérieur par un périphérique de sortie (ex. imprimante, écran, fichier, etc.) ;
- Cette information peut être numérique, chaîne de caractères, ...

Notation : $ECRIRE(\text{information1}, \text{information2}, \dots)$.

- $ECRIRE(x)$: affiche la valeur de la variable x ;

- $ECRIRE('Salut')$: affiche ou écrit le message : *Salut* (le texte doit être toujours entre deux apostrophes).

Exemple : si la variable x est de type *entier* et qu'elle contient la valeur 13, alors l'instruction $ECRIRE('x = ', x)$ a pour effet d'afficher $x = 13$.

Dans la programmation classique (structurée), on distingue trois structures :

- La séquence;
- La sélection;
- La répétition.

1- La séquence

On dit aussi instruction composée : c'est une suite d'instructions (arithmétiques ou autres), délimitée par les mots-clés, **DEBUT** et **FIN**, qui s'exécutent séquentiellement (c.à.d., l'une après l'autre).

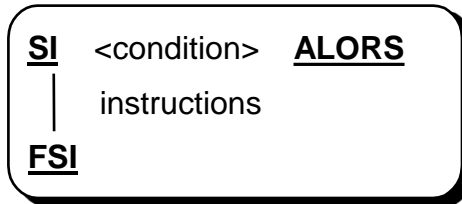
Exemple : L'algorithme suivant lit les variables x et y , en fait la somme z et affiche le résultat :

```
x, y : entier ;  
DEBUT  
  | lire(x) ;  
  | lire(y) ;  
  | z ← x + y ;  
  | écrire('z =', z) ;  
FIN
```

2- La sélection

- Elle comporte les instructions conditionnelles : c'est à dire qu'une instruction (simple ou composée) ne sera exécutée qu'après remplissage (satisfaction) de certaines conditions ;
- Dans ce qui suit **<condition>** a comme type les booléens {*vrai, faux*}; c'est une expression construite à partir des variables booléennes et des connecteurs (opérateurs) logiques « **et** », « **ou** », « **non** ».

Forme 1 :

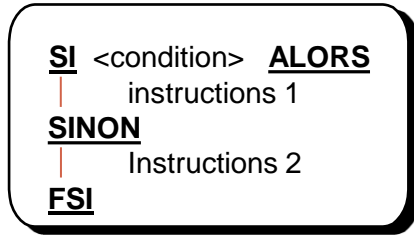


- Si la condition <condition> est vraie, alors exécuter les instructions; sinon on ne fait rien. On passe à l'exécution de l'instruction suivante (qui est juste après FSI).

Exemple :

```
x: entier ;
DEBUT
| lire(x);
| SI (x ≥ 0) ALORS
| | écrire('la valeur de x est positive' );
| FSI
FIN
```

Forme 2 :



Si la condition <condition> est vraie, alors exécuter les instructions 1; sinon exécuter les instructions 2 ;

Exemple : L'algorithme suivant compare les variables x et y . La variable z contient la différence $x - y$ si x est supérieure ou égale à y , sinon z contient $y - x$. Ensuite affiche la valeur absolue de $x - y$.

```
 $x, y, z$  : entier ;  
DEBUT  
| lire( $x, y$ ) ;  
| SI ( $x \geq y$ ) ALORS  
| |  $z = x - y$  ;  
| SINON  
| |  $z = y - x$  ;  
| FSI  
| écrire('la valeur absolue de  $x-y$  est : ',  $z$ ) ;  
FIN
```

Remarque :

- L'instruction après ALORS ou SINON est quelconque et peut être une instruction SI. Dans ce cas, on dit qu'on a des SI imbriqués.

Exemple : l'instruction ci-dessous affecte à la variable réelle x sa moitié si a et b sont nulles. si a = 0, x est doublée.

```
SI (a = 0) ALORS  
| SI (b = 0) ALORS  
| | x = x/2 ;  
| FSI  
SINON  
| x = 2*x ;  
FSI
```

3- La répétition

- Permet de répéter (ré-exécuter) une ou plusieurs instructions (quelconques) tant qu'une condition est vérifiée ;
- Elle est réalisée par des boucles de trois formes.

Forme1 :

```
TANT QUE <condition> FAIRE  
    instructions  
FTANT QUE
```

- On évalue d'abord la condition < condition> ; si elle est vraie on exécute les instructions «instructions » et on retourne pour réévaluer la condition. Dès que la condition est fausse on exécute l'instruction qui suit la boucle TANT QUE ... FAIRE...

Exemple :

```
x, s, n : entier ;  
DEBUT  
  lire (n) ;  
  x = 0 ; s = 0 ;  
  TANT QUE (x < n) FAIRE  
    x = x + 1 ;  
    s = s + x ;  
  FTANT  
  écrire(s) ;  
FIN
```

- L'algorithme ci-dessus calcule la somme des entiers de 1 à n. On observe qu'il faut initialiser les variables x et s à 0 (au début du programme les variables contiennent des valeurs quelconques) ;
- La variable x servant de compteur, alors que la variable s contiendra à la fin du programme la somme cherchée.

Remarque :

La condition peut ne pas être remplie dès le départ. Dans ce cas aucune instruction, à l'intérieur de la boucle, ne sera exécutée.

Chap. 1

Algorithmique

IV- Instructions de contrôle

15

Exemple : Dans l'algorithme suivant la boucle n'a pas d'effet sur la variable entière y vu que la condition de la boucle est toujours fausse. Donc y gardera à la fin la valeur 0.

```
x : booléen ;
y : entier ;
DEBUT
| x = FAUX;
| y=0;
| TANT QUE ( x = VRAI ) FAIRE
| | y = y + 1;
| FTANT
| | écrire(y);
FIN
```

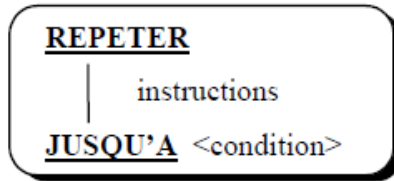
NB : Il faut faire attention aux boucles ouvertes :
condition d'arrêt toujours vraie !

Exemple :

L'instruction "écrire(y)" ne sera jamais exécutée.

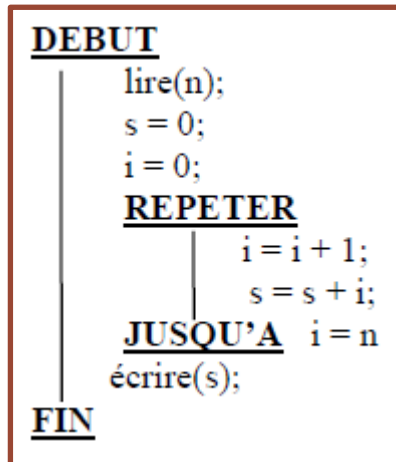
```
x : booléen ;
y : entier ;
DEBUT
| x = FAUX;
| y=0;
| TANT QUE non (x) FAIRE
| | y = y + 1 ;
| FTANT
| | écrire(y);
FIN
```

Forme2 :



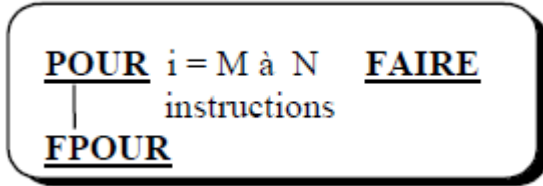
- On répète le traitement (instructions) jusqu'à ce que la condition <condition> soit vraie.

Exemple : Une autre version pour calculer la somme des entiers de 1 à n .



- A la différence de la première forme, les instructions dans la 2^{ème} forme seront exécutées au moins une seule fois, même si la condition d'arrêt est vérifiée dès le départ.
- Faire attention à l'ordre des instructions : $(i = i + 1; s = s + i;)$ est différente de $(s = s + i; i = i + 1;)$. la première séquence donne comme résultat $s = 1 + 2 + \dots + n$ et la 2^{ème} calcule $1 + 2 + \dots + n - 1$.

Forme3 :



- On exécute les instructions spécifiées (N-M+1) fois, la variable i sert de compteur.
- On utilise cette forme lorsqu'on connaît le nombre de fois qu'on exécutera les instructions en question.

Exemple : La version suivante, calculant la somme des entiers de 1 à n, est meilleure que les précédentes vu qu'on connaît à priori le nombre d'étapes de l'exécution de l'itération.

```
i, n, s : entier ;
DEBUT
|
s = 0;
lire(n);
POUR i = 1 à n FAIRE
|
s = s + i;
FPOUR
|
écrire(s);
FIN
```

Remarque :

L'exemple de la somme des entiers de 1 à n ayant servi pour illustration. Bien entendu l'algorithme suivant fait le même travail :

```
n, s : entier ;
```

```
DEBUT
```

```
    lire(n) ;
```

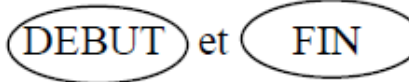
```
    s := n*(n+1)/2 ;
```

```
    écrire(s) ;
```

```
FIN
```

- L'organigramme est une représentation graphique d'un algorithme. Il permet de donner, à priori, une idée claire sur l'enchaînement, la logique et le déroulement d'un algorithme.

Formalisme :

- L'ellipse représente le DEBUT et la FIN d'un algorithme : 

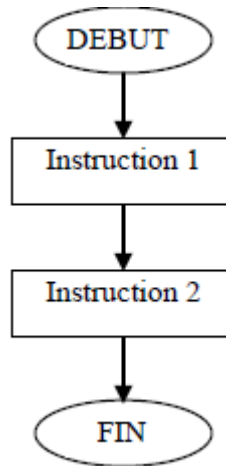
- Une opération ou un traitement est représenté par un rectangle : 

- Une condition est mise dans un losange : 

- La flèche désigne le sens de l'exécution de l'algorithme : 

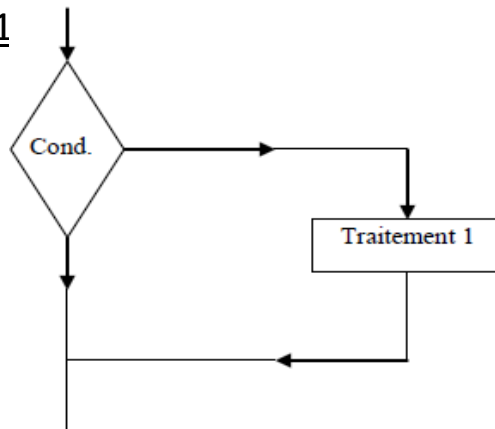
Ainsi les trois structures peuvent être représentées comme suit :

1- La séquence

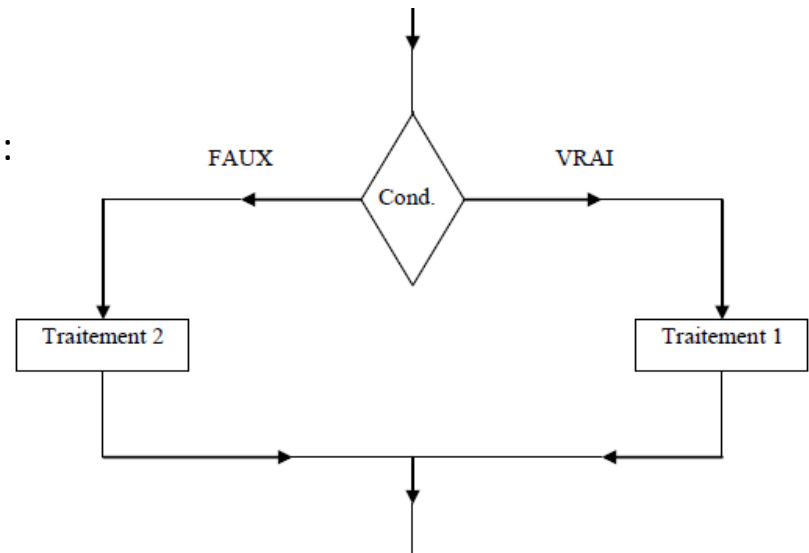


2- La sélection

Forme 1

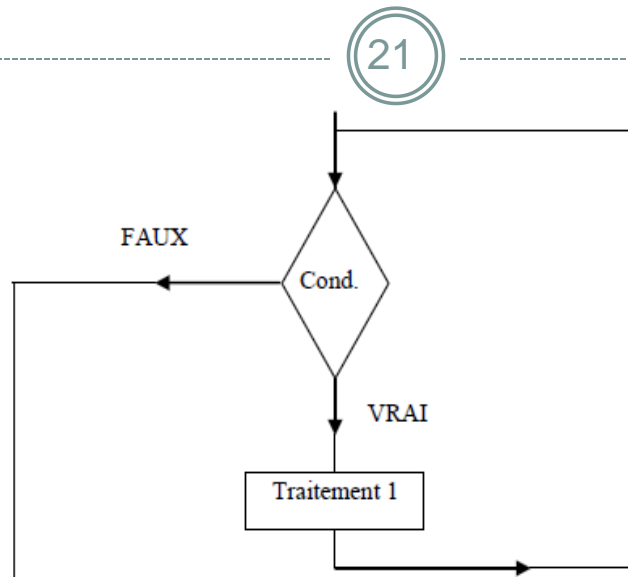


Forme 2 :

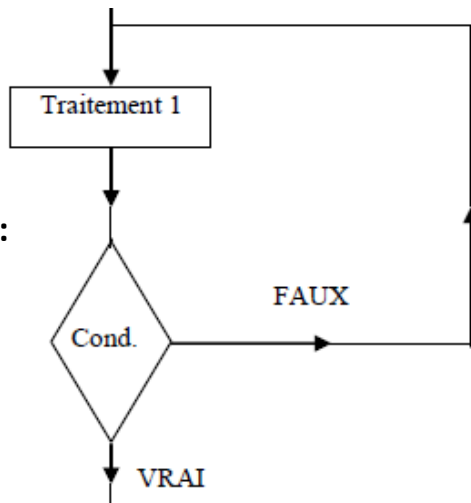


3- La répétition

Forme 1 :



Forme 2 :



Forme 3 :

