

**Ministère de l'Enseignement Supérieur et de la Recherche Scientifique**  
**Université des sciences et de la technologie**  
**Mohamed Boudiaf (Oran)**  
**Faculté : Génie Mécanique**  
**Département : Génie Maritime**



**SUPPORT DE COURS EN**  
**INFORMATIQUE 02**  
**Algorithmique et programmation**

**Cours**  
**Travaux dirigés**  
**Travaux pratiques**

Par Dr. Nateche tahar

Année Universitaire 2016-2017

## Table des Matières

Introduction Générale.....	1
<b>Leçon 1 : Généralités sur l'Algorithmique</b>	
1.1 Ordinateur et programmation.....	2
1.2 Définitions et unités de mesure.....	2
1.3 Algorithmes .....	2
1.4 Langage de programmation.....	4
1.5 Structure d'un fichier Fortran.....	5
1.6 Le Codage .....	6
1.7 Les variables et les constantes.....	7
1.8 Opérandes et operateurs.....	12
1.9 L'instruction d'affectation.....	13
1.10 Les notions de lecture et d'écriture.....	14
Travaux Dirigés 1- Généralités sur l'Algorithmique.....	15
Solutions des exercices .....	16
Travaux Pratiques 1- Généralités sur l'Algorithmique.....	19
Solutions des exercices.....	21
<b>Leçon 2 : Les opérations de lecture et d'écriture</b>	
2.1 Introduction.....	23
2.2 Syntaxe générale.....	23
2.3 Entrées-sorties standards : Ecriture en format libre.....	24
2.4 Entrées-sorties standards : Lecture en format libre.....	24
2.4 Les formats.....	25
2.5 Les fichiers.....	26
Travaux Dirigés 2- Les opérations de lecture et d'écriture .....	28
Solutions des exercices .....	29
Travaux Pratiques 2- Les opérations de lecture et d'écriture .....	31
Solutions des exercices.....	32
<b>Leçon 3 : Les structures itératives</b>	
3.1 Définition.....	34
3.2 La boucle « POUR ».....	34
3.3 La boucle TANT QUE.....	35
3.4 La boucle REPETER ... JUSQUA ... ..	37
Travaux Dirigés 3- Les structures itératives .....	39
Solutions des exercices .....	40
Travaux Pratiques 3- Les structures itératives.....	43
Solutions des exercices.....	44

<b>Leçon 4 : Structures de contrôle conditionnel</b>	
4.1 Introduction.....	46
4.2 Expression logique.....	46
4.3 Evaluation d'une expression logique.....	46
4.4 Tableaux d'évaluations.....	47
4.4 Test alternatif simple.....	48
4.5 Test alternatif double.....	49
4.6 TESTS IMBRIQUES.....	52
Travaux Dirigés 4- Structures de contrôle conditionnel .....	55
Solutions des exercices .....	56
Travaux Pratiques 4- Structures de contrôle conditionnel .....	60
Solutions des exercices.....	61
<b>Leçon 5: Les tableaux</b>	
5.1 Définition.....	65
5.2 Déclaration des tableaux.....	65
5.3 Terminologie des tableaux.....	66
5.4 Manipulation d'un tableau.....	66
5.5 Tri d'un tableau.....	68
Travaux Dirigés 5- Les tableaux .....	72
Solutions des exercices .....	74
Travaux Pratiques 5- Les tableaux .....	77
Solutions des exercices.....	78

# Introduction

## Introduction

Ce polycopie est le fruit d'une expérience dans le domaine de l'algorithmique et de la programmation. Il constitue un support de cours pour des étudiants n'ayant aucune connaissance en programmation. Il est aussi destiné à des étudiants ayant déjà une première expérience en programmation et qui veulent connaître davantage sur l'art de la programmation.

On trouvera l'ensemble des questions posées aux différents examens écrits depuis que le cours d'Informatique est devenu obligatoire pour la majorité des sections de la Faculté de génie mécanique. Les exercices proprement dits sont ceux repris dans l'application "Le langage Fortran" qui se trouve à disposition sur les ordinateurs de la salle de travaux pratiques. Pour ces exercices, nous donnons tout d'abord l'ensemble des énoncés pour permettre aux étudiant(e)s d'y réfléchir, de tenter de les résoudre sans être influencés par la lecture des solutions proposées. Toutefois, pour faciliter la lecture, nous redonnons l'énoncé avant chaque solution; celle-ci reprend non seulement l'algorithme mais aussi le programme Fortran correspondant avec éventuellement une ou des variantes et également les données nécessaires à son exécution. Toutes les solutions proposées ne sont pas commentées autant que nous l'aurions voulu! On soumettra toutes les solutions proposées à une critique attentive. Nous accueillerons volontiers les remarques, corrections que les lecteurs voudront bien formuler.

La première partie de la polycopie traite les notions fondamentales de l'algorithmique : types de base, instructions simples, etc. La seconde partie est consacrée aux instructions de lecture et d'écriture et les formats de conversion des données. La deuxième et la troisième partie sont consacrées aux structures de contrôle itératives et conditionnelles. Elles sont structurées de manière à fournir à l'étudiant les bases de la programmation en Fortran afin que ce dernier puisse maîtriser pour les applications au calcul scientifique. Les notions des tableaux, fait l'objet de la dernière partie, elle constitue l'atout important de la programmation en Fortran avec l'introduction des notations matricielles et des fonctions intrinsèques manipulant les tableaux multidimensionnels.

Pour terminer, la polycopie contient environ une centaine d'exercices souvent conçus comme une application du cours à des situations de la vie professionnelle. La solution proposée à chaque exercice n'est pas unique et dans plusieurs cas elle n'est optimale car on a toujours privilégié l'apport pédagogique et la simplicité.

## Leçon 1 : Généralités sur l'Algorithmique

## Leçon 1 : Généralités sur l'Algorithmique

### *Objectifs*

- Connaître le vocabulaire de base en programmation
- Comprendre la démarche de programmation

### 1.1 Ordinateur et programmation

L'informatique intervient aujourd'hui dans de nombreux secteurs d'activité. Parmi les applications courantes on peut citer la bureautique, la gestion, le calcul scientifique, la communication, l'accès à des ressources d'information (au travers d'internet en particulier), le multimédia, les jeux etc. Ces applications ne sont possibles que grâce à un ordinateur. Cependant, l'ordinateur seul ne suffit pas. Pour chaque application, il est nécessaire de lui fournir un logiciel (ou programme) adapté. La programmation est donc une activité fondamentale en informatique. La programmation peut être vue comme l'art de déterminer un algorithme (une démarche) pour résoudre un problème et d'exprimer cet algorithme au moyen d'un langage de programmation.

### 1.2 Définitions et unités de mesure

Un bit (binary digit) est un élément binaire. Sa valeur est donc 0 ou 1. Un octet (ou byte) est un ensemble de 8 bits. Les longueurs couramment utilisées sont des ensembles de 16, 32 ou 64 bits. Un kilo-octet (abréviation : Ko) correspond à 1024 octets, soit 8192 bits. Un méga-octet (Mo) correspond à 1024 Ko, soit 8192 Ko. Un giga-octet (Go) est un ensemble de 1024 Mo, soit 8192 Mo. Ces unités de mesures sont fréquemment utilisées pour indiquer des tailles (ou capacités) de mémoires.

### 1.3 Algorithmes

#### 1.3.1 Définition

L'algorithmique est un terme d'origine arabe, composé d'une suite d'instructions élémentaires, qui une fois exécutée correctement, conduit à un résultat donné.

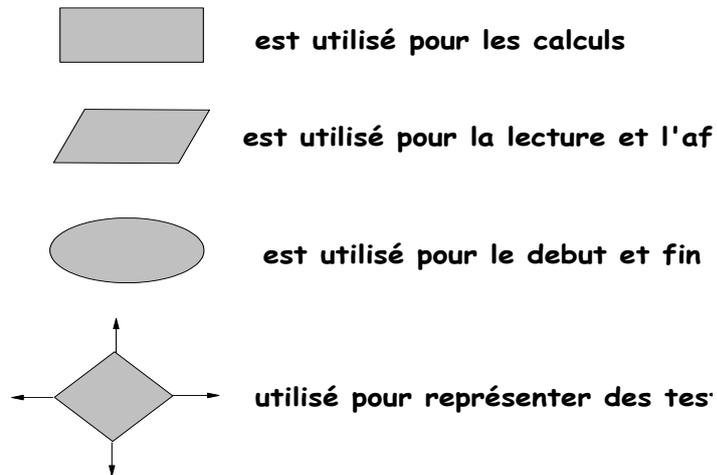
#### 1.3.2 Représentation graphique ou organigramme

La représentation graphique permet une lecture aisée des algorithmes mais présente toutefois l'inconvénient de consommer une place importante. Les opérations dans un organigramme sont représentées par les symboles dont les formes sont normalisées. Ces

## Leçon 1 : Généralités sur l'Algorithmique

---

symboles sont reliés entre eux par des lignes fléchées qui indiquent le chemin. C'est ainsi qu'on a:



### Exemple :

Créer un algorithme pour calculer la moyenne de 3 notes.

### Solution :

Algo moyenne

Variables note, moyenne : entier

Début

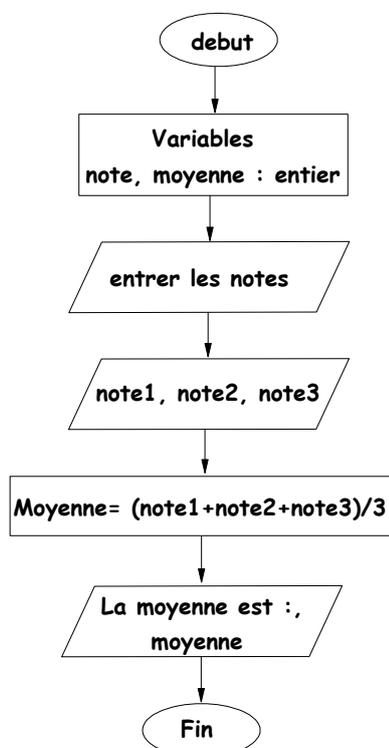
Ecrire (entrer les notes)

Lire (note1, note2, note3)

$Moyenne = (note1 + note2 + note3) / 3$

Ecrire (la moyenne est :, moyenne)

Fin



### 1.3.3 Les étapes de résolution d'un problème

1. Comprendre l'énoncé du problème
2. Décomposer le problème en sous-problèmes plus simple à résoudre
3. Associer à chaque sous problème, une spécification :
  - Les données nécessaires
  - Les données résultantes
  - La démarche à suivre pour arriver au résultat en partant d'un ensemble de données.
4. Elaboration d'un algorithme.

### 1.4 Langage de programmation

Il existe de nombreux langages de programmation : C, C++, Java, Basic, Pascal, Fortran, ... Le langage Fortran est utilisé dans ce cours en raison de son caractère pédagogique.

#### Exemple :

Ecrire et exécuter le programme fortran qui demande à l'utilisateur d'entrer la largeur et la longueur et afficher la surface (S) et le périmètre (P) d'un rectangle

#### Solution :

##### Algorithme :

```
Algo rectangle
Début
Variables largeur, longueur, S, P : réel
Ecrire (entrer la largeur)
Lire (largeur)
Ecrire (entrer la longueur)
Lire (longueur)
 $S = \text{largeur} * \text{longueur}$ 
 $P = 2 * (\text{largeur} + \text{longueur})$ 
Ecrire (la surface d'un rectangle est :), S
Ecrire (le périmètre d'un rectangle est :), P
Fin
```

##### Programme fortran :

```
program rectangle
real largeur, longueur, S, P
write(*,*) 'entrer la largeur'
read(*,*) largeur
```

## Leçon 1 : Généralités sur l'Algorithmique

---

```
write(*,*)' entrer la longueur'  
read(*,*) longueur  
S = largeur*longueur  
P = 2*(largeur+longueur)  
write(*,*)'la surface d'un rectangle est :', S  
write(*,*)'le périmètre d'un rectangle est :', P  
Fin
```

**Exécution du programme :**

```
entrer la largeur  
5 ↵  
entrer la longueur  
10 ↵  
la surface d'un rectangle est : 50  
le périmètre d'un rectangle est : 30
```

**Etat de l'écran**

largeur	5			
longueur		10		
S			50	
P				30

**Etat de la mémoire**

### 1.5 Structure d'un fichier Fortran

Le fichier fortran doit posséder un suffixe .for pour être reconnu par le compilateur (par exemple : perim.for).

La ligne comprend 80 caractères. Elle comprend plusieurs zones (1-5 type, 6 suite, 7-72 ordre fortran, 73-80 identification)

Les colonnes 1 à 5 contiennent le type:

Si la colonne 1 contient un C, cette ligne est ignorée par le programme, cela vous permet de mettre des commentaires pour faciliter la lecture de votre programme (ou des lignes blanches pour l'alléger). Il est important que votre programme soit bien lisible; les commentaires aident beaucoup.

## Leçon 1 : Généralités sur l'Algorithmique

---

Les colonnes 1 à 5 contiennent éventuellement des étiquettes, c'est-à-dire un nombre repérant une ligne et qui sert de référence ailleurs.

La colonne 6 indique, si elle contient un caractère, qu'il faut trouver dans cette ligne la suite de l'instruction de la ligne précédente.

Les colonnes 7 à 72 contiennent l'instruction proprement dite.

Les colonnes 73 à 80 sont ignorées par le compilateur. Vous pouvez y mettre un commentaire; elles servaient autrefois pour numéroter les cartes.

Le programme est structuré de la manière suivante :

- Le mot PROGRAM suivi d'un titre
- Déclarations des variables
- Lecture des données
- Traitement des données
- Sortie des résultats
- Le mot END pour signaler la fin du programme

### 1.6 Le Codage

Le système de numération utilisé habituellement est le système décimal. Un ordinateur étant basé sur le système binaire, il est utile de connaître les systèmes binaire (base 2), hexadécimal (base 16) et octal (base 8), ainsi que les techniques de conversion entre ces différents systèmes.

#### 1.6.1 La base décimale

Dans le cas du système décimal :

- a base est 10
- 10 chiffres : 0, 1, 2, 3, 4, 5, 6, 7, 8 et 9
- Ecriture d'un nombre décimal N quelconque :  $N = a_n 10^n + a_{n-1} 10^{n-1} + \dots + a_2 10^2 + a_1 10^1 + a_0 10^0 + a_{-1} 10^{-1} + a_{-2} 10^{-2} + \dots + a_{-p} 10^{-p}$

Avec  $0 \leq a_i \leq 9$

Ou encore :  $N = a_n, a_{n-1}, \dots, a_2, a_1, a_0, a_{-1}, a_{-2}, \dots, a_{-p}$

**Exemple :**

$$123,45 = 1 \times 10^2 + 2 \times 10^1 + 3 \times 10^0 + 4 \times 10^{-1} + 5 \times 10^{-2}$$

#### 1.6.2 La base binaire

## Leçon 1 : Généralités sur l'Algorithmique

---

Dans le cas du système binaire :

- la base est 2
- 2 chiffres : 0, 1

Représentation d'un entier naturel N :

$$N = a_n 2^n + a_{n-1} 2^{n-1} + \dots + a_2 2^2 + a_1 2^1 + a_0 2^0 + a_{-1} 2^{-1} + a_{-2} 2^{-2} + \dots + a_{-p} 2^{-p}$$

Avec  $0 \leq a_i \leq 9$

**Exemple :**

$$1010,101 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}$$

$$1010,101 = 8 + 2 + 0,5 + 0,125$$

$$1010,101 = 10,625 \text{ en base } 10$$

$$1010,101 = 10,625_{10}$$

Remarque : le nombre de bits nécessaires à la représentation d'un nombre N donné est k tel que :  $2^{k-1} \leq N \leq 2^k$  Il faut donc :  $k = E(\log_2 N) + 1$  bits

### 1.6.3 Autres systèmes

**Le système octal**

- la base est 8
- 8 chiffres : 0, 1, 2, 3, 4, 5, 6 et 7

**Système hexadécimal**

- la base est 16
- 16 chiffres : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E et F (A correspond à 10 en décimal, B à 11, ..., F à 15)

### 1.6.4 Correspondances entre bases

Il est recommandé de bien connaître la correspondance des 16 premiers nombres dans les quatre bases

Tableau 1 : Table de conversions des 17 premiers symboles

Décimal	binaire	Octal	Hexa décimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10

### 1.6.5 Conversion base b - système décimal

On développe le nombre selon les puissances de la base b.

**Exemple :**

$$1001110101_2 = 1 \times 2^0 + 1 \times 2^2 + 1 \times 2^4 + 1 \times 2^5 + 1 \times 2^6 + 1 \times 2^9 = 629_{10}$$

### 1.6.6 Conversion système décimal - base b

On applique le principe de la division euclidienne :

$$n = b * q + r \text{ avec : } 0 \leq r < b$$

On fait des divisions euclidiennes des quotients successifs par b jusqu'à ce que l'un des quotients devienne inférieur à b-1.

La liste inversée des restes ainsi obtenus constitue la décomposition recherchée. Ainsi, on a :  $125_{10} = 1111101_2$

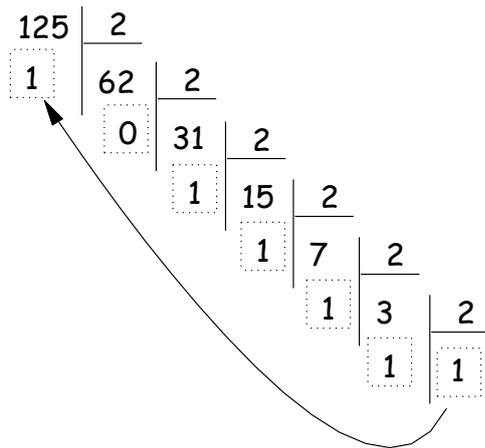


Figure 1 : Conversion système décimal - base b

## 1.7 Les variables et les constantes

### 1.7.1 Les variables

Une variable sert à stocker la valeur d'une donnée dans un langage de programmation, elle désigne un emplacement mémoire dont le contenu peut changer au cours d'un programme (d'où le nom de variable). La variable doit être déclarée avant d'être utilisée, elle doit être caractérisée par un nom (Identificateur) qui indique l'ensemble des valeurs que peut prendre la variable (entier, réel, booléen, caractère, chaîne de caractères, ...). Le choix du nom d'une variable est soumis à quelques règles qui varient selon le langage, mais en général un nom doit commencer par une lettre alphabétique exemple : E1 (1E n'est pas valide), doit être constitué uniquement de lettres, de chiffres et du soulignement (« \_ ») (Éviter les caractères de alphanumérique et les espaces), doit être différent des mots réservés du langage fortran.

**Exemples** : G20, Nom, TTC\_2012 (TTC 2012, TTC-2012, TTC+2012, read, end) : sont non valides).

### 1.7.2 Types des variables

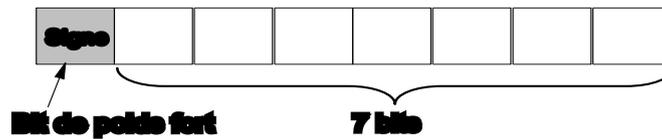
Lorsqu'on déclare une variable, il ne suffit pas de réserver un emplacement mémoire; il faut préciser ce que l'on voudra mettre dedans, car de cela dépendent la taille de l'emplacement mémoire et le type de codage utilisé.

Si l'on réserve un octet pour coder un nombre, on ne pourra coder que  $2^8 = 256$  valeurs différentes. Cela peut signifier par exemple les nombres entiers de 1 à 256, ou de 0 à 255, ou de -127 à +128. Si l'on réserve deux octets, on a droit à  $2^{16} = 65\ 536$  valeurs ; avec trois octets,  $2^{24} = 16\ 777\ 216$ , etc.

Les variables et les constantes peuvent avoir les types suivants :

### a) Le type INTEGER (entiers)

Le choix du nombre d'octets retenus pour écrire en mémoire un entier dépend du couple ordinateur plus logiciel qui l'équipe. Souvent, un octet est utilisé pour le codage d'un nombre entier. Un bit va servir pour coder le signe du nombre.



Donc un bit est perdu pour le signe, il ne reste plus que 7 bits pour coder l'entier. Or 7 bits représente  $2^7=128$  combinaison possible. Considérant que 0 est codé comme +0, on peut écrire sur 7 bits les nombres entre 0 127.

Pour les entiers négatifs, on peut commencer à -1 jusqu'à -128.

Donc avec 1 bit de signe, on pourra coder tous les entiers entre  $(-2^7 ; -1) \cup (0 ; 2^7-1) \Rightarrow (-128 \text{ et } 127)$ . On retiendra que même si on augmente le nombre d'octets pour coder un entier, il y aura toujours une limite à sa taille.

Question : Que se passe-t-il lorsqu'on veut faire la somme suivante  $200+200$  qui dépasse 127.

Il se produit une erreur : plantage ou effet de bords.

### Conclusion :

De ce fait, sur n bits, les nombres représentables sont :

$-2^{n-1} \leq i \leq 2^{n-1}-1$  d'où leur domaine d'utilisation est :

-128 à 127 (1 octet) pour INTEGER\*1 (pour un entier codé sur 1 octet).

-32 768 à 32 767 (2 octets) pour INTEGER\*2 (pour un entier codé sur 2 octets).

-2 147 483 648 à 2 147 483 647 (4 octets) pour INTEGER\*4 (pour un entier codé sur 3 octets).

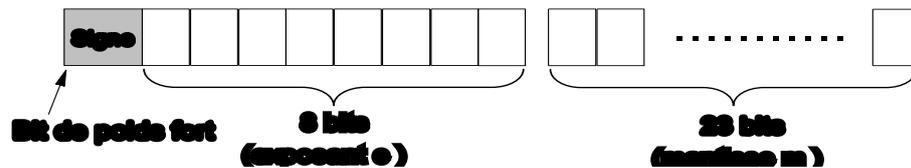
### Exemples :

Type entier tels que : 0, 405, -10,...

### b) Le type REAL (réels)

Un nombre réel ou flottant est caractérisé par : son signe, son exposant ou caractéristique et sa mantisse.

**Représentation d'un nombre réel sur 32 bits (codage en 4 octets).** Ce type de réel, appelé réel simple précision, admet un motif binaire de la forme **Seeeeeeem....m**.



**s** : bit de signe, **e** : exposant sur 8 bits à excédent 127, et **m** : mantisse sur 23 bits.

Ce type de représentation permet de représenter les nombres :

$1.2 \cdot 10^{-38} \leq |r| \leq 3.4 \cdot 10^{38}$  avec 6 chiffres significatifs. La déclaration en fortran se fait par l'instruction REAL ou REAL\*4.

**Représentation d'un nombre réel sur 64 bits (codage en 8 octets).** Ce type de réel, appelé réel double précision, admet un motif binaire de la forme :  
seeeeeeeeeem—m

**s** : bit de signe, **e** : exposant sur 11 bits à excédent 1047, **m** : mantisse sur 52 bits. Le nombre représenté correspond à :

$$r = s \cdot 1.m \cdot 2^{e-1023}$$

Ce type de représentation permet de représenter les nombres :

$2.2 \cdot 10^{-308} \leq |r| \leq 1.8 \cdot 10^{308}$  avec 15 chiffres significatifs. La déclaration en fortran se fait par l'instruction DOUBLE PRECISION ou REAL\*8.

### Exemples :

Type réel tels que : 0.5, -3.67, 1.5e+5,...

#### c) Le type LOGICAL (logique)

C'est le type qui permet de gérer les booléens en Fortran. Les deux valeurs possibles sont .true. et .false.

#### d) Le type CHARACTER (caractères)

C'est le type qui permet de gérer les chaînes de caractères. Les chaînes de caractères sont écrites indifféremment entre guillemets ou entre apostrophes.

Type caractère tels que : 'a', 'B', '\*', '9', '@', ' ', ...

### 1.7.3 Les constantes

Une **CONSTANTE**, comme une variable, peut représenter un chiffre, un nombre, un caractère, une chaîne de caractères, un booléen. Toutefois, contrairement à une variable dont la valeur peut être modifiée au cours de l'exécution de l'algorithme, la valeur d'une constante ne varie pas.

### 1.8 Opérandes et operateurs

#### 1.8.1 Définitions

Un OPERATEUR est un outil qui permet d'agir sur une variable ou d'effectuer des calculs.

Un OPERANDE est une donnée utilisée par un opérateur.

**Exemple** : Dans «8+3», «+» désigne l'opérateur ; «8» et «3» sont les opérandes.

#### 1.8.2 Types d'opérateurs

Il existe plusieurs types d'opérateurs :

**Les opérateurs arithmétiques** qui permettent d'effectuer des opérations arithmétiques entre opérandes numériques :

- Opérateurs élémentaires : «+», «-», «x», «÷»
- Changement de signe : «-»
- Elévation à la puissance : «^»
- Reste d'une division entière : «mod»

**Les opérateurs de comparaison** («=», «≠», «>», «<», «≥» et «≤») qui permettent de comparer deux opérandes et produisent une valeur booléenne, en s'appuyant sur des relations d'ordre :

**Les opérateurs logiques** qui combinent des opérandes booléennes pour former des expressions logiques plus complexes

- Opérateur unaire : «non» (négation)
- Opérateurs binaires : «et» (conjonction), «ou» (disjonction),

**L'opérateur d'affectation**, représenté par le symbole «←», qui confère une valeur à une variable ou à une constante.

**Exemple** : (affectation de la valeur 7 à la variable (ou à la constante) a)

**Opérateurs sur les entiers et les réels** : addition, soustraction, multiplication, division, puissance, comparaisons, modulo (reste d'une division entière)

**Opérateurs sur les booléens** : comparaisons, négation, conjonction, disjonction

**Opérateurs sur les caractères** : comparaisons

**Opérateurs sur les chaînes de caractères** : comparaisons, concaténation

#### 1.8.3 Priorité des opérateurs

A chaque opérateur est associée une priorité. Lors de l'évaluation d'une expression, la priorité de chaque opérateur permet de définir l'ordre d'exécution des différentes

## Leçon 1 : Généralités sur l'Algorithmique

---

opérations. Aussi, pour lever toute ambiguïté ou pour modifier l'ordre d'exécution, on peut utiliser des parenthèses.

**Ordre de priorité décroissante des opérateurs arithmétiques et de concaténation :**

- Les parenthèses
- « ^ » (élévation à la puissance)
- « - » (changement de signe)
- « x », et « ÷ »
- « mod »
- « + » et « - »

**Ordre de priorité décroissante des opérateurs logiques :**

- « non », « et », « ou ».

**Remarque :**

- La question de l'ordre de priorité des opérateurs de comparaison ne se pose pas.
- Les opérations entre parenthèses sont prioritaires.

**Exemples :**

$$3^{**}2+4 = 9+4=13$$

$$3^{**} (2+4)=3^{**}6 \text{ car les parenthèses sont plus prioritaires}$$

$$17 \text{ MOD } 10 \text{ DIV } 3 = (17 \text{ MOD } 10) \text{ DIV } 3 = 7 \text{ DIV } 3 = 2$$

$$12*3+5 \text{ et } (12*3) + 5 \text{ valent strictement la même chose, à savoir 41.}$$

$$\text{Par contre, } 12*(3+5) \text{ vaut } 12*8 \text{ soit 96.}$$

$$5. + 4.*9. **2 = 5. + (4.*(9. **2)) = 329$$

$$e.\text{OR}.f.\text{AND}.g = e.\text{OR}.(f.\text{AND}.g)$$

$$a^{**}b+c.\text{GT}.d.\text{AND}.e = (((a^{**}b)+c).\text{GT}.d).\text{AND}.e$$

$$7 + 9 / 3 - 10 * 2 = 7 + 3 - 10 * 2 = 7 + 3 - 20 = 10 - 20 = -10$$

$$(5 * 6 + (8 + 2*7 - 4 / 2)) = (5 * 6 + (8+14 - 4 / 2)) = (5 * 6 + (8+14 - 2)) = (5 * 6 + (22 - 2)) = (5 * 6 + 20) = 30 + 20 = 50$$

$$(5+2 > 8-6) \text{ AND } (7 < 9) = (7 > 2) \text{ AND } (7 < 9) = \text{TRUE AND TRUE} = \text{TRUE}$$

### 1.9 L'instruction d'affectation

L'affectation est l'opération qui consiste à stocker une valeur dans une variable. Cette opération se fait à l'aide de la syntaxe suivante :

**Nom- variable ← Valeur**

**Exemple :**

$$x \leftarrow 5$$

Signifie mettre la valeur 5 dans la case identifiée par x. A l'exécution de cette instruction, la valeur 5 est rangée en x (nom de la variable).

### 1.10 Les notions de lecture et d'écriture

L'ENTREE ou la lecture de données correspond à l'opération qui permet de saisir des valeurs pour qu'elles soient utilisées par le programme. Cette instruction est notée « lire identificateur ».

La SORTIE ou l'écriture des données permet l'affichage des valeurs des variables après traitement. Cette instruction est notée « afficher identificateur ».

#### Exemple de lecture et d'écriture :

Ecrire et exécuter le programme fortran qui demande un entier à l'utilisateur, puis affiche son carré.

**Solution :**

**Algorithme :**

Algorithme Calcul\_du\_Carre

Début

Variables A, B : entier

Écrire ("entrer la valeur de A ")

Lire(A)

$B \leftarrow A * A$

Écrire ("le carre de ", A, "est :", B)

Fin

**Programme fortran :**

```
program Calcul_du_Carre
```

```
INTEGER A, B
```

```
write(*,*)' entrer la valeur de A '
```

```
read(*,*) A
```

```
B = A*A
```

```
write(*,*)' le carre de', A, 'est',B
```

```
Fin
```

**Exécution du programme :**

entrer la valeur de A
10 ↵
le carre de 10 est 100

**Etat de l'écran**

A	10	
B		100

**Etat de la mémoire**

## Travaux Dirigés 1- Généralités sur l'Algorithmique

---

### Exercice 1 :

Comme identificateurs, quels sont ceux qui sont inacceptables en Fortran:  
B7, 3, BONUS, 5B, Z, Nombre, C34, End, F%, AB, Program

### Exercice 2:

Considérons les algorithmes ci-dessous. Quel sera le contenu des variables a, b et éventuellement c après leur exécution ?

a)	b)	Algo PS
Algo A1	Algo A2	Entier A, B
Entier a, b, c :	Caractère a, b	Début
Debut	Début	$A \leftarrow 5$
$a \leftarrow 1$	$a \leftarrow '1'$	$B \leftarrow 2$
$b \leftarrow 2$	$b \leftarrow '2'$	$A \leftarrow B$
$b \leftarrow a + b$	$a \leftarrow a + b$	$B \leftarrow A$
$c \leftarrow a + b$	Fin	Fin
Fin		

### Exercice 3 :

- 1) Quelles seront les valeurs des variables A et B après l'exécution de l'algorithme PS ?
- 2) Les deux dernières instructions permettent-elles d'échanger les deux valeurs de B et A ?
- 3) Si l'on inverse les deux dernières instructions, cela change-t-il quelque chose ?

### Exercice 4 :

Écrire  $(34)_{10}$  et  $(27)_{10}$  en binaire puis effectuer l'opération en binaire  $(34)_{10} + (27)_{10}$  et vérifier que le résultat obtenu soit le bon.

### Exercice 5:

**A-** Quel est l'ordre de priorité des différents opérateurs de l'expression suivante :  
 $a + b ** c / d / e$  ;  $a ** b ** c * d + e$  ;  $((3 * a) - x ^ 2) - (((c - d) / (a / b)) / d)$

**B-** Que valent les réels suivants?

$$R1 = 12./3./2.$$

$$R2 = 12./3.*2.$$

$$R3 = 2/3$$

$$R4 = R1 + 24.*3./(-2.)**3 + 2.*(5/2 + 1)$$

$$R5 = 12/ (3/2)$$

**C-** Sachant que  $a = 4$ ,  $b = 5$ ,  $c = -1$  et  $d = 0$ , évaluer les expressions logiques suivantes :

1-  $(a < b)$  ET  $(c \geq d)$

2- NON  $(a < b)$  OU  $(c \neq b)$

3- NON  $((a \neq b ^ 2)$  OU  $(a * c < d)$ )

**Solutions des exercices :**

**Exercice 1 :**

**Identificateurs acceptables en Fortran:**

B7, BONUS, Z, Nombre, C34, AB

Comme identificateurs, quels sont ceux qui sont inacceptables en Fortran:

**Identificateurs inacceptables en Fortran:**

3, 5B, End, F%, Program

**Exercice 2:**

a)

a = 1

b = 3

c = 4

b)

Ne marche pas. On ne peut pas additionner des caractères.

**Exercice 3 :**

1) Après l'exécution de l'algorithme la valeur des variables est :

A = 5

B = 2

A = 2

B = 2

2) Les deux dernières instructions ne permettent donc pas d'échanger les deux valeurs de B et A, puisque l'une des deux valeurs (celle de A) est ici écrasée.

3) Si l'on inverse les deux dernières instructions, cela ne changera rien du tout, hormis le fait que cette fois c'est la valeur de B qui sera écrasée

**Exercice 4 :**

Convertissons tout d'abord 34 en binaire. Cela donne

$$34 = 2 \times 17 + 0$$

$$17 = 2 \times 8 + 1$$

$$8 = 2 \times 4 + 0$$

$$4 = 2 \times 2 + 0$$

$$2 = 2 \times 1 + 0$$

$$1 = 2 \times 0 + 1$$

On a donc  $(34)_{10} = (100010)_2$ . Convertissons maintenant 27 en binaire. On a

$$27 = 2 \times 13 + 1$$

$$13 = 2 \times 6 + 1$$

## Travaux Dirigés 1- Généralités sur l'Algorithmique

$$6 = 2 \times 3 + 0$$

$$2 = 2 \times 1 + 1$$

$$1 = 2 \times 0 + 1$$

Et  $(27)_{10} = (11011)_2$ . On effectue maintenant l'addition de  $(100010)_2$  et  $(11011)_2$ . Pour rappel, l'addition en binaire fonctionne de la manière suivante :

+	0	1
0	0	1
1	1	10

D'où l'opération suivante

$$\begin{array}{r}
 1 \ 0 \ 0 \ 0 \ 1 \ 0 \\
 + \quad 1 \ 1 \ 0 \ 1 \ 1 \\
 \hline
 1 \ 1 \ 1 \ 1 \ 10 \ 1
 \end{array}$$

On a  $(100010)_2 + (11011)_2 = (111101)_2$ . Or  $(34)_{10} + (27)_{10} = (61)_{10}$ , vérifions si  $(61)_{10} = (111101)_2$ .

$$2 \times 0 + 1 = 1$$

$$2 \times 1 + 1 = 3$$

$$2 \times 3 + 1 = 7$$

$$2 \times 7 + 1 = 15$$

$$2 \times 15 + 0 = 30$$

$$2 \times 30 + 1 = 61$$

On a bien  $(61)_{10} = (111101)_2$ , le résultat obtenu en binaire est bien conforme au résultat obtenu en base 10

### Exercice 5 :

A-

$$a + b^{**} c / d / e$$

$$4 \ 1 \ 2 \ 3$$

$$a^{**} b^{**} c * d + e$$

$$2 \ 1 \ 3 \ 4$$

$$((3 * a) - x^2) - (((c - d) / (a / b)) / d)$$

$$1 \ 3 \ 2 \ 8 \ 4 \ 6 \ 5 \ 7$$

B- (2, 8, 0, -1, 12)

C- 1- Faux

2- Vrai

3- Faux

Le résultat d'une expression logique est toujours Vrai ou Faux

### Exercice 1 :

Lancer l'environnement Fortran Power Station, écrire un programme fortran qui affiche 'Bonjour'. Sauvegarder ce programme (cliquer sur le menu File puis le sous-menu Save). sous le nom Tp1.for. Exécuter le programme.

### Exercice 2:

Algo Exercice

début

entier a, b, c

réel d

$a \leftarrow 5.1$

$b \leftarrow 3$

Écrire a, b

$c \leftarrow 2$

Écrire (c / b)

$d \leftarrow c / b$

Écrire (d)

Fin

Traduire l'algorithme en fortran puis exécuter et commenter les résultats affichés.

### Exercice 3:

Saisir le programme suivant :

Program ExO2

```
integer*1 a,b
```

```
integer*2 c,d
```

```
a =1200
```

```
write(*,*)'a', a
```

```
d=122
```

```
write(*,*) 'd',d
```

```
b=32767+3
```

```
write(*,*) 'b',b
```

```
c=-32768+3
```

```
write(*,*)'c',c
```

```
End
```

- 1) Expliquer les déclarations: integer\*1 ; integer\*2 ; integer\*4.
- 2) Exécuter le programme ; d'où vient l'erreur survenue dans les valeurs de a et b ?

### Exercice 4:

Saisir et exécuter le programme suivant :

```
program Exo3
```

```
character n
```

```
character*30 p
```

```
write (*,*)'donner votre nom'
```

## Travaux Pratiques 1- Généralités sur l'Algorithmique

---

```
read (*,*) n
write (*,*) n
end
```

- 1- Quelle est la différence entre les déclarations `character n` et `character*30`
- 2- Que fait l'instruction `write (*,*) 'donner votre nom'`
- 3- Saisir puis exécuter le programme et introduire votre nom. Commenter les résultats affichés.
- 4- Ajouter à la fin du programme les instructions suivantes :  

```
read (*,*)
print*, "la chaine saisie en p est:",p
```
- 5- Exécuter le programme en donnant une phrase de 30 caractères.

### Exercice 5:

- 1- Saisir sous Fortran le programme suivant, puis corriger les éventuelles erreurs survenues.

```
program Exo4
Caractère nom*15
Read*4 a;b,c,d
A=2*5/6
B=2*[5/6]
C=2*(5/6.0)
D=2.0*(5.0/6.0)
write*'la valeur de a=',A
write('la valeur de b=',B
write*, 'la valeur de c=',C
D=2*A+5***B
Fin
```

- 2- Comment l'ordinateur a-t-il évalué les quatre expressions arithmétiques ?

### Exercice 6:

Une variable de type logique ne peut prendre que deux valeurs distincts vrai ou faux. On peut introduire une valeur dans une variable logique soit : Directement par affectation ou par clavier en saisissant T ou true ; F ou false.

Saisir et exécuter le programme Exo5 :

#### Program Exo5

```
logical a,b,c,d
a=.true.
b=.false.
Read(*,*) c ! T ou true ; F ou false
D=((5.LT.6).and.(8.GE.7))
Write(*,*) a,b,c,d
end
```

Quelle sera la valeur de d si la variable c est « false ».

### Solutions des exercices :

#### Exercice 1 :

```
Program Tp1
write (*,*) 'Bonjour '
End
```

#### Exercice 2:

```
Program Exo2
integer a, b, c
real d
a =5.1
b = 3
write (*,*) 'a=',a, 'b=',b
c = 2
write (*,*) '(c / b)=', (c / b)
d = 2. / b
write (*,*) 'd=',d
end
```

a=5, b=3, (c / b) =0, d=0.6666667.

L'affichage de la variable **a** se fait que sur la partie entière ou a=5. Pour le calcul de (c/b) se fait entre deux variable entières **c** et **d**, donc L'affichage sera que sur la partie entière (c / b) =0.

#### Exercice 3:

1)

La déclaration INTEGER\*1 pour un entier coder sur 1 octet, et le domaine d'utilisation est : -128 à 127

La déclaration INTEGER\*2 pour un entier coder sur 2 octet, et le domaine d'utilisation est : -32 768 à 32 767.

La déclaration INTEGER\*4 pour un entier coder sur 4 octet, et le domaine d'utilisation est : -2 147 483 648 à 2 147 483 647.

2)

L'exécution du programme donne le résultat suivant :

```
a      -80
d      122
b       2
c     -32765
```

## Travaux Pratiques 1- Généralités sur l'Algorithmique

---

L'erreur survenue dans les valeurs de a et b est causée par les types de déclarations INTEGER\*1 et INTEGER\*2. Donc il faut que la déclaration soit INTEGER\*4.

### Exercice 4 :

- 1- La différence entre les déclarations character et character\*30 est que la taille maximale des variables pour la première est une seule case et pour la deuxième est 30 cases.
- 2- Le rôle de l'instruction write(\*,\*) 'donner votre nom' est d'afficher le message 'donner votre nom'
- 3- après l'exécution du programme introduire votre nom on remarque que la taille de la variable n est une seule case.

### Exercice 5 :

- 1- Les corrections nécessaires des erreurs sont soulignées et en gras.

program Exo4

character nom\*15

Real\*4 a,b,c,d

A=2\*5/6

B=2\* ( 5/6)

C=2\*(5/6.0)

D=2.0\*(5.0/6.0)

write(\*,\*)' la valeur de a=' ,A

write(\*,\*)' la valeur de b=' ,B

write(\*,\*)' la valeur de c=' ,C

write(\*,\*)' la valeur de d=' ,D

end

- 2-

Dans le calcul de la variable A l'ordre est  $2*5=6$  puis  $6/6=1$ .

Pour le calcul de la variable B l'ordre est  $(5/6)=0$  puis  $2*0=0$ .

Pour le calcul de la variable C l'ordre est  $(5/6.0)=0.833333$  puis  $2*0.833333=1.666667$ .

Pour le calcul de la variable D l'ordre est  $(5.0/6.0)=0.833333$  puis  $2.0*0.833333=1.666667$

La valeur de d si la variable c est « false » est « true ».

## Leçon 2 : Les opérations de lecture et d'écriture

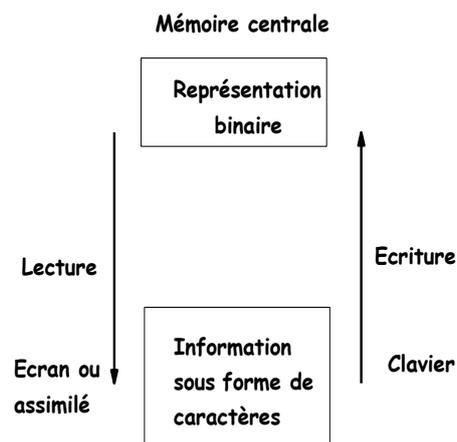
## Leçon 2 : Les opérations de lecture et d'écriture

### Objectif :

Manipuler correctement les opérations de lecture et d'écriture

### 2.1 Introduction

- Les ordres d'entrée/sortie permettent de transférer des informations entre la mémoire centrale et les unités périphériques (terminal, clavier, disque, imprimante,...)
- Les opérations d'entrée/sortie engendrent des conversions caractères alphanumériques/binaires



### 2.2 Syntaxe générale

#### - Lecture

Read (périphérique, format [, options]) liste

Read format, liste

#### - Ecriture

Write (périphérique, format [, options]) liste

Print format, liste

#### - Remarque :

## Leçon 2 : Les opérations de lecture et d'écriture

---

Les deuxièmes formes de syntaxe correspondent à l'entrée et la sortie dites standard (en général l'écran).

(options) reprend l'ensemble des arguments qui permettent une gestion des erreurs, un contrôle du changement de ligne,...

### 2.3 Entrées-sorties standards : Ecriture en format libre

```
print *, 'Energie totale = ', Etot, ' eV'
```

```
write(*,*) 'Energie totale = ', Etot, ' eV'
```

- Le \* associé au format indique à l'instruction d'écriture de formater automatiquement les données. (Ex: données numériques écrites avec toutes leurs décimales représentées en machine)
- Le \* associé au périphérique (write) correspond à la sortie standard.

#### Exemple:

```
Write (*,*) a, b, c, d
```

Écrit les valeurs a, b, c, d sur une ligne, séparées par des blancs.

```
Write (*,*) ' La valeur de A est ', A
```

Écrit la chaîne « La valeur de A est », suivie de la valeur A.

### 2.4 Entrées-sorties standards : Lecture en format libre

```
read *, title, nb, x
```

```
read(*,*) title, nb, x
```

Le \* signifie que la conversion des données vers leur représentation machine doit se faire automatiquement.

Les données d'entrée doivent être séparées

- par des espaces
- ou des virgules
- ou des fins de ligne (à éviter)

Les chaînes de caractères se notent entre apostrophes (') ou entre guillemets (")

## Leçon 2 : Les opérations de lecture et d'écriture

---

### 2.4 Les formats

#### 2.4.1 Introduction

##### Motivations :

Lecture : adaptation à des données existantes non adaptées à un format libre

Ecriture : présentation des résultats

Le format se présente comme une liste de descripteurs :

(i3, f5.3)

Il y a deux manières de donner le format d'entrée/sortie. La première dans une instruction à part, dotée d'une étiquette.

`write(*,1000) liste`

1000 format ( liste\_descripteur)

Et la deuxième dans l'instruction d'entrée/sortie elle-même

`write(*, ( liste_descripteur')) liste`

#### 2.4.2 Descripteur des entiers

##### Syntaxe I w

w : nombre de caractères

Lecture : w caractères sont lus, y compris le signe éventuel et des espaces (interprétés comme zéro) devant.

`read(*,(i3,i4)) n,p`

Ecriture : Impression de la valeur entière justifiée à droite dans les w caractères, avec le signe éventuel et des blancs pour compléter le champ w.

`write(*,(i3,i4)) n,p`

#### 2.4.2 Descripteur des réels

##### Sans exposant Fw.d

w : nombre de caractères (chiffres, point décimal, signe, blancs)

d : nombre de chiffre décimaux

Valeur interprétée	Format	Ecriture
4.86	f4.1	^4.9
-5.23	f5.2	-5.23
4.25	f10.2	^^^^^^4.25

## Leçon 2 : Les opérations de lecture et d'écriture

---

**Avec exposant** Ew.d

w : nombre de caractères (chiffres, point décimal, signe, exposant, blancs)

d : nombre de chiffre décimaux

Attention :  $w \geq d+6$

**Exemple :**

WRITE(\*,'(F6.3)') k

F6.3 : écriture du réel k sur 6 caractères avec 3 chiffres en partie décimale.

Valeur interprétée	Format	Ecriture
0.0123456	e13.6	^0.123456E-01
-47.678	e12.4	^-0.4768E+02
4.25	e12.3	^^^^0.425E01

Remarque :

Gw.d  $\Rightarrow$  choix automatique entre Fw.d et Ew.d

Exposant à plus de deux chiffres : E w.d E e avec e nombre de chiffres de l'exposant

### 2.4.3 Descripteur des logiques

**Syntaxe :**

L w

w-1 blancs suivis de la lettre T pour une valeur .true. ou F pour une valeur .false.

write(\*,'(L5)') test

## 2.5 Les fichiers

### 2.5.1 Introduction

Un fichier permet de stocker des données des enregistrements sur des disques. On lui donne généralement une extension .dat, .txt, .out, etc. il peut être généré par un programme fortran ou par un éditeur externe.

## Leçon 2 : Les opérations de lecture et d'écriture

---

Un fichier est formaté si les données sont stockées sous forme de caractères alphanumériques. C'est le cas des fichiers produits un éditeur externe. Il est non formaté si les données sont stockées sous forme binaire (représentation machines).

On peut accéder au contenu d'un fichier de deux manières :

- En accès séquentiel : on doit lire tous les enregistrements qui précèdent celui au quel on souhaite accéder. les enregistrements peuvent avoir des tailles différentes.
- En accès direct : l'ordre de lecture /écriture indique le numéro de l'enregistrement. les enregistrements doivent tous avoir la même taille.

### 2.5.2 Ouverture et création d'un fichier

Exploiter un fichier au sein d'un programme nécessite son ouverture. En Fortran, on utilise l'instruction **open**.

Open (10, file='result.dat') ! Forme compacte

Open (10, file='result.dat', status='old') ! Forme avec option

L'instruction permet :

- de connecter le nom de fichier à un numéro d'unité logique (numéro repris par la suite pour toute opération de lecture/écriture).
- de spécifier le mode désiré : lecture, écriture ou les deux.
- d'indiquer le mode de transfert (avec ou sans formatage).
- d'indiquer le mode d'accès (séquentiel ou direct).
- de gérer les erreurs d'ouverture

La forme générale simple d'un ordre de lecture est :

```
read((unit=num_unite), (fmt=liste_descripteurs) liste
```

```
read(10,'(2g15.6)') x,y
```

```
read(10,*) x,y
```

```
read(numfich,fmt=*) x,y,chaine
```

```
read(unit=20,fmt='(a)') chaine
```

La forme générale simple d'un ordre d'écriture est :

```
write((unit=num_unite), (fmt=liste_descripteurs) liste
```

```
write(10,'(2g15.6)') x,y
```

```
write(10,*) x,y
```

```
write(numfich,fmt=*) x,y,chaine
```

```
write(unit=20,fmt='(a)') chaine
```

### Exercice 1 :

Ecrire un algorithme qui demande à l'utilisateur de saisir son nom et son prénom puis afficher bonjour suivi de nom et de prénom de l'utilisateur.

### Exercice 2 :

Ecrire l'algorithme permettant de calculer et d'afficher le salaire net d'un employé. Sachant que :

Le salaire net (SN) = Salaire brut - Valeur de l'impôt

Salaire brut (SB) = (Salaire de base (SDB) + Prime des enfants) \* Taux de travail

Taux de travail (TT) = Nombre de jours travaillés (NJ) / 26

Prime des enfants (PE) = Prime d'un enfant(P1E) \* Nombre d'enfants (NE)

Valeur de l'Impôt (VI) = Taux de l'Impôt(TI) \* Salaire Brut

### Exercice 3 :

On cherche à écrire un algorithme qui affiche la table de multiplication d'un nombre n quelconque. Exemple : n = 9

$$1 * 9 = 9$$

$$2 * 9 = 18$$

$$3 * 9 = 27$$

$$4 * 9 = 36$$

$$5 * 9 = 45$$

$$6 * 9 = 54$$

$$7 * 9 = 63$$

$$8 * 9 = 72$$

$$9 * 9 = 81$$

### Solutions des exercices :

#### Exercice 1 :

Algorithme bonjour  
Caractère nom, prénom

Début  
Ecrire " donner votre nom : "  
Lire (nom)  
Ecrire " donner votre prénom : "  
Lire (prénom)  
Ecrire " bonjour" , nom, prénom  
Fin

#### Exercice 2 :

Algorithme Salaire  
Réel SN, SB, TT, PE, VI

Début  
Ecrire " donner le Salaire Brut : "  
Lire (SB)  
Ecrire " donner le Taux de l'Impôt: "  
Lire (TI)  
 $VI = SB + TI$   
Ecrire " donner le Nombre d'enfants: "  
Lire (NE)  
Ecrire " donner la Prime d'un enfant: "  
Lire (P1E)  
 $PE = P1E * NE$   
Ecrire " donner le Nombre de jours travaillés: "  
Lire (NJ)  
 $TT = NJ / 26$   
Ecrire " donner le Salaire de base: "  
Lire (SDB)

$SB = (SDB + PE) * TT$   
 $SN = SB - VI$

Ecrire " Le salaire net est : " , SN

Fin

**Exercice 3 :**

L'affichage ci-contre est obtenu par l'algorithme suivant :

Algo Multiplication

Début

Entier  $i, n$

Lire ( $n$ )

Pour  $i=1, 10$

Ecrire  $i, " \times ", n, " \text{ est égal à } ", i \times n$

Fin

### Exercice 1 :

Ecrire le programme fortran qui donne l'exécution au dessous, (ce programme calcul la conversion des angles 0, 10, 20, ....., 90 de Degrés vers Radians) :

```
*****
* Degres * Radians *
*****
* 0 * .00000 *
* 10 * .17453 *
* 20 * .34907 *
* 30 * .52360 *
* 40 * .69813 *
* 50 * .87267 *
* 60 * 1.04720 *
* 70 * 1.22173 *
* 80 * 1.39627 *
* 90 * 1.57080 *
*****
Stop - Program terminated.

Press any key to continue
```

### Note :

180 Degrés = 3.142857 Radians

### Exercice 2 :

1- Écrire un programme fortran qui calcul la moyenne de 5 note de type réel (afficher la valeur de la moyenne sur 10 caractères avec 5 chiffres en partie décimale).

2- Exécuter le programme en donnant 5 notes (1, 2, 3, 4 et 5) réel; représentée l'état d'écran de l'exécution.

### Exercice 3 :

Ecrire le programme fortran qui donne cette exécution.

```
_____ -1024 _____ 666 _____ 112358
```

```
_____ -1024 _____ +3.1416
```

```
_____ +666 _____ -88.9000
```

```
_____ +112358 _____ +1.2346
```

### Solutions des exercices :

#### Exercice 1:

```
PROGRAM DEGRAD
! Imprime une table de conversion degrés -> radians
! =====
  INTEGER DEG
  REAL RAD, COEFF
  WRITE (*, 10)
10 FORMAT (' ', 20('*') /
  &      ' * Degrés * Radians *' /
  &      ' ', 20('*') )
! Corps de programme
  COEFF = (2.0 * 3.1416) / 360.0
  DO DEG = 0, 90, 10
    RAD = DEG * COEFF
    WRITE (*, 20) DEG, RAD
20 FORMAT (' * ', I4, ' * ', F7.5, ' *')
  END DO
! Fin du tableau
  WRITE (*, 30)
30 FORMAT (' ', 20('*') )
  STOP
END PROGRAM DEGRAD
```

#### Exercice 2:

```
1-
  program mean
  real a1,a2,a3,a4,a5,av
c  input five real numbers
  write(*,*) 'Input five real numbers : '
  read(*,*) a1,a2,a3,a4,a5
c
  av = (a1+a2+a3+a4+a5)/5.0
  write(*,100) av
100 format(' Average of five numbers is : ',4x,f10.4)
  stop
  end
```

2-

```
Input five real numbers :  
1. 2. 3. 4. 5.  
Average of five numbers is :    3.0000  
Stop - Program terminated.  
  
Press any key to continue
```

Etat d'écran

**Exercice 3:**

PROGRAM AFFICHAGE

```
A = 3.14159  
B = -88.9  
C = 123.4567E-02  
I = -1024  
J = 666  
K = 112358  
WRITE(*,702)I,J,K  
702 FORMAT(' ',6I10)  
WRITE(*,*)  
WRITE(*,703)I,A,J,B,K,C  
703 FORMAT(' ',SP,I10,F10.4)
```

END

## Leçon 3 : Les structures itératives

## Leçon 3 : Les structures itératives

### Objectif

Construire des algorithmes comportant des traitements itératifs.

### 3.1 Définition

Une boucle permet de parcourir une partie d'un programme un certain nombre de fois. Une itération est la répétition d'un même traitement plusieurs fois. Un indice de boucle varie alors de la valeur minimum (initiale) jusqu'à la valeur maximum (finale).

### 3.2 La boucle « Pour »

Cette structure est une BOUCLE ITERATIVE ; elle consiste à répéter un certain traitement un nombre de fois fixé à l'avance

Cette structure utilise une variable (indice) de contrôle d'itérations caractérisée par :

- sa valeur initiale,
- sa valeur finale,
- son pas de variation.

La syntaxe de la boucle pour est comme suit :

**Pour I de 1 jusqu'à N pas 1**

**Instructions**

**Fin Pour**

I : variable

1 : valeur initiale

N : valeur finale

Le pas de variation égale à 1

### Exemple :

Ecrire un algorithme qui demande un nombre de départ, et qui ensuite affiche les dix nombres suivants. Par exemple, si l'utilisateur entre le nombre 17, le programme affichera les nombres de 18 à 27.

### Solution :

#### Algorithme :

Algorithme nombres

Début

Variables I, N : entier

Écrire ("entrer la valeur de I ")

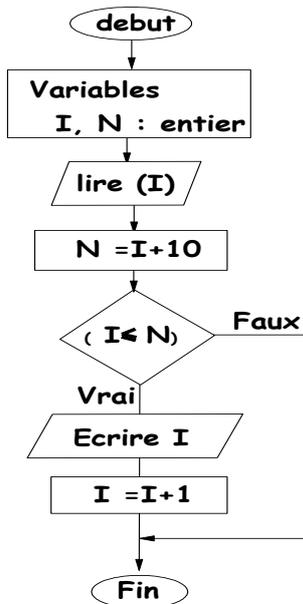
Lire (I)

## Leçon 3 : Les structures itératives

---

```
N=I+10
Pour I de 17 jusqu'à N pas 1
  Ecrire (I)
Fin Pour
Fin
```

Organigramme :



Programme fortran :

```
program nombres
INTEGER I, N
write(*,*) ' entrer la valeur de I '
read(*,*) I
N=I+1
DO I=17,N,1
write(*,*) I
ENDDO
END
```

### 3.3 La boucle TANT QUE

Une action ou un groupe d'actions est exécuté répétitivement tout le temps où une condition est vraie.

**Syntaxe**

**Tant Que (Condition) Faire**

**Instructions**

**Fin Tant que**

**Remarque :** la vérification de la condition s'effectue avant les actions. Celles-ci peuvent donc ne jamais être exécutées.

### Leçon 3 : Les structures itératives

---

#### Exemple :

On veut écrire un algorithme qui calcul la somme des entiers positifs inférieurs ou égaux à N.

#### Solution :

##### Algorithme :

Algorithme somme

Début

Variables I, N, S : entier

Écrire ("entrer la valeur de N ")

Lire (N)

I=0

S=0

Tant Que (I ≤ N) alors

S=S+I

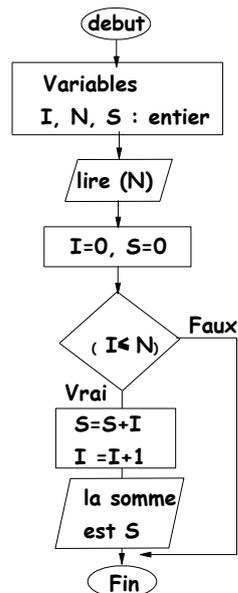
I=I+1

Fin Tant Que

Ecrire ('la Somme est', S)

Fin

##### Organigramme :



##### Programme fortran :

```
program somme
INTEGER I, N, S
write(*,*) 'entrer la valeur de N '
read(*,*) S
I=0
S=0
DO While (I.LE.N) Then
S=S+I
I=I+1
ENDDO
write(*,*) 'la Somme est', S
END
```

### 3.4 La boucle REPETER ... JUSQUA ...

Une action ou un groupe d'actions est exécuté répétitivement jusqu'à ce qu'une condition soit vérifiée.

#### Syntaxe

#### Répéter

Instructions

Jusqu'à (Condition)

**Remarque** : la vérification de la condition s'effectue après les actions. Celles-ci sont donc exécutées au moins une fois.

#### Exemple :

Ecrire un algorithme qui demande successivement 10 nombres à l'utilisateur, et qui lui dise ensuite quel était le plus grand parmi ces 10 nombres :

Entrez le nombre numéro 1 : 7

Entrez le nombre numéro 2 : 24

.....etc.

Entrez le nombre numéro 10 : 13

Le plus grand de ces nombres est : 24

#### Solution :

##### Algorithme :

Algorithme plus\_grand

Début

Variables I, N, PG : entier

I=1

Écrire ("entrer la valeur de N ")

Lire (N)

PG = N

Répéter

Lire (N)

Si (PG  $\leq$  N) alors

PG = N

Fin Si

I=I+1

Jusqu'à (I = 10)

Ecrire ('La valeur la plus grand est', PG)

Fin

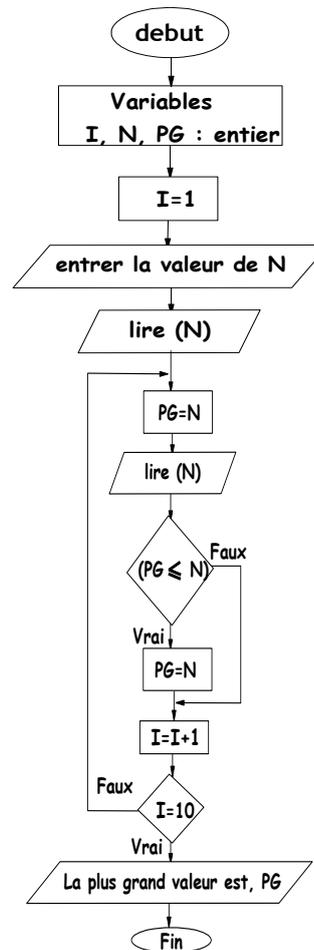
### Leçon 3 : Les structures itératives

---

#### Programme fortran :

```
program plus_grand
  INTEGER I, N, PG
  I=1
  write(*,*) 'entrer la valeur de N '
  read(*,*) N
5  PG = N
  read(*,*) N
  IF (PG.LE.N) then
  PG = N
  ENDIF
  I=I+1
  IF (I.NQ.10) GOTO 5
  write(*,*) 'La valeur la plus grand est', PG
  END
```

#### Organigramme :



### Travaux Dirigés 3 - Les structures itératives

---

#### Exercice 1 :

Écrire un algorithme qui demande un nombre de départ, et qui calcule sa factorielle.

NB : la factorielle de 8, notée  $8!$ , vaut  $1 \times 2 \times 3 \times 4 \times 5 \times 6 \times 7 \times 8$

#### Exercice 2 :

Écrire un algorithme qui demande 10 entiers, compte le nombre d'entiers positifs saisis, et affiche ce résultat.

#### Exercice 3 :

Écrire un algorithme permettant d'écrire un échiquier de 8 fois 8. On représentera les cases noires par des 'x' et les cases blanches par des espaces (figure 1).

x		x		x		x	
	x		x		x		x
x		x		x		x	
	x		x		x		x
x		x		x		x	
	x		x		x		x
x		x		x		x	
	x		x		x		x

#### Exercice 4 :

La population de la ville Alpha est de 1, 000, 000 d'habitants et elle augmente de 50,000 habitants par an. Celle de la ville Beta est de 500, 000 habitants et elle augmente de 4% par an. Écrire un algorithme permettant de déterminer dans combien d'années la population de la ville Beta dépassera celle de la ville Alpha.

#### Exercice 5 :

Écrire un algorithme qui demande à l'utilisateur un nombre est :

- Affiche les diviseurs de ce nombre.
- Le nombre de ces diviseurs.
- La somme des diviseurs de ce nombre.

#### Exercice 6 :

Écrire un algorithme pour afficher les premiers termes des suites suivantes (nombre de termes (n) demandé à l'utilisateur) :

Suite arithmétique :  $u_{n+1} = u_n + 2, u_0 = 1$

### Solutions des exercices :

#### Exercice 1 :

##### Algorithme :

Algo factorielle

Début

Variables N, I, F : Entier

Ecrire (Entrez un nombre N)

Lire (N)

F = 1

Pour i = 2 à N

F = F \* i

Fin Pour

Ecrire (La factorielle est :), F

Fin

##### Programme fortran :

```
program factorielle
```

```
INTEGER N, I, F
```

```
write(*,*) 'Entrez un nombre N'
```

```
read(*,*) N
```

```
F = 1
```

```
DO I=1, N
```

```
F = F * I
```

```
ENDDO
```

```
write(*,*) 'La factorielle est :', N
```

```
END
```

#### Exercice 2 :

Algo positifs

Entier a, i, cc

Début

Écrire "Saisir des entiers"

i = 0

cc = 0

Tant que (i < 10) faire

Lire (a)

i = i + 1

Si (a ≥ 0) alors

cc = cc + 1

Fin si

Fin Tant que

### Travaux Dirigés 3 - Les structures itératives

---

Écrine cc, "sont positifs"  
Fin

Ou bien, avec une boucle de type "pour"  
Algo positifs  
Entier a, i, cc  
Début  
Écrine "Saisir des entiers"  
cc = 0  
Pour i = de 1, 10  
Lire (a)  
Si ( $a \geq 0$ ) alors  
cc = cc + 1  
Fin si  
Fin pour  
Écrine cc, "sont positifs"  
Fin

#### Exercice 3 :

Algo Exercice  
Début  
Entier i, j  
Pour i = 1, 8  
  Pour j = 1, 8  
    Si  $((i+j) \bmod 2 = 0)$  alors  
      Écrine "x"  
    Sinon  
      Écrine " "  
    Fin si  
  Fin pour  
Fin pour  
Fin

#### Exercice 4 :

Algorithme Populations  
Début  
Entier années, alpha, beta  
alpha = 10 000 000  
beta = 5 000 000  
années = 0  
Tant que ( $\beta \leq \alpha$ ) alors  
  années = années + 1  
  alpha = alpha + 50 000  
  beta = beta \* 1.04

### Travaux Dirigés 3 - Les structures itératives

---

Fin tant que  
afficher "Il faut " , années , " années pour que la population de beta dépasse celle de  
alpha"  
Fin

#### Exercice 5 :

Algo nbr\_premier  
Variable compt, s, i, N : entier  
Ecrire ( entrer N )  
Lire (N)  
Compt = 0  
S = 0  
Pour i = 2 à N-1  
Si (N mod (i) = 0) alors  
Ecrire (i)  
Compt = compt+1  
S = S+i  
Fin Si  
Fin pour  
Ecrire (la somme est :, S)  
Ecrire (le nombre des diviseurs est :, compt)  
Fin

#### Exercice 6 :

Algo Exo5  
  
Variable u, i, n : entier  
Début  
Lire "nombre de termes ?", n  
u = 1  
i = 0  
Tant que ( i ≤ n ) alors  
écrire "U(",i,")=",u  
u = u + 2  
i = i + 1  
Fin Tant que  
Fin

### Travaux Pratiques 3 - Les structures itératives

---

#### Exercice 1 :

Écrire un programme fortran permettant d'écrire une table de multiplication comme celle présentée ci-contre. Dans un premier temps on ne s'occupera pas du nombre d'espaces entre les nombres, puis on affinera en en tenant compte.

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

#### Exercice 2:

1- Ecrire un programme fortran qui lit une valeur entière  $n$ , puis calcule et écrit les  $n$  premiers termes de la suite  $u_{n+1} = 2u_n + 3$ , sachant que  $u_0 = 1$ .

2- Écrire un algorithme pour afficher les premiers termes des suites suivantes (nombre de termes ( $n$ ) demandé à l'utilisateur) :

Suite arithmétique :  $u_{n+1} = u_n + 2$ ,  $u_0 = 1$

#### Exercice 3 :

Ecrire un programme fortran qui lit les valeurs des variables :  $x$ ,  $n$ ,  $a$ ,  $b$  puis calcule les sommes et les produits suivants :

1- 
$$S = \sum_{i=1}^n (X^i / i!) = X + (X^2 / 2) + (X^3 / 6) + \dots + (X^n / n!)$$

2- 
$$P = \prod_{i=1}^n (-1)^{i-2} / X^{i+1} = ((-1)^{-1} / X^2) + ((-1)^0 / X^3) + ((-1)^1 / X^4) + \dots + ((-1)^{n-2} / X^{n+1})$$

## Travaux Pratiques 3 - Les structures itératives

---

Solutions des exercices :

Exercice 1 :

Programme fortran :

```
PROGRAM multiplication
  INTEGER I
  DO I=1,10
    WRITE(*,20)I,2*I,3*I,4*I,5*I,6*I,7*I,8*I,9*I,10*I

20 FORMAT(' ',10I5)
ENDDO
END
```

Exercice 2:

```
1- PROGRAM suite1
  INTEGER I,U,n
  U=1
  read(*,*)n
  DO I=1,n

  WRITE(*,*)U
  U=2*U+3
ENDDO
END
```

```
2- PROGRAM suite2
  INTEGER I,U,n
  U=1
  read(*,*)n
  DO I=1,n

  WRITE(*,*)U
  U=U+2
ENDDO
END
```

Exercice 3 :

```
1- PROGRAM SOMME
  INTEGER i,j,n,F
```

### Travaux Pratiques 3 - Les structures itératives

---

```
real S,X
S=0

read(*,*)n , X
DO i=1,n
F=1
DO j=1,i
F=F*j
ENDDO
S=S+((X**i)/F)
ENDDO
WRITE(*,*)S

END
```

2-

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
PROGRAM PRODUIT
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
INTEGER i,n
real S,X

S=0
read(*,*)n , X

DO i=1,n
S=S+((-1)**(i-2))/X**(i+1)
ENDDO

WRITE(*,*)'la somme S est:'

WRITE(*,*)S

END
```

## Leçon 4 : Structures de contrôle conditionnel

## Leçon 4 : Structures de contrôle conditionnel

### Objectif

Construire des algorithmes comportant des traitements conditionnels.

### 4.1 Introduction

Ces structures sont utilisées pour décider de l'exécution d'un bloc d'instruction : est-ce que ce bloc est exécuté ou non. Ou bien pour choisir entre l'exécution de deux blocs différents.

### 4.2 Expression logique

Une expression logique est un énoncée ou proposition qui peut être vraie ou fausse selon ce qu'on est entrain de parler. En mathématiques, c'est une expression contenant une ou plusieurs variables et qui est susceptible de devenir une proposition vraie ou fausse selon les valeurs attribuées à ces variables.

#### Exemple :

$(10 < 15)$  est une expression logique vrai

$(10 < 3)$  est un prédicat faux

### 4.3 Evaluation d'une expression logique

Une condition est une expression de type logique. Ils lui correspondent deux valeurs possibles VRAI et FAUX qu'on note par V ou F.

Considérons deux variables logiques A et B. Voyons quels sont les opérateurs logiques et leurs ordres de priorités.

Notons que :

$(A = \text{faux}) \Leftrightarrow \text{non } A$

$(A = \text{vrai}) \Leftrightarrow A$

Les opérateurs logiques sont :

La négation : "non"

L'intersection : "et"

L'union : "ou"

Notation et Ordre de priorité des opérateurs logiques

1. Non :  $\neg$

2. Et :  $\wedge$

3. ou :  $\vee$

#### 4.4 Tableaux d'évaluations

##### Operateurs relationnels

Opérateur	Opération
.LT. ou <	Strictement plus petit
.LE. ou ≤	Inferieur ou égal
.EQ. ou =	égal
.NE. ou ≠	Non égal
.GT. ou >	Strictement plus égal
.GE. ou ≥	Supérieur ou égal

##### Operateurs logiques

Opérateur	Opération
.NOT.	négations logique
.AND.	conjonction logique
.OR.	disjonction inclusive
.EQV.	équivalence logique
.NEQV.	non-équivalence logique

##### La négation d'une condition (A)

A	Non A
.true.	.false.
.false.	.true.

##### L'intersection de deux conditions A et B

	A et B	.true.	.false.
B			
.true.		.true.	.false.
.false.		.true.	.false.

## Leçon 4 : Structures de contrôle conditionnel

---

### L'union de deux conditions A et B

	A ou	.true.	.false.
B			
	.true.	.true.	.true.
	.false.	.true.	.false.

Nous avons deux types de structures conditionnelles :

### 4.4 Test alternatif simple

Un test simple contient un seul bloc d'instructions. Selon une condition (expression logique), on décide est-ce-que le bloc d'instructions est exécuté ou non. Si la condition Est vraie, on exécute le bloc, sinon on ne l'exécute pas. La syntaxe d'un test alternatif simple est comme suit :

#### Cas 1 :

**IF** (expression logique) **then**

Instructions ! Exécuté si la condition est vraie

**END IF**

**Exemple :** Écrire un algorithme qui demande un entier (A) à l'utilisateur, teste si ce nombre est positif ( $A \geq 0$ ) et affiche "positif".

**Solution :**

**Algorithme :**

Algorithme positif

Début

Variables A: entier

Écrire ("entrer la valeur de A ")

Lire (A)

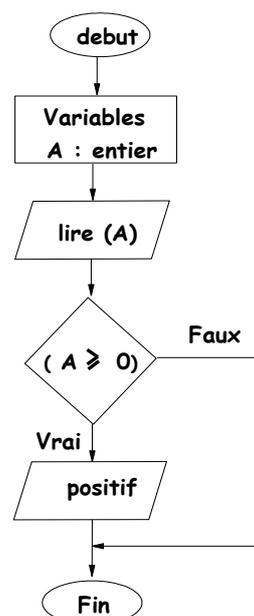
Si ( $A \geq 0$ ) alors

Écrire (" positif ")

FinSi

Fin

**Organigramme :**



## Leçon 4 : Structures de contrôle conditionnel

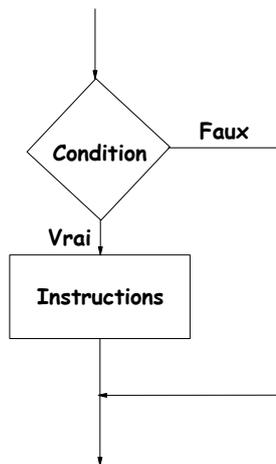
---

**Programme fortran :**

```
program positif
INTEGER A
write(*,*)' entrer la valeur de A '
read(*,*) A
IF (A.GE.0) then
write(*,*)' positif '
END IF
END
```

### Cas 2 :

**IF** (expression logique) Instruction ! Exécuté si la condition est vraie



### 4.5 Test alternatif double

Un test double contient deux blocs d'instructions : on est amené à décider entre le premier bloc ou le second. Cette décision est réalisée selon une condition (expression logique ou booléenne) qui peut être vraie ou fausse. Si la condition est vraie on exécute le premier bloc, sinon on exécute le second. La syntaxe d'un test alternatif double est comme suit :

#### Cas 1 :

**IF** (expression logique) **then**

Instructions 1

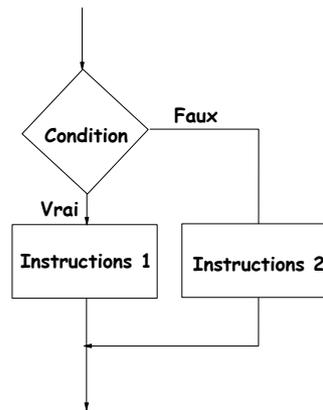
**ELSE**

Instructions 2

**END IF**

## Leçon 4 : Structures de contrôle conditionnel

---



### Exemple :

Écrire un algorithme qui demande un entier (A) à l'utilisateur, teste si ce nombre est positif ( $A \geq 0$ ) ou non, et affiche "positif" ou "négatif".

### Solution :

#### Algorithme :

Algorithme positif

Début

Variables A: entier

Écrire ("entrer la valeur de A ")

Lire (A)

Si ( $A \geq 0$ ) alors

Écrire (" positif ")

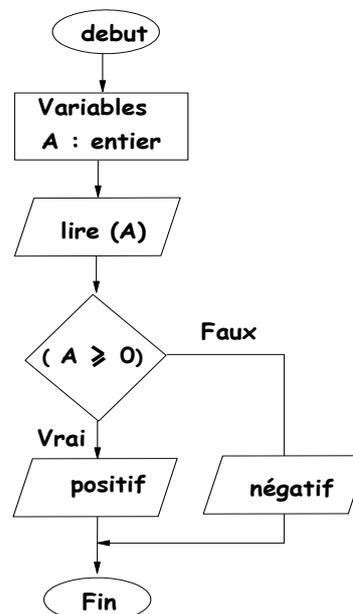
Sinon

Écrire (" négatif ")

FinSi

Fin

#### Organigramme :



#### Programme fortran :

```
program positif
```

```
INTEGER A
```

```
write(*,*) 'entrer la valeur de A '
```

```
read(*,*) A
```

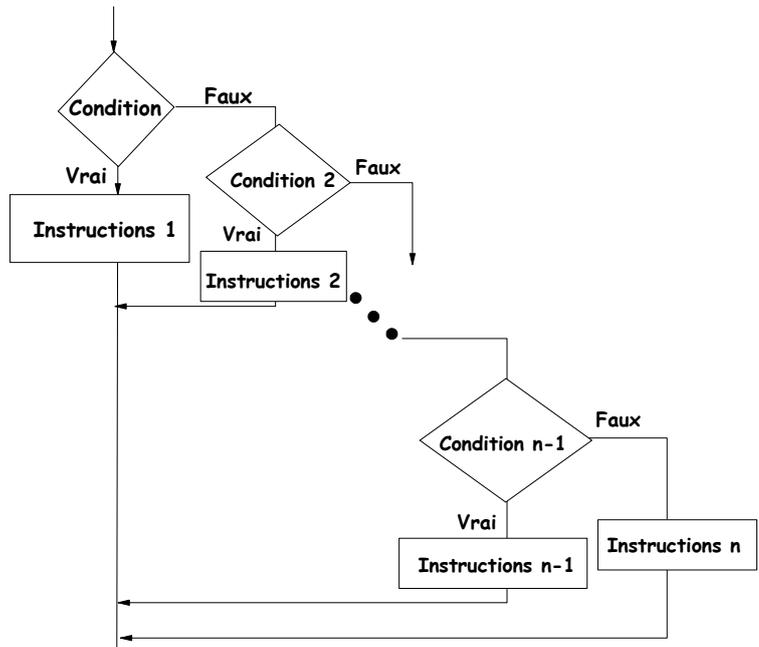
```
IF (A.GE.0) then
```

## Leçon 4 : Structures de contrôle conditionnel

```
write(*,*) ' positif '  
ELSE  
write(*,*) ' négatif '  
END IF  
END
```

### Cas 2:

```
IF (expression logique) then  
Instructions 1  
ELSEIF  
Instructions 2  
....  
ELSEIF  
Instructions n-1  
ELSE  
Instructions n  
END IF
```



### **Example:**

Écrire un algorithme qui demande un entier à l'utilisateur, teste si ce nombre est strictement positif, nul ou strictement négatif, et affiche ce résultat.

### **Solution :**

#### **Algorithme :**

Algorithme positif

Début

Variables A: entier

Écrire ("entrer la valeur de A ")

Lire (A)

Si ( $A \geq 0$ ) alors

Écrire (" positif ")

## Leçon 4 : Structures de contrôle conditionnel

Sinon

Si (A = 0)

Ecrire (" nul ")

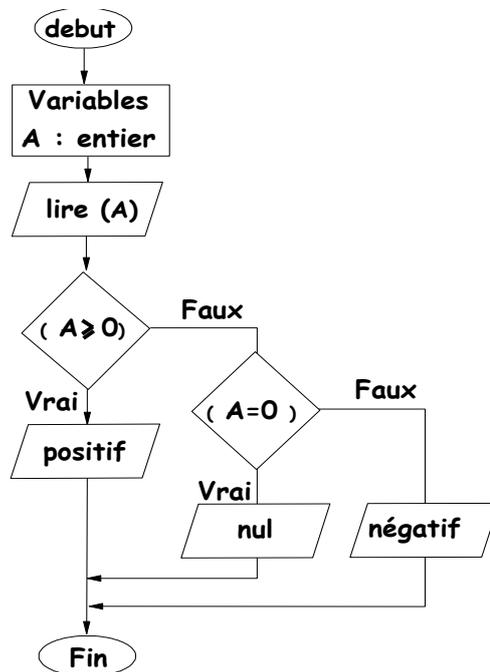
Sinon

Écrire (" négatif ")

FinSi

Fin

Organigramme :



Programme fortran :

```
program positif
INTEGER A
write(*,*)' entrer la valeur de A '
read(*,*) A
IF (A.GE.0) then
write(*,*)' positif '
ELSE
IF (A.EQ.0) then
write(*,*)' nul '
ELSE
write(*,*)' négatif '
END IF
END IF
END
```

### 4.6 TESTS IMBRIQUES

Plusieurs structures conditionnelles peuvent être imbriquées, si bien que dans une structure peut (peuvent) figurer une ou plusieurs structure(s) conditionnelle(s), et les tests peuvent avoir un degré quelconque d'imbrications.

**Exemple :**

Ecrire un algorithme qui demande à l'utilisateur la température de l'eau et afficher son état (solide, liquide, vapeur).

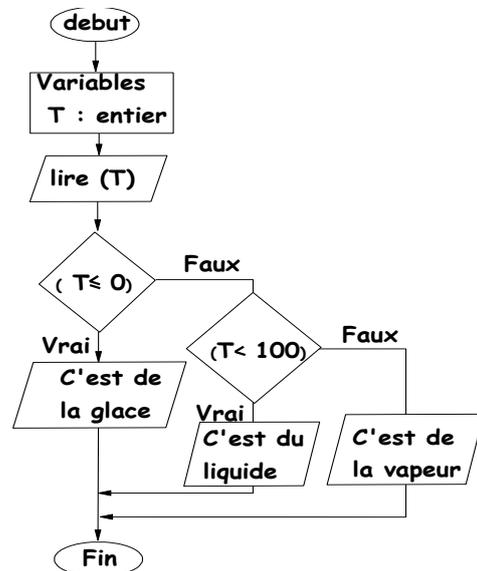
## Leçon 4 : Structures de contrôle conditionnel

**Solution :**

**Algorithme :**

```
Algo températureH2O
Variables T : réel
Début
Ecrire (entrer la température de H2O :)
Lire (T)
Si (T<= 0) Alors
Ecrire "C'est de la glace"
Sinon
Si (T< 100) Alors
Ecrire "C'est du liquide"
Sinon
Ecrire "C'est de la vapeur"
Fin si
Fin si
Fin
```

**Organigramme :**



**Programme fortran :**

```
program températureH2O
INTEGER T
write(*,*) 'entrer la valeur de T '
read(*,*) T
IF (T.LE.0) then
write(*,*) 'C'est de la glace '
ELSE
    IF (T.LT.100) then
write(*,*) 'C'est du liquide '
    ELSE
write(*,*) 'C'est de la vapeur '
    END IF
END IF
END
```

**Exercice :**

Ecrivons un algorithme qui précise si un triangle  $ABC$  est rectangle (et dans ce cas en quel point) ou s'il ne l'est pas.

**Remarque :** D'après la réciproque du théorème de Pythagore, si  $AB^2=AC^2+BC^2$ , alors le triangle  $ABC$  est rectangle en  $C$ .

## Leçon 4 : Structures de contrôle conditionnel

---

### Solution :

Algo rectangle

Variables  $x_A, y_A, x_B, y_B, x_C, y_C, AB_{\text{carre}}, AC_{\text{carre}}, BC_{\text{carre}}$  : réel

Début

Ecrire (Saisir l'abscisse et l'ordonnée des points A, B et C)

Lire ( $x_A, y_A, x_B, y_B, x_C, y_C$ )

$AB_{\text{carre}} = (x_B - x_A)^2 + (y_B - y_A)^2$

$AC_{\text{carre}} = (x_C - x_A)^2 + (y_C - y_A)^2$

$BC_{\text{carre}} = (x_C - x_B)^2 + (y_C - y_B)^2$

Si ( $AB_{\text{carre}} + BC_{\text{carre}} = AC_{\text{carre}}$ ) Alors

Ecrire "le triangle ABC est rectangle en B"

sinon

Si ( $AB_{\text{carre}} + AC_{\text{carre}} = BC_{\text{carre}}$ ) Alors

Ecrire "le triangle ABC est rectangle en A"

Sinon

Si ( $AC_{\text{carre}} + BC_{\text{carre}} = AB_{\text{carre}}$ ) Alors

Ecrire "le triangle ABC est rectangle en C"

Sinon

Ecrire "le triangle n'est pas rectangle"

finsi

fin

### Programme fortran :

```
program rectangle
```

```
real xA,yA, xB,yB xC,yC,ABcarre,ACcarre,BCcarre
```

```
write(*,*) Saisir l'abscisse et l'ordonnée des points A, B et C'
```

```
read(*,*) xA,yA, xB,yB xC,yC
```

```
ABcarre= (xB -xA)^2+(yB-yA) ^2
```

```
ACcarre= (xC -xA) ^2+(yC-yA) ^2
```

```
BCcarre= (xC -xB) ^2+(yC-yB) ^2
```

```
IF (ABcarre+ BCcarre .EQ.ACcarre) then
```

```
write(*,*)le triangle ABC est rectangle en B'
```

```
ELSE
```

```
IF (ABcarre+ ACcarre .EQ.BCcarre) then
```

```
write(*,*)le triangle ABC est rectangle en A'
```

```
ELSE
```

```
IF (ACcarre+ BCcarre .EQ.ABcarre) Alors
```

```
write(*,*)le triangle ABC est rectangle en C'
```

```
ELSE
```

```
write(*,*)le triangle n'est pas rectangle'
```

```
ENDIF
```

```
END
```

## Travaux Dirigés 4 - Structures de contrôle conditionnel

---

### Exercice 1 :

Écrire un algorithme puis un organigramme permettant de trouver et d'afficher la plus petite valeur entre trois nombres réels distincts A, B et C

### Exercice 2 :

Écrire un algorithme qui demande un réel à l'utilisateur et affiche sa valeur absolue (sans utiliser de fonction prédéfinie évidemment).

### Exercice 3 :

Écrire un algorithme qui demande à l'utilisateur d'entrer la note et qui affiche la mention comme suite :

Faible ; si  $note \leq 10$

Passable ; si  $10 < note \leq 12$

A. Bien ; si  $12 < note \leq 14$

Bien ; si  $14 < note \leq 16$

T. Bien ; si  $16 < note \leq 18$

Excellent ; si  $18 < note \leq 20$

### Exercice 4 :

Écrire un algorithme permettant de résoudre une équation du second degré.

### Exercice 5 :

Écrire un algorithme permettant de :

- Lire un nombre fini de notes comprises entre 0 et 10
- Afficher la meilleure note, la mauvaise note et la moyenne de toutes les notes.

### Exercice 6 :

**Nombres parfaits** Un nombre parfait est un nombre présentant la particularité d'être égal à la somme de tous ses diviseurs, excepté lui-même. Le premier nombre parfait est  $6 = 3 + 2 + 1$ . Écrire un algorithme qui affiche tous les nombres parfaits inférieurs à 500.

## Travaux Dirigés 4 - Structures de contrôle conditionnel

---

Solutions des exercices :

Exercice 1:

Algorithme :

Algo petite

Début

Variables A, B, C : réel

Lire (A, B, C)

Si (A<B) alors

    Si (A<C) alors

        Ecrire A

    Si non

        Ecrire C

    Fin Si

Sinon

    Si (B<C) alors

        Ecrire B

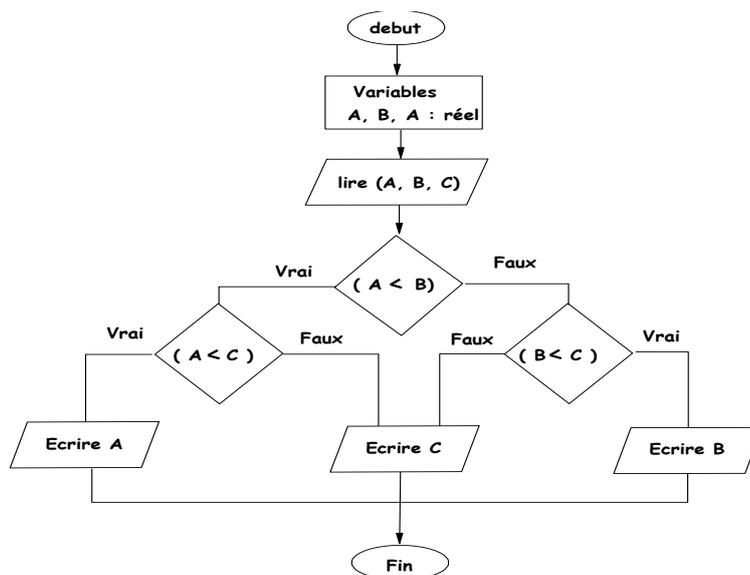
    Si non

        Ecrire C

    Fin Si

Fin

Organigramme :



### Exercice 2:

```
algo absolue
Début
Réal a
Lire "tapez un réel", a
Si (a>0) alors
Écrire "|", a,"|=", a
Sinon
Écrire "|", a,"|=", -a
Fin si
Fin
```

### Exercice 3 :

```
Algo mention
Variable note : réel
Début
Ecrire (entrer la note :)
Lire (note)
Si (note <10) alors

Ecrire (faible)
Si non
  Si (note <12) alors
    Ecrire (passable)
  Si non Si (note <14) alors
    Ecrire (A. Bi en)
  Si non Si (note <16) alors
    Ecrire (Bien)
  Si non Si (note <18) alors
    Ecrire (T. Bien)
  Si non
    Ecrire (Excellent)

Fin Si

Fin
```

### Exercice 4 :

```
Algo second_degré
Début
Réal a, b, c, delta
```

## Travaux Dirigés 4 - Structures de contrôle conditionnel

---

Début : Ecrire (saisissez les valeurs a, b et c de l'équation  $ax^2+bx+c=0$  )  
Lire (a, b, c)  
Si (a=0) alors  
Écrire (équation du premier degré)  
Sinon  
Delta= $b^2-4*a*c$   
Si (delta>0) alors  
Écrire (les solutions de l'équation sont)  
Écrire  $(-b-\sqrt{\text{delta}})/(2*a)$ ,  $(-b+\sqrt{\text{delta}})/(2*a)$   
Sinon Si (d=0) alors  
Écrire  $(-b/ (2a))$   
Sinon  
Écrire (pas de solutions réelles)  
Fin Si  
Fin Si  
Fin

### Exercice 5 :

Algorithme Notes  
Variables n, i : Entier  
note, min, max, s : Réel  
Début  
Ecrire ("Entrer le nombre de notes : ")  
Lire(n) (On suppose que n est toujours supérieur à zéro)  
s =0  
max =0  
min =10  
Pour i de 1 à n  
Ecrire ("Entrer une note : ")  
Lire (note)  
s =s + note  
Si (note > max) Alors  
max =note  
Fin Si  
Si (note < min) Alors  
min =note  
Fin Si  
Fin Pour  
Ecrire ("Meilleure note = ", max)  
Ecrire ("Mauvaise note = ", min)  
Ecrire ("Moyenne des notes = ", s/n)  
Fin.

### Exercice 6 :

```
Algorithme parfaits
Variables n, d, S : entier
Début
Ecrire ("Entrez la valeur de n : ")
Lire (n)
S = 1
d=2
Tant Que (d*d <= n)
Si (mod(n,d)=0) alors
S = S + d + n/d
Fin si
d =d + 1
Fin Tant Que
Si (S=n) alors
Ecrire ("le nombre n est parfait ")
Sinon
Ecrire ("le nombre n n'est pas parfait ")
Fin si
Fin
```

### Programme fortran :

```
Program parfaits
INTEGER n, d, S
WRITE (*,*)"Entrez la valeur de n : "
READ (*,*) n
Do i=1, n
S = 1
d=2
Do while (d*d.LE.i)
IF (mod(i,d).EQ.0) then
S = S + d + i/d
endif
d=d+1
enddo
IF (S.EQ.i) then
WRITE (*,*)"le nombre",i," est parfait "
endif
enddo
End
```

## Leçon 5 : Les tableaux

## Leçon 5: Les tableaux

### Objectif

Développer des algorithmes de traitement de tableaux.

### 5.1 Définition

Un tableau est un ensemble d'éléments de même type, repérés au moyen d'indices entiers. Les éléments d'un tableau sont rangés selon un ou plusieurs axes appelés dimensions du tableau. Dans les tableaux à une dimension (qui permettent de représenter par exemple des vecteurs au sens mathématique du terme), chaque élément est repéré par un seul entier, mais le fortran accepte des tableaux jusqu'à 7 dimensions, où chaque élément est désigné par un 7 indices.

Un tableau à une dimension est parfois appelé vecteur. Il peut être représenté sous la forme suivante :

L(1)	L(2)	L(3)	L(4)	.....		L(n)
------	------	------	------	-------	--	------

- Dimension du tableau : 1
- Taille du tableau : n
- Les  $L(i)$ ,  $i=1, 2, \dots, n$  doivent être de même type

### 5.2 Déclaration des tableaux

Comme pour les variables simples se pose le problème du type du tableau, c'est-à-dire de ses éléments. Tous les éléments du tableau sont de même type. Si le nom du tableau apparaît dans un ordre type (REAL, INTEGER, ....) le problème est résolu. Sinon c'est que le tableau est apparu dans un ordre DIMENSION, et la première lettre lève toute confusion, si cette lettre est I, j, K, L, M ou n le tableau est entier, sinon réel.

#### Exemples:

REAL A (10), B (15, 5), I, C (3, 7, 9)

INTEGER X, Y (0:3)

DIMENSION Z (7, 3), K (12)

Ces déclarations indiquent :

A est un tableau réel à 1 dimension, de 10 éléments

B est un tableau réel à 2 dimensions, de 15 lignes et 5 colonnes.

I est une variable simple réelle

J est un tableau réel à 3 dimensions égales à 3, 7 et 9

X est une variable simple entière

Y est un tableau entier à 1 dimension de 4 éléments

## Leçon : 5 Les tableaux

---

Z est un tableau réel à 2 dimensions, de 7 lignes et 3 colonnes

K est un tableau entier à 1 dimension de 12 éléments

### 5.3 Terminologie des tableaux

- rang d'un tableau : nombre de ses dimensions
- étendue (extent) d'un tableau selon une de ses dimensions : nombre d'éléments dans cette dimension
- bornes (bounds) d'un tableau selon une de ses dimensions : limites inférieure et supérieure des indices dans cette dimension. La borne inférieure par défaut vaut 1.
- profil (shape) d'un tableau : vecteur dont les composantes sont les étendues du tableau selon ses dimensions ; sa taille est le rang du tableau.
- taille (size) d'un tableau : nombre total des éléments qui le constituent, c'est-à-dire le produit des éléments du vecteur que constitue son profil.
- deux tableaux sont dits conformants (conformable) s'ils ont le même profil La déclaration d'un tableau s'effectue grâce à l'attribut DIMENSION qui indique le profil du tableau, mais aussi éventuellement les bornes, séparées par le symbole « : ».

### Exemples :

REAL, DIMENSION X(15)

REAL, DIMENSION Y (1:5,1:3)

REAL, DIMENSION Z (-1:3,0:2)

Le tableau X est de rang 1, Y et Z sont de rang 2.

L'étendue de X est 15, Y et Z ont une étendue de 5 et 3.

Le profil de X est le vecteur (/ 15 /), celui de Y et Z est le vecteur (/ 5,3 /).

La taille des tableaux X, Y et Z est 15.

Les tableaux Y et Z sont conformants.

### 5.4 Manipulation d'un tableau

Une fois déclaré, un tableau peut être utilisé comme un ensemble de variables simples. Les trois manipulations de base sont l'affectation, la lecture et l'écriture.

#### a- L'affectation :

Pour affecter une valeur à un élément i d'un tableau nommé par exemple A, on écrira :  
 $A(i) \leftarrow \text{valeur}$ .

Par exemple, l'instruction :  $A(0) \leftarrow 20$  ; affecte au premier élément du tableau A la valeur 20.

Pour affecter la même valeur à tous les éléments d'un tableau A de type numérique et de dimension 100, on utilise une boucle.

#### Exemple :

Pour  $i=1$  à 100

## Leçon : 5 Les tableaux

---

$A(i) \leftarrow 0$  ;  
Fin pour

Cette boucle permet de parcourir le tableau A élément par élément et affecter à chacun la valeur 0. La variable i est appelée indice.

### b- La lecture :

Comme les variables simples, il est possible aussi d'assigner des valeurs aux éléments d'un tableau lors de l'exécution c.-à-d. les valeurs sont saisies par l'utilisateur à la demande du programme.

Exemple :

Écrire "Enter une note :"

Lire A(6)

Dans cet exemple, la valeur saisie est affectée au sixième (6ème) élément du tableau A.

### c- L'écriture :

De façon analogue à la lecture, l'écriture de la valeur d'un élément donné d'un tableau s'écrira comme suit : écrire A(i) Cette instruction permet d'afficher la valeur de l'élément i du tableau A.

### Exemple :

Ecrire un algorithme qui déclare et remplisse un tableau de 7 valeurs numériques réels en les mettant toutes à zéro.

### Solution :

#### Algorithme :

Algorithme remplissage

Tableau A(7) : réel

Variable i : entier

Début

Pour i = 1 à 7

A(i) = 0

Fin Pour

Fin

### Programme Fortran:

Program remplissage

Dimension A (7)

INTEGER I

Do I=1, 7

A(i) = 0

## Leçon : 5 Les tableaux

---

Enddo  
END

### Exemple :

Ecrire une procédure qui recherche dans un vecteur V de dimension N, l'indice de la composante contenant la valeur X. Considérer d'abord le cas où l'on est sûr que X se trouve dans le vecteur, puis le cas général.

### Solution :

#### Algorithme :

Algorithme Indice

Tableau V(N) : réel

Variable I, N, Indice : entier

Début

Lire (N)

Pour I = 1 à N

Si (V(I)=X) alors

Indice = I

Fin Si

Fin Pour

Ecrire (l'indice de la composante contenant la valeur X est), Indice

Fin

### Programme Fortran:

Program Indice

Dimension V (N)

INTEGER I, N, Indice

Read (\*,\*) N

Do I=1, N

IF (V (I).EQ.I) then

Indice = I

ENDIF

ENDDO

write(\*,\*) 'l'indice de la composante contenant la valeur X est', Indice

END

## 5.5 Tri d'un tableau

On présente quelques algorithmes utiles, qui permettent d'ordonner les éléments d'un tableau dans un ordre croissant ou décroissant. L'ordre est par défaut croissant. Un vecteur est dit trié si  $V(i) \leq V(i+1)$ ,  $\forall i \in [1..n-1]$ .

## Leçon : 5 Les tableaux

---

Il existe plusieurs stratégies possibles pour trier les éléments d'un tableau ; nous en verrons deux : le tri par sélection, et le tri à bulles. Champagne !

### 5.5.1 Tri par sélection

La technique du tri par sélection est la suivante :

On met en bonne position l'élément le plus petit. Puis on met en bonne position l'élément suivant. Et ainsi de suite jusqu'au dernier. Par exemple, si on a un tableau de 10 éléments et on veut que le trier dans l'ordre croissant.

45	122	12	3	21	78	64	53	89	28
----	-----	----	---	----	----	----	----	----	----

On commence par rechercher, parmi les 10 valeurs, quel est le plus petit élément, et où il se trouve. On l'identifie en quatrième position (c'est le nombre 3), et on l'échange alors avec le premier élément (le nombre 45). Le tableau devient ainsi :

3	122	12	45	21	78	64	53	89	28
---	-----	----	----	----	----	----	----	----	----

On recommence à chercher le plus petit élément, mais cette fois, seulement à partir du deuxième (puisque le premier est maintenant correct, on n'y touche plus). On le trouve en troisième position (c'est le nombre 12). On échange donc le deuxième avec le troisième :

3	12	122	45	21	78	64	53	89	28
---	----	-----	----	----	----	----	----	----	----

On recommence à chercher le plus petit élément à partir du troisième (puisque les deux premiers sont maintenant bien placés), et on le place correctement, en l'échangeant, ce qui donnera in fine :

3	12	21	45	122	78	64	53	89	28
---	----	----	----	-----	----	----	----	----	----

Et répète l'opération jusqu'à l'avant dernier.

## Leçon : 5 Les tableaux

---

Donc on peut écrire le processus de la manière suivante :

Algo tri

Début

Entier i, Posmini, j

Réel temp

Dimension t(100)

Pour i=0, 10

Lire t(i)

Fin Pour

Pour i=0, 10

posmini = i

Pour j=i + 1, 11

Si (t(j) < t(posmini)) Alors

Posmini= j

Fin si

Fin Pour

Temp=t(posmini)

t(posmini)=t(i)

t(i)=temp

Fin Pour

Fin

### 5.5.2 Tri à bulles

L'idée de départ du tri à bulles consiste à se dire qu'un tableau trié en ordre croissant, c'est un tableau dans lequel tout élément est plus petit que celui qui le suit. En effet, prenons chaque élément d'un tableau, et comparons-le avec l'élément qui le suit. Si l'ordre n'est pas bon, on permute ces deux éléments. Et on recommence jusqu'à ce que l'on n'ait plus aucune permutation à effectuer. Les éléments les plus grands remontent ainsi peu à peu vers les dernières places. En fait, tout ce qu'on peut dire, c'est qu'on devra effectuer le tri jusqu'à ce qu'il n'y ait plus d'éléments qui soient mal classés.

Algo Classement

Début

Variable logique Yapermute

Dimension t(100)

Pour i=0, 10

Lire t(i)

## Leçon : 5 Les tableaux

---

Fin Pour

Yapermut=Vrai

TantQue (Yapermut) alors

Yapermut=Faux

Pour i=0, 10

Si ( $t(i) > t(i+1)$ ) alors

Temp=t(i)

t(i)=t(i+1)

t(i+1)=temp

Yapermut=Vrai

Finsi

Fin Pour

FinTantQue

Fin

### Exercice 1 :

Que produit l'algorithme suivant ?

Algo Carre

Début

Tableau Nb(5) en Entier

Variable i en Entier

Pour i = 0, 5

Nb(i)  $\leftarrow$  i \* i

Fin Pour

Pour i = 0, 5

Ecrire Nb(i)

Fin Pour

Fin

Peut-on simplifier cet algorithme avec le même résultat ?

### Exercice 2 :

Écrivez un algorithme permettant, à l'utilisateur de saisir les notes d'une classe. Le programme, une fois la saisie terminée, renvoie le nombre de ces notes supérieures à la moyenne de la classe.

### Exercice 3:

Que font les algorithmes suivants ?

a)

Var i, n (10) entier

Debut

n [0]  $\leftarrow$  1

Pour i allant de 1 a 9

Faire

N(i)  $\leftarrow$  n (i-1) + 2

Fin

b)

Var i, n [10] entier

Début

Pour i allant de 0 a 9

Faire

N(i)  $\leftarrow$  2 \* i

Fin

## Travaux Dirigés 5 - Les tableaux

---

### Exercice 4:

Ecrire un algorithme permettant de résoudre le problème suivant :

- Données : un tableau contenant 100 entiers
- Résultat : "vrai" si le tableau est trié du plus petit au plus grand et "faux" sinon

### Exercice 5:

Ecrire un algorithme permettant de saisir 100 valeurs et qui les range au fur et à mesure dans un tableau.

### Exercice 6:

Ecrire un algorithme qui calcule le plus grand écart dans un tableau d'entiers. Rappel : l'écart entre deux entiers  $x$  et  $y$  est la valeur absolue de leur différence  $|x - y|$ .

### Solutions des exercices :

#### Exercice 1 :

Cet algorithme remplit un tableau avec le carré de six valeurs : 0, 1, 2, 3, 4, 5.  
Il les écrit ensuite à l'écran.

Simplification :

Algo Carre

Début

Tableau Nb(5) en Entier

Variable i en Entier

Pour i = 0, 5

Nb(i)  $\leftarrow$  i \* i

Ecrire Nb(i)

Fin Pour

Fin

#### Exercice 2 :

Algo classe

Début

Tableau T(Nb)

Variable i, Nb, Nbsup en Entier

Variable Som, Moy en réel

Ecrire "Entrez le nombre de notes à saisir : "

Lire (Nb)

Pour i = 1, Nb

Ecrire "Entrez le nombre numéro", i

Lire T(i)

Fin Pour

Som = 0

Pour i = 1, Nb

Som = Som + T(i)

Fin Pour

Moy = Som / Nb

NbSup = 0

Pour i = 1, Nb

Si (T(i) > Moy) Alors

NbSup = NbSup + 1

FinSi

Fin Pour

Ecrire NbSup, "élèves dépassent la moyenne de la classe"

Fin

## Travaux Dirigés 5 - Les tableaux

---

### Exercice 3:

- a) Initialisation d'un tableau avec les 10 premiers nombres impairs
- b) Initialisation d'un tableau avec les 10 premiers nombres pairs

### Exercice 4:

Algorithme Test

Début

Entier i

Tableau T (100)

Logique Trié

Trié = vrai

i = 0

Tant que ((Trié = vrai) et (i < 99)) alors

Trié = (T(i) ≤ T(i+1))

i = i + 1

Fin tant que

Ecrire Trié

Fin

### Exercice 5:

Algorithme Exercice

Entier i, j, x

Tableau Entier T1 (100)

Logique positionné

Début

Pour i=0, 100

Afficher "Entrez votre valeur : "

Lire x

j = i

Tant que ((j > 0) et (T1(j-1) > x)) alors

T1(j) = T1(j-1)

j = j - 1

T1(j) = x

Fin

### Exercice 6:

Algorithme écart

## Travaux Dirigés 5 - Les tableaux

---

```
Entier tableau(n), i, Min, Max
Début
Pour i=0, n -1
Lire tableau(i)
Fin Pour
Min = tableau(i)
Max = tableau (i)
Pour i=0, n -1
Si (tableau(i)) < min) alors
Min = tableau (i)
Fin si
Si (tableau(i)> max) alors
Max = tableau(i)
Fin si
Ecrire (max - min)
Fin
```

### Exercice 1 :

Ecrire un algorithme qui inverse l'ordre d'un tableau des 100 entiers triés. En d'autres termes, si le tableau est trié du plus petit au plus grand, alors l'algorithme retourne le tableau trié du plus grand au plus petit ; réciproquement, si le tableau est trié du plus grand au plus petit, alors l'algorithme retourne le tableau trié du plus petit au plus grand.

### Exercice 2 :

1- Que produit l'algorithme suivant ?

Tableau Nb(5) en Entier

Variable i en Entier

Début

Pour i ← 0 à 5

Nb(i) ← i \* i

Fin pour

Pour i ← 0 à 5

Ecrire Nb(i)

Fin pour

Fin

2- Peut-on simplifier cet algorithme avec le même résultat ?

3- Traduire l'algorithme en programme fortran puis exécuter et affichés l'état de l'écran.

### Exercice 3:

- 1- Ecrire un programme fortran qui effectue la lecture d'une matrice carrée A ainsi que sa taille n et affiche la trace de A (pour une matrice  $A(a_{i,j})$ ,  $\text{Trace}(A) = \sum a_{i,i}$  la somme des éléments sur la diagonale).
- 2- Ecrire un programme fortran qui effectue la lecture d'une matrice carrée A ainsi que sa taille n et affiche la matrice transposée tA de A (Pour une matrice  $A(a_{i,j})$ ,  $tA(a_{j,i})$ ).

## Travaux Pratiques 5 - Les tableaux

---

### Solutions des exercices :

#### Exercice 1 :

Algorithme Inversion tableau

Variables entier i, n

Tableau T [100]

Début

Lire n

Pour i de 0 à n

Lire T(i)

Fin

Pour i de 0 à n/2

C=T(i)

T(i) = T (n -i -1)

T (n -i -1) =c

Fin pour

Fin

#### Exercice 2 :

1- Cet algorithme remplit un tableau avec six valeurs : 0, 1, 4, 9, 16, 25. Il les écrit ensuite à l'écran.

2- Simplification :

Tableau Nb(5) en Numérique

Variable i en Numérique

Début

Pour i = 0 à 5

Nb(i) = i \* i

Ecrire Nb(i)

Fin pour

Fin

3-

```
c=====
c=                                     =
      PROGRAM EXO2
c=                                     =
c=====
      INTEGER I
      DIMENSION NB(10)
      DO I=1,5
      NB(i)=i*i
      WRITE(*,*)NB(i)
```

## Travaux Pratiques 5 - Les tableaux

---

```
        ENDDO
C=====
        END
```

Exécution du programme :

```
1
4
9
16
25
Press any key to continue
```

Etat de l'écran

Exercice 3:

```
1-
C=====
C=                                     =
    PROGRAM matrice
C=                                     =
C=====

    INTEGER I,J,n
    DIMENSION A(100,100)
    WRITE(*,*)'Donner la taille de la matrice A est n='
    read(*,*) n
    TRACE_A=0
    WRITE(*,*)'Donner les éléments de la matrice A '
    DO I=1,n
    DO J=1,n

    read(*,*) A(i,j)
    ENDDO
    ENDDO
    DO I=1,n
    TRACE_A=TRACE_A+A(i,i)
    ENDDO
    WRITE(*,*)'La trace de la matrice A est:',TRACE_A
C=====
    END
```

## Travaux Pratiques 5 - Les tableaux

---

2-

```
C=====
C=                                     =
    PROGRAM transposé
C=                                     =
C=====
    INTEGER I,J,n
    DIMENSION A(100,100),TA(100,100)

WRITE(*,*)'Donner la taille de la matrice A est n='
read(*,*) n

WRITE(*,*)'Donner les éléments de la matrice A '

    DO I=1,n
    DO J=1,n
    read(*,*) A(i,j)
    ENDDO
    ENDDO
    DO I=1,n
    DO J=1,n

    TA(i,j)=A(j,i)
    WRITE(*,*)TA(i,j)

    ENDDO
    ENDDO
C=====
    END
```