

Module : Algorithmique et Bases de Programmation

Filière : Génie Informatique, Semestre 1

Descriptif du module

- ✓ **Intitulé du module:** Algorithmique et Bases de Programmation.
- ✓ **Objectifs du module:**
 - Apprendre les bonnes techniques de programmation en écrivant des algorithmes
 - Maîtriser les bases du langage C.
- ✓ **Les éléments du module:**
 1. Algorithmique
 2. Les bases de programmation
- ✓ **Enseignements et/ou activités :** Cours, Travaux dirigés, Travaux pratiques, évaluations, ...)
- ✓ **Mode d'évaluation:**
 1. Contrôles continus: Devoirs surveillés + évaluation des TP
 2. Examen de TP, Examen final

Descriptif du module

✓ **Note du module:**

1. Algorithmique : 1 DS (50 %), Examen final (50 %)
2. Les bases de programmation : 1 DS (30 %), évaluation séance tenante + projets (30 %), Examen TP (40 %)

1. Algorithmique : 50 %
 2. les bases de programmation : 50 %
- Moyenne du module = $0,5 * (\text{Note Algorithmique}) + 0,5 * (\text{Note les bases de programmation})$

Descriptif du module

✓ Modalités de validation du module

- Le module est acquis par validation si sa note **est supérieure ou égale à 12 sur 20** sans qu'aucune note des éléments le composant ne soit **inférieure strictement à 6 sur 20**.
- Si la moyenne du module est **supérieure ou égale à 8 sur 20** sans qu'aucune note des éléments le composant ne soit **inférieure strictement à 6 sur 20** peut être acquis par compensation à la fin de l'année universitaire en considérant **tous les modules des deux semestres de l'année**.

Algorithmique et Bases de Programmation



Algorithmique et Bases de Programmation

□ Planning et plan

- Début des cours magistraux: **semaine du 04 octobre**
- Début des travaux dirigés et pratiques: **semaine du 18 octobre**

Algorithmique et Bases de Programmation

❑ But et motivation

Être capable de :

- ❑ Comprendre un algorithme et expliquer ce qu'il fait.
- ❑ Savoir transcrire les différentes étapes de résolution d'un problème sous forme d'algorithme.
- ❑ Construire un algorithme simple, le programmer et comprendre comment se déroule son exécution.
- ❑ Modifier un algorithme existant pour obtenir des résultats améliorés.
- ❑ Développer des programmes (applications) informatiques qui soient:
 - ✓ Correctes
 - ✓ Simples
 - ✓ Efficaces
 - ✓ Rapides

Plan de cours

1 Généralité sur l'informatique

- ✓ Notions: Informatique, Données, Ordinateur, Système d'exploitation

2 Algorithmique

- ✓ Chapitre 1: Les éléments de base d'un algorithme
- ✓ Chapitre 2: Les structures conditionnelles et répétitives
- ✓ Chapitre 3: Les tableaux
- ✓ Chapitre 4: Les structures
- ✓ Chapitre 5: Les fonctions
- ✓ Chapitre 8: Les algorithmes de tri et de recherche

Plan de cours

3 Programmation « Langage C »

Chapitre 1: Introduction à la programmation

- I. Algorithmique et Programmation
- II. Langage de programmation
- III. Compilation et exécution d'un programme en C

Chapitre 2: La programmation sous C

- I. Les composants élémentaires du C
- II. Types de base, variables et constantes
- III. L'affectation (Assignment)
- IV. Les opérateurs et expressions
- V. Les entrées/sorties en C
- VI. Les instructions sélectives (alternatives)
- VII. Les instructions itératives (répétitives)
- VIII. Les tableaux
- IX. Les fonctions/procédures
- X. Les pointeurs
- XI. Les chaînes de caractères
- XII. Les structures

Partie 1: Généralité sur l'informatique

Généralité sur l'informatique

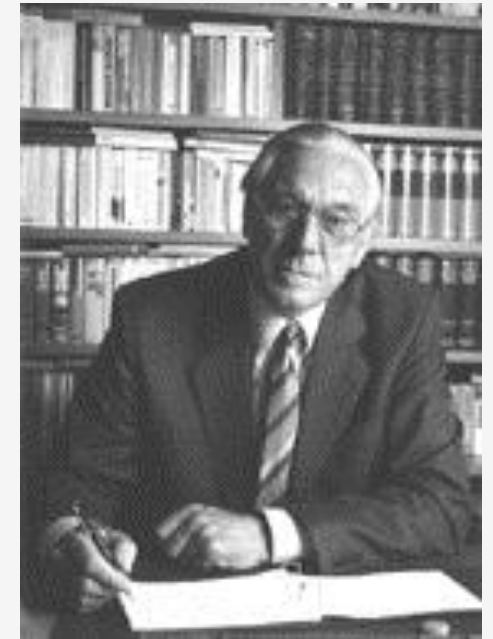
□ Définition

D'où vient le mot **informatique**?

le mot informatique fait sa première apparition en 1957 sous la plume d'un ingénieur allemand, nommé **Karl Steinbuch**. Ce dernier, est un auteur d'un ouvrage sur le traitement automatisé de données.

Informatique = **Information** + **Automatique**

→ **Définition:** *L'informatique est la science du traitement automatique de l'information.*



Généralité sur l'informatique

□ Personnalités



Charles Babbage: Il fut le premier à énoncer l'idée et le principe d'un ordinateur.



Steve Jobs: Cofondateur de la société APPLE, il a inventé l'interface graphique des systèmes d'exploitation. Il a également révolutionné le monde mobile grâce notamment au iPhone.



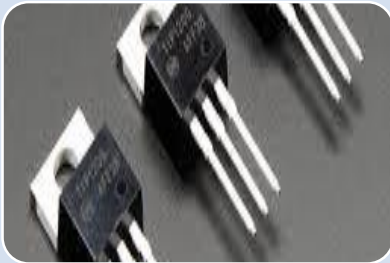
Bill Gates: Il est le fondateur de Microsoft. Il a inventé le système d'exploitation le plus utilisé de nos jours (Microsoft Windows).



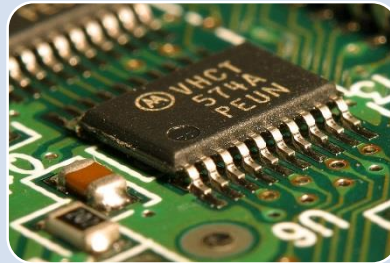
Tim Bernes Lee: est un informaticien britannique, celui qui a inventé le www qui est à la base de l'internet que nous connaissons aujourd'hui.

Généralité sur l'informatique

❑ Essor



Invention du transistor (1947):
L'invention du transistor.



1958: C'est l'invention du circuit intégré.



Invention du microprocesseur: En 1969, E. Hoff, ingénieur à Intel inventa le premier microprocesseur Intel 4004 commercialisé, le 15 novembre 1971.



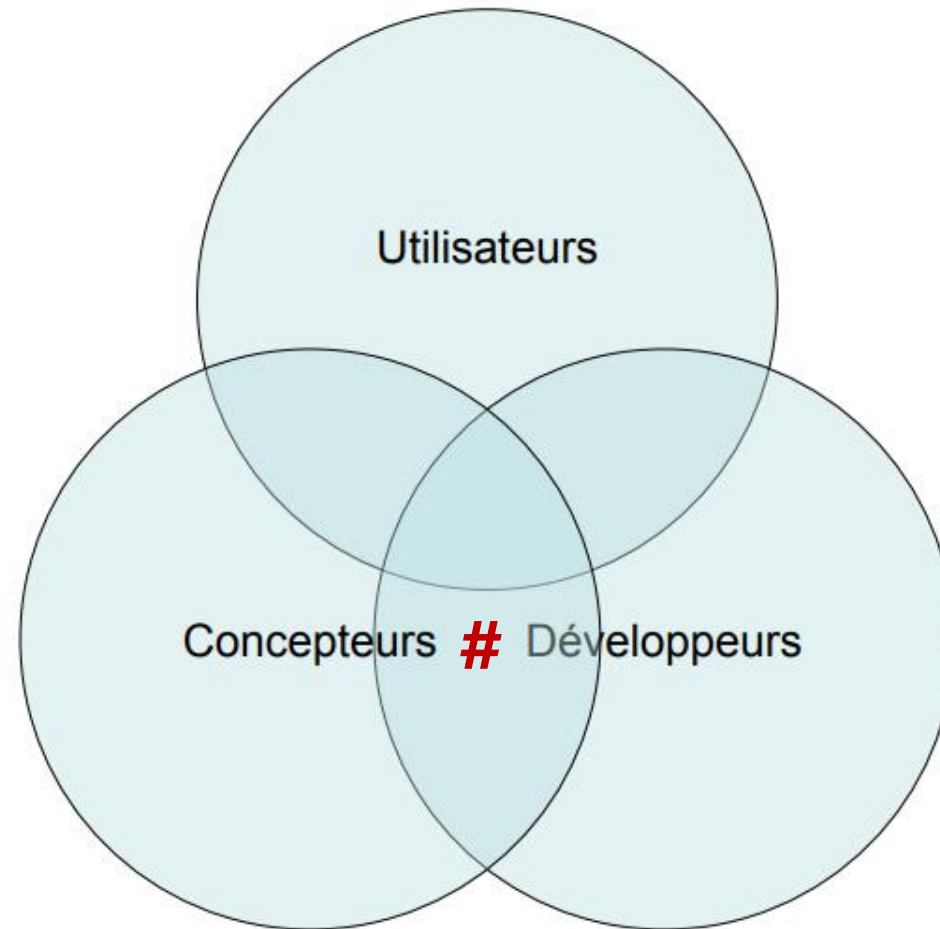
L'ordinateur personnel (PC): c'est un ordinateur destiné à l'usage d'une personne, de prix accessible et dont les dimensions sont assez réduites pour tenir sur un bureau. La première machine appelée micro-ordinateur est le Micral N, breveté en 1973 par le Français François Gernelle.

ont permis de nouvelles possibilités de miniaturisation et de puissance de calcul.

Généralité sur l'informatique

□ Usagers de l'informatique

- **Concepteurs:** un concepteur est une personne qui est chargée d'identifier les besoins des utilisateurs et de les spécifier. Son rôle consiste en particulier à expliquer les concepts à des experts non informaticiens.
- **Développeurs:** un développeur (programmeur) est un informaticien qui réalise des logiciels et les met en œuvre à l'aide de langages de programmation.
- **Utilisateurs:** Une personne qui utilise un système informatisé (ordinateur ou robot) mais qui n'est pas nécessairement informaticien.



Généralité sur l'informatique

❑ Technologie de l'information

Formats de données ??

- 
- 1 Nombres ✓ prix, poids, volume, température, vitesse, pression...
 - 2 Textes ✓ courrier, publications, articles de presse...
 - 3 Images ✓ plans, dessins, graphiques, diagrammes, cartes géographiques, photos, images 3D...
 - 4 Sons ✓ paroles, audios, bruitages ou musique.
 - 5 Vidéo ✓ prises de vue, clips ou films.

Généralité sur l'informatique

❑ Données / Informations

- Il existe une différence subtile entre les données et les informations. Les données sont les faits ou les détails à partir desquels l'information **est dérivée**.
- Les données individuelles sont rarement utiles seules. Pour que les données deviennent des informations, elles doivent être **mises en contexte**.

❑ Donnée :

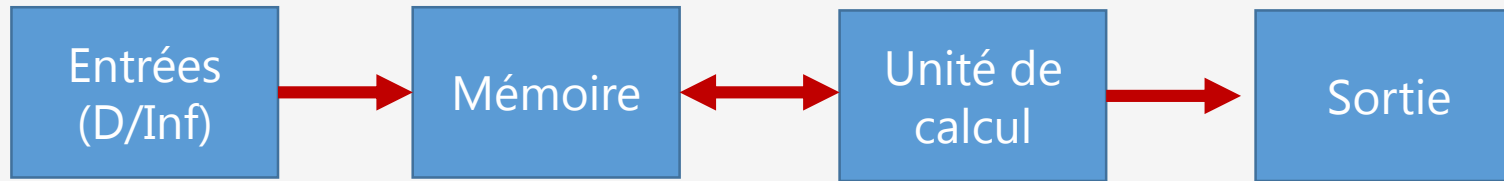
- Les données sont des faits bruts et non organisés qui doivent être traités.
- Les données sont toujours interprétées, par un homme ou une machine, pour en tirer un sens.
- Les données contiennent des chiffres, des énoncés et des caractères sous forme brute.

❑ Information :

- Lorsque des données sont **traitées, organisées, structurées** ou présentées dans un contexte donné afin de les rendre utiles, on les appelle des **informations**.
- Lorsque les données sont transformées en informations, elles ne contiennent jamais de détails inutiles.

Généralité sur l'informatique

□ Système de traitement de l'information



- Un système de traitement de l'information est composé de quatre unités:
- l'unité d'entrée (anglais: input), qui permet de faire entrer les informations dans le système. (l'entrée des informations utilise notamment des procédés de numérisation)
 - l'unité de stockage (anglais: storage) qui permet de conserver les informations.
 - l'unité de traitement (anglais: processor) qui comme son nom l'indique va traiter les informations.
 - l'unité de sortie (anglais: output) qui permet de faire sortir les résultats des traitements.

Généralité sur l'informatique

□ Représentation et stockage des données

- Elles sont de toutes sortes (texte, image, son, video), mais sont numérisées sous forme de 0 et de 1 (codées en binaire), on parle de **codage de l'information**.
 - **Le codage d'une information** consiste à établir une **correspondance** entre la représentation externe (habituelle) de l'information (le caractère A ou le nombre 36 par exemple), et sa représentation interne dans la machine, **qui est une suite de bits**.
 - On utilise la représentation **binaire** car:
 1. les données binaires sont plus faciles à mémoriser sur des supports physiques (bandes magnétiques, disques, usb, etc.)
 2. La représentation binaire est facile à réaliser techniquement à l'aide de bistables (système à deux états réalisés à l'aide de transistors).
- Par exemple, si l'on veut stocker un nombre entier sur le disque dur d'un ordinateur, on code généralement ce nombre en base 2 (langage machine) $(10)_2$ au lieu de le coder en base 10 (système décimal).

Exemple: le nombre 12 (en base 10) sera codé en base 2 par la suite binaire **00001100**,

Généralité sur l'informatique

□ Représentation et stockage des données

- On mesure la quantité de mémoire stockée dans les ordinateurs en :
 - Octet : 8 bits → permet de différencier **256** informations (ex : caractères)
 - Kilo Octet (Ko) : **2¹⁰** octets = 1024 octets
 - Mega Octet (Mo) : **2²⁰** octets = 1024 Kilo octets = 1 million d'octets
 - Giga Octet (Go) : **2³⁰** octets = 1 073 741 824 octets = **1024** Mo = 1 milliard d'octets
 - Tera Octet, Peta Octet, Exa Octet etc....

Généralité sur l'informatique

❑ Architecture de base d'un ordinateur

Définition: *Un ordinateur est une machine de traitement de l'information. Il est capable **d'acquérir** de l'information, de la **stocker**, de la **transformer** en effectuant des traitements quelconques, puis de la **restituer** sous une autre forme.*

❑ Principe de fonctionnement

Les deux principaux constituants d'un ordinateur sont **la mémoire principale** et le **processeur**. La mémoire principale (MP) permet de stocker de l'information, (**programmes** + données) tandis que le processeur exécute pas à pas les **instructions** composant les programmes.

Généralité sur l'informatique

□ Notion de programme

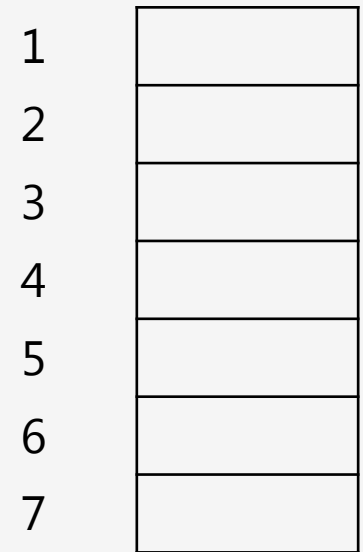
Un programme est une suite **d'instructions** élémentaires, qui vont être exécutées dans l'ordre par le **processeur**, Ces instructions correspondent à des **actions** très simples, comme **additionner** deux nombre, **lire** ou **écrire** une case mémoire, etc.

N.B: Chaque instruction est codifiée en mémoire sur quelques octets.

Généralité sur l'informatique

❑ Notion de mémoire principale

- La mémoire est divisée en emplacements de taille fixe (par exemple 8 bits), utilisés pour stocker les instructions et les données (Au moins temporairement).
- La mémoire de la machine est formée d'un grand nombre de cellules numérotées.
- Le numéro d'une case mémoire est appelé « adresse mémoire ».



Cases mémoire

Généralité sur l'informatique

❑ Notion de processeur

Est le cerveau de l'ordinateur. Il permet de manipuler des informations numériques, c'est-à-dire des informations codées sous forme binaire {0, 1}, et d'exécuter les instructions stockées en mémoire.

Pour chaque instruction, le processeur effectue schématiquement les opérations suivantes :

1. Lire en mémoire (MP) l'instruction à exécuter.
2. Effectuer le traitement correspondant.
3. Passer à l'instruction suivante.

❑ Notion de périphériques

Le programme reçoit des données des périphériques en entrée, et communique ses résultats en sortie à des périphériques. Une liste (non exhaustive) de périphériques usuels est :

- le clavier qui permet à l'utilisateur de saisir du texte ;
- la souris qui permet à l'utilisateur de sélectionner, d'activer ou de créer à la main des objets graphiques ;
- l'écran qui permet aux programmes d'afficher des données sous forme graphique ;
- l'imprimante qui permet de sortir des données sur support papier ;

Généralité

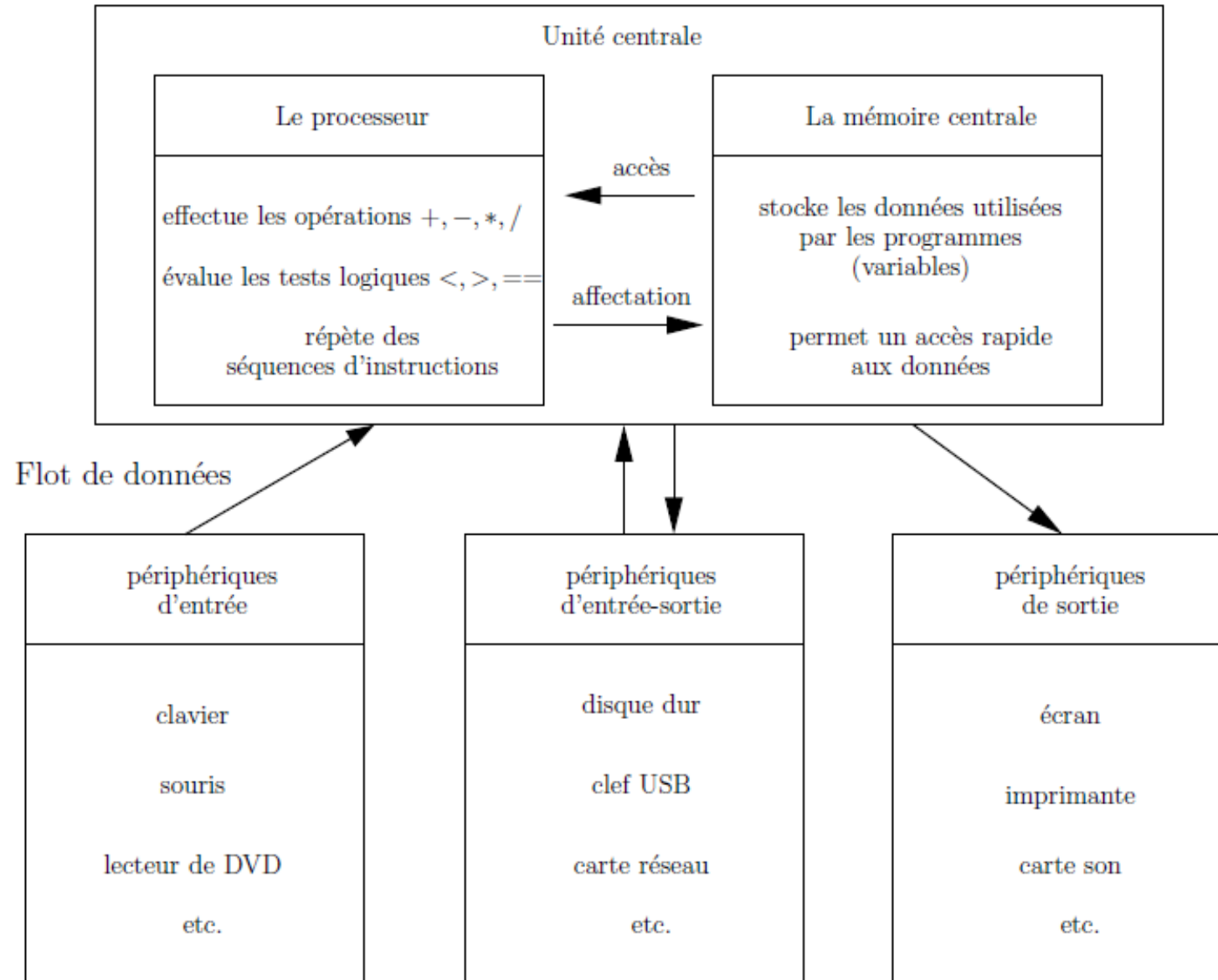


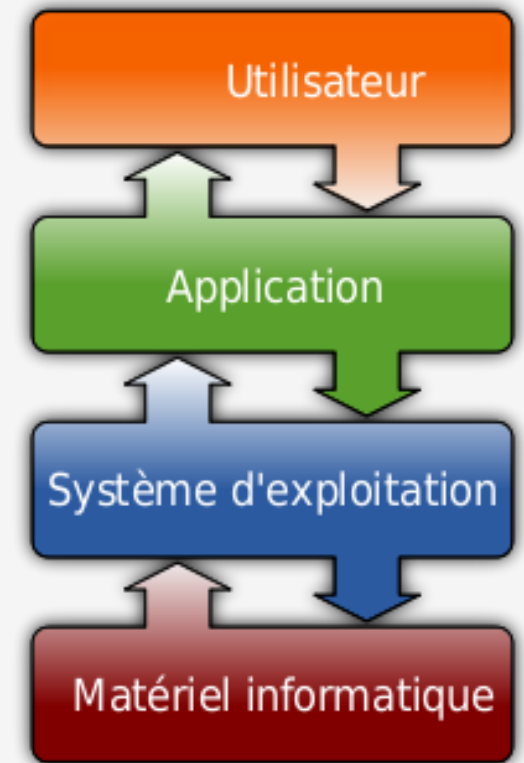
Figure 1.1- Schéma d'architecture d'un ordinateur.

Généralité sur l'informatique

❑ Notion de système d'exploitation

→ Un programme informatique doit recevoir des données pour les traiter, et produire d'autres données. Pour que le programme puisse fonctionner, il faut du matériel (composants électroniques), et il faut une couche logicielle **intermédiaire** avec le matériel, appelée système d'exploitation. Le système d'exploitation assure la communication entre le programme informatique et le matériel, et permet au programme **d'agir** sur le matériel.

→ Autre définition: Un système d'exploitation (OS: Operating System) est un ensemble de programmes qui **dirige** l'utilisation des ressources d'un ordinateur par des logiciels applicatifs.



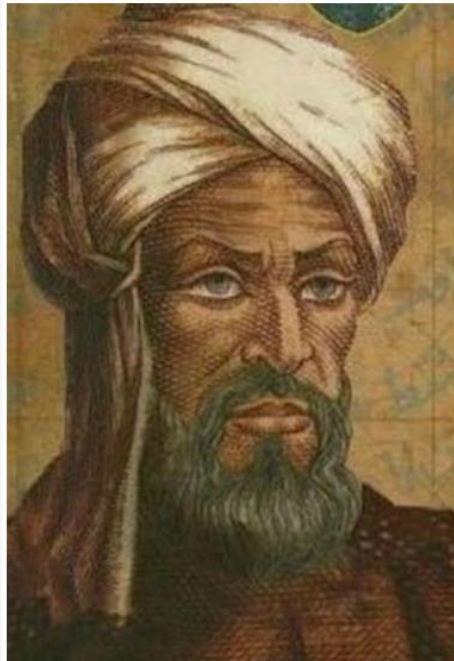
Partie 2: Algorithmique

Chapitre 1: Les éléments de base d'un algorithme

Algorithme – Notion et définition d'un algorithme

D'où vient le mot **algorithme**?

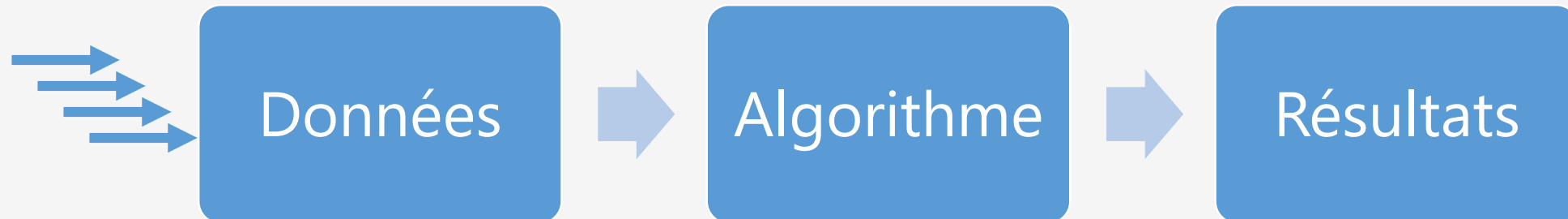
Muhammad Ibn Mûsâ al-Khuwârizmî (~ 780-850)
mathématicien, géographe, astrologue et astronome



Algorithme – Notion et définition d'un algorithme

Le terme **algorithme** vient du nom du mathématicien Al-Khwârizmî. Il est employé en informatique pour décrire une **méthode de résolution de problème** programmable sur machine.

Un algorithme est la description de la méthode de **résolution** d'un **problème** donné en utilisant des **instructions** élémentaires permettant d'aboutir à un **résultat**. Ces instructions deviennent compréhensibles par l'ordinateur lors de **la traduction de l'algorithme en programme**.



Algorithme – Notion et définition d'un algorithme

Exemple 1: (Algorithme de plantation d'un arbre)

Creuser un trou, reboucher un trou et placer un arbre dans un trou sont des opérations élémentaires (des actions simples) que toute personne (machine dans le cas de l'informatique) est censée savoir exécuter. Néanmoins, si un jardinier (programmeur) veut faire planter un arbre par une personne **qui ne sait pas le faire**, il doit lui fournir "un descriptif" (un algorithme) qui lui indique les opérations à faire ainsi que leur ordre d'exécution (séquencement).

----- Algorithme de plantation d'un arbre -----

1. **Creuser un trou**
2. **Placer l'arbre dans le trou**
3. **Reboucher le trou**

Algorithme – Notion et définition d'un algorithme

Exemple 2 (Algorithme de plantation et d'arrosage de plusieurs arbres):

Pour planter et arroser un ensemble d'arbres, on peut procéder de la manière suivante: planter l'ensemble d'arbres et les arroser tous à la fin.

----- Algorithme de plantation et d'arrosage de plusieurs arbres -----

- 1. Creuser un trou.**
- 2. Placer un arbre dans le trou.**
- 3. Reboucher le trou.**
- 4. S'il existe encore des arbres exécuter les actions 1, 2, 3 et 4.
Sinon exécuter les actions suivantes.**
- 5. Arroser les arbres.**

Remarque:

Pour résoudre le problème précédent, on peut procéder autrement : planter et arroser arbre par arbre. On conclut alors qu'à un problème donné pourraient correspondre plusieurs algorithmes.

Algorithme – Algorithmique et programmation

Tout problème à programmer doit être résolu, d'abord sous forme d'algorithme, puis converti en programme dans le langage de votre choix.

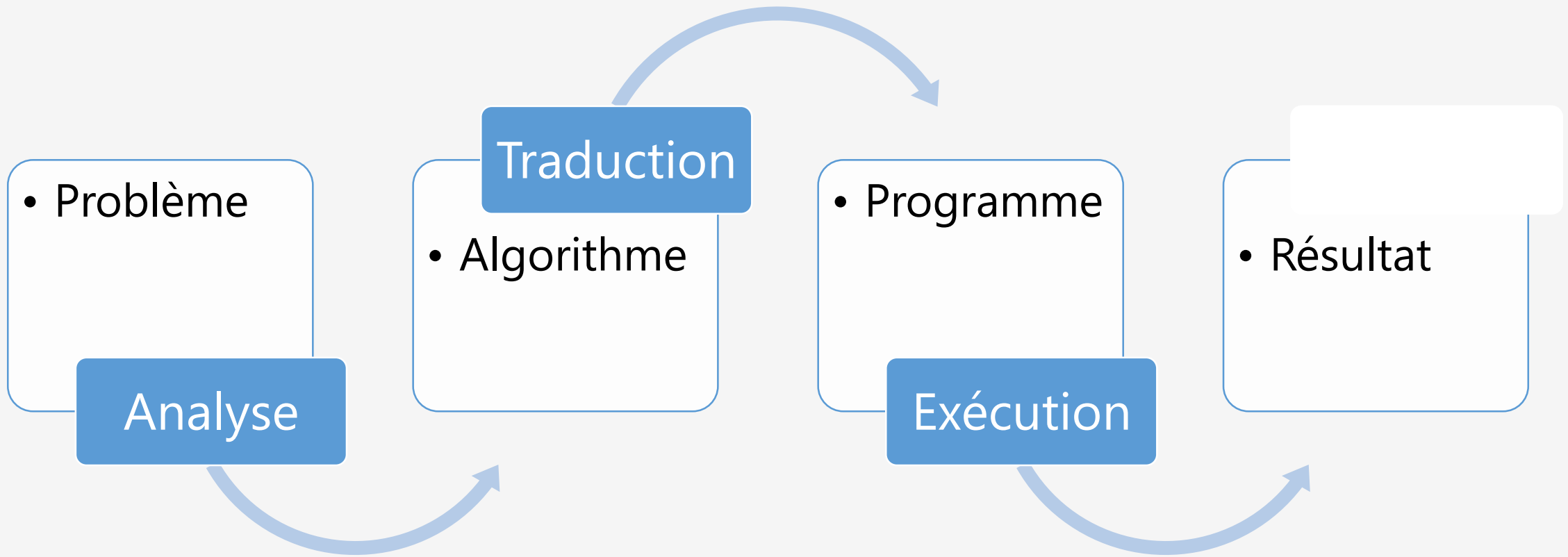
Un algorithme est indépendant du langage de programmation utilisé.

Programme:

Est un enchainement d'instructions, écrit dans **un langage de programmation**, exécutées par un ordinateur, permettant de traiter un problème et de renvoyer des résultats. Il présente **la traduction d'un algorithme** à l'aide d'un langage de programmation.

Algorithme – Algorithmique et programmation

❑ Cycle de développement d'un programme informatique:



Algorithme – Structure générale d'un algorithme

En général, un algorithme est composé de trois parties principales:

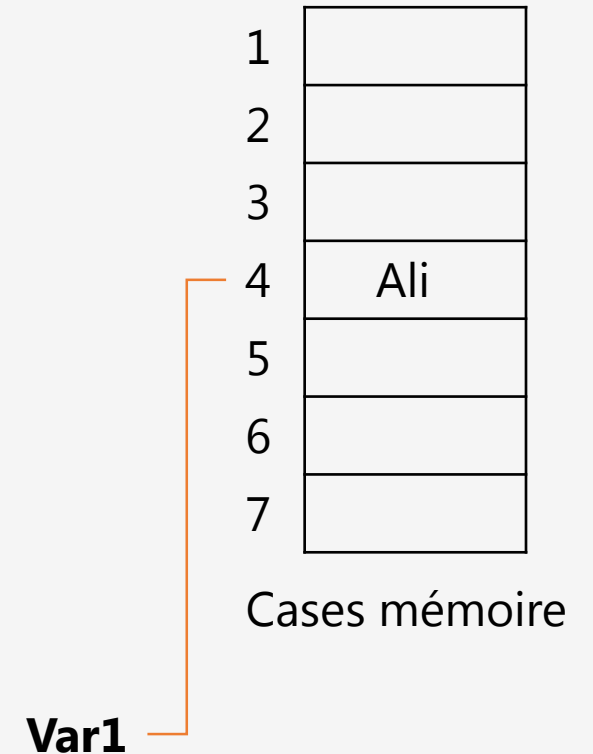
- ❑ L'en-tête: Cette partie sert à donner un nom à l'algorithme. Elle est précédée par le mot **Algorithme**.
- ❑ La partie déclarative: Dans cette partie, on déclare les différents objets que l'algorithme utilise (variables, constantes, ...)
- ❑ Le corps de l'algorithme: Cette partie contient les instructions (opérations) de l'algorithme. Elle est délimitée par les mots **Début** et **Fin**.

| | | |
|--|---|---|
| En-tête | { | Algorithme Nom de l'algorithme |
| Déclaration des variables et constantes | { | Variable Identificateur: type Constante Identificateur=valeur |
| Corps de l'algorithme (traitement) | { | Début <i>Instruction 1</i> <i>Instruction 2</i> <i>Instruction n</i> Fin |

Algorithme – Les variables et les constantes

□ Notion de variable

- L'algorithme ne manipule pas directement les valeurs des données.
→ C'est pour cela que les données ainsi que les résultats des calculs intermédiaires ou finaux, sont organisées dans des cases mémoires qui correspondent à des **variables**.
- Une variable est associée à une zone de la mémoire de la machine appelé « case mémoire » ou « mot mémoire ».
- Elle est repérée par un nom, ou **identificateur**.
- Elle contient une **valeur** qui peut varier au cours de l'exécution de l'algorithme, d'où le nom de variable.
- Au départ la valeur est indéfinie
- Toutes les valeurs qu'elle peut prendre sont d'un même type, c'est à dire sont de même nature et peuvent subir les mêmes opérations de base.
- Quand une variable est déclarée dans un algorithme, la machine lui associe une adresse mémoire.
- Dans l'algorithme, on ne manipule que le nom des variables, pas l'adresse mémoire.



Algorithme – Les variables et les constantes

□ Notion de variable

→ Une variable possède donc :

- ✓ Une valeur contenue par la case mémoire.
- ✓ Un identificateur: nom unique par lequel on peut accéder à son contenu.
- ✓ Un type qui définit la taille de la place occupée.

→ Ne pas confondre la variable et son contenu

- ✓ Une variable est un contenant(case ou boîte)
- ✓ Le contenu d'une variable est une valeur numérique, alphanumérique...

Algorithme – Les variables et les constantes

❑ Déclaration des variables (1)

La partie déclaration consiste à **énumérer toutes les variables** dont on aura besoin au cours de l'algorithme.

Chaque déclaration doit comporter le **nom de la variable** (identificateur) et **son type**.

La syntaxe: → **Variable** identificateur : type

Exemple →

Variable X : entier

Variable age : réel

Variable a,b,c : entiers

Variable prenom : chaîne

Variable resultat : logique

Algorithme – Les variables et les constantes

❑ Déclaration des variables (2)

- Identificateur

Un identificateur est le nom donné à une variable, une constante, fonction, etc. Ce nom doit obligatoirement **commencer** par une lettre suivie d'une suite de lettres et de chiffres et ne doit pas contenir d'espace.

Algorithme – Les variables et les constantes

❑ Déclaration des variables (2)

- Types de données

Un type détermine en particulier les valeurs possibles de la variable et les opérations primitives applicables à cette variable.

| Type de données | Numérique | | Alphanumérique | | Booléen |
|-----------------|--------------------------|--------------------------------|----------------------------|--|---|
| | Entier (Sans la virgule) | Réel (Avec et sans la virgule) | Caractères | Chaîne de caractères | |
| Exemple | -17 109 2020 | 34,17 -177,19 10 | "A" "#" "? " "\$" | "2" "+" "al-Khuwarizmi", "Bonjour", "112020" | Deux valeurs possibles: Soit Vrai, soit Faux |

Algorithme – Les variables et les constantes

❑ Déclaration des variables (2)

- Types de données

Préciser le type des données suivantes:

| Donnée | Type |
|-----------------------------|------|
| -700 | |
| "9" | |
| "Welcome to EST FBS" | |
| -40.99 | |
| " # " | |
| Faux | |
| "Vrai" | |

Algorithme – Les variables et les constantes

❑ Déclaration des variables (2)

- Types de données

Préciser le type des données suivantes:

| Donnée | Type |
|----------------------|----------------------|
| -700 | entier |
| "9" | caractère |
| "Welcome to EST FBS" | Chaine de caractères |
| -40.99 | réel |
| " # " | Chaine de caractères |
| Faux | booléen |
| "Vrai" | Chaine de caractère |

Algorithme – Les variables et les constantes

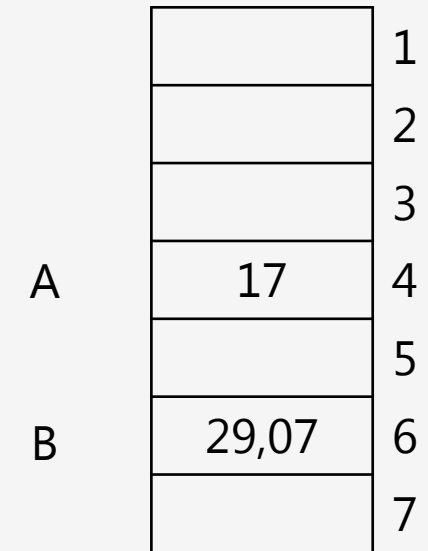
❑ Déclaration des variables (3)

Déclarer une variable :

- réserver une place en mémoire
- attribuer l'identificateur à cette place

Identificateur: → **A**
Type: → **Entier**
Valeur: → **17**

Identificateur: → **B**
Type: → **réel**
Valeur: → **29,07**



Cases mémoire

Algorithme – Les variables et les constantes

❑ Les Constantes

Une constante est un objet dont l'état reste inchangé durant toute l'exécution d'un programme. On ne peut jamais modifier sa valeur et celle-ci doit donc être précisée lors de la définition de l'objet.

Déclaration des constantes

Syntaxe: → Constante identificateur = valeur

Exemple → Constante **PI** = **3.14**

Algorithme – Les instructions de base

Une instruction est une action élémentaire commandant à la machine un calcul. Les instructions de base sont:

1. L'instruction d'affectation: L'affectation, notée par le symbole \leftarrow , est l'opération qui évalue une expression (constante ou une expression arithmétique ou logique) et attribue la valeur obtenue à une variable.

Syntaxe: \rightarrow Variable \leftarrow expression

$a \leftarrow 17$ (a reçoit la valeur 17)

$a \leftarrow (a*b)+c$ (a reçoit le résultat de $(a*b)+c$)

Exemple d'affectation

$a \leftarrow 'K'$ (a reçoit la lettre K)

$a \leftarrow X \text{ et } Y$ (a reçoit le résultat de l'expression logique)

Algorithme – Les instructions de base

2. L'instruction d'entrée ou de lecture

Cette instruction permet d'attribuer à une variable une valeur introduite au moyen d'un organe d'entrée (généralement le clavier).

Syntaxe: → Lire (identificateur)

Exemple

Lire (A)

Lire (A,B,C)

Remarque: Lorsque le programme rencontre cette instruction, l'exécution s'interrompt et attend que l'utilisateur tape une valeur. Cette valeur est rangée en mémoire dans la variable désignée.

Algorithme – Les instructions de base

3. L'instruction de sortie ou de l'écriture

Cette instruction permet de communiquer une valeur donnée ou un résultat d'une expression à l'organe de sortie. En d'autres termes elle permet d'afficher des informations à l'écran.

Syntaxe: → **Ecrire** (expression)

expression: une valeur, un résultat, un message, etc

Exemple 1

Ecrire (A) (affiche à l'écran la valeur de la variable A)

Exemple 2

A ← 17

Ecrire ("La valeur de A est = ", A) Que affiche la dernière ligne ?

Algorithme – Les opérateurs et expressions

→ Les opérations possibles sur les variables **dépendent de leur type**.

- Exemple: On ne peut pas multiplier des mots

1. Les opérateurs de comparaison: $>$, $>=$, $<$, $<=$, $=$, \neq (entiers, réels et caractères)

2. Les opérateurs arithmétiques:

Les opérateurs arithmétiques permettent de faire des calculs entre des valeurs numériques :

- Entre des réels: $+$, $-$, $*$ (la multiplication), $/$ (la division), $^$ (Exposant) + les opérateurs de comparaison.
- Entre deux entiers: $+$, $-$, $*$ (la multiplication), $/$ (la division), $^$ (Exposant), **DIV** (la division entière), **MOD** (le reste de la division entière) + les opérateurs de comparaison.

3. Les opérateurs de concaténations: Entre chaînes: Les opérateurs de comparaison, la concaténation (**&**)

4. Les opérateurs logiques: **ET** , **OU** , **NON**

Algorithme – Exemple d'algorithme

1. Nom d'algorithme

2. Déclaration de variables

3. Corps de l'algorithme

```
Algorithme nbr_poissons
Variable nbr_poiss : entier
           poids_t, poids_poiss : réel
Debut
poid_t <-- 0
nbr_poiss <-- 0
TantQue poid_t < 1000
Ecrire ('Donner le poids du poisson pesé')
Lire (poids_poiss)
poid_t <-- poid_t + poids_poiss
nbr_poiss <-- nbr_poiss + 1
FinTantQue
Ecrire ('Le nombre de poisson est:', nbr_poiss)
Fin
```

Algorithme – Exemple d'algorithme

Algorithme – Exercice

Exercice 1: Calcule PTTC

Ecrire un algorithme qui permet de saisir le prix HT (PHT) d'un produit et de calculer son prix total TTC (PTTC), TVA = 20%.

Algorithme – Exercice

Solution:

Ecrire un algorithme qui permet de saisir le prix HT (PHT) d'un produit et de calculer son prix total TTC (PTTC), TVA = 20%.

```
Algorithme Calcul_PTTC
Variables PHT, PTTC : réel
Constante TVA = 0.2
Début
    Ecrire ("Entrer le prix hors taxes:")
    Lire (PHT)
     $PTTC \leftarrow PHT + (PHT * TVA)$ 
    Ecrire ("Le prix TTC est =", PTTC)
Fin
```

Algorithme – Exercice

Exercice 2: **Ecrire un algorithme qui permet de calculer le périmètre et la surface d'un cercle.**

Algorithme – Exercice

Solution :

Ecrire un algorithme qui permet de calculer le périmètre et la surface d'un cercle

Algorithme Cercle

Variables Ray, Per, Surf : réel

Constante $P = 3.14$

Début

 Ecrire ("Entrer le rayon:")

 Lire (Ray)

$Per \leftarrow 2 * R * P$

$Surf \leftarrow R * R * P$

 Ecrire ("La surface est =", Surf)

 Ecrire ("Le périmètre est =", Per)

Fin

Algorithme – Exercice

Exercice 3: Écrire un programme qui permute la valeur de deux variables c1 et c2 de type caractère

Algorithme – Exercice

Solution

Algorithme Permutation_deux_caractères

Variables c1, c2 : caractère

Variable inter : caractère

Début

Ecrire ("Entrer le premier caractère:")

Lire (c1)

Ecrire ("Entrer le deuxième caractère:")

Lire (c2)

inter \leftarrow c1

c1 \leftarrow c2

c2 \leftarrow inter

Ecrire ("Les valeurs de c1 et c2 après la permutation sont:", c1,c2)

Fin

Algorithme – Exercice

Exercice 4: **Écrire un programme qui permet de calculer la note moyenne du semestre.**

Remarques :

- Un semestre comprend 4 modules.
- La note d'un module /20

Partie 2: Algorithmique

Chapitre 2: Les structures conditionnelles et répétitives

A- Les structures conditionnelles (Alternatives)

A- Les structures conditionnelles (Alternatives)

Introduction:

Traitement séquentiel # Structure Conditionnelle

Un traitement séquentiel contient une procédure étape par étape à exécuter, alors que la structure conditionnelle ou alternative permet d'exécuter ou non une série d'instructions selon la valeur d'une condition.

A- Les structures conditionnelles (Alternatives)

1. La structure Si ... Alors ... FinSi

Syntaxe:

Si condition **Alors**

 traitement -- *une séquence d'instructions*

FinSi

Règle : La condition est nécessairement une ***expression*** ou une ***variable booléenne***.

Évaluation (3 étapes) :

1. la condition est évaluée ;
2. si la condition est **vraie**, le **traitement** est exécuté puis le contrôle passe à l'instruction qui suit le **FinSi** ;
3. si la condition est **fausse**, la machine saute directement à l'instruction qui suit le **FinSi**.
→ En d'autres termes, le traitement est effectué si et seulement si la condition est **VRAI**.

A- Les structures conditionnelles (Alternatives)

Exercice 1:

Écrire un algorithme qui lit une valeur entière au clavier et affiche « pair » si elle est pair.

A- Les structures conditionnelles (Alternatives)

Solution

```
Algorithme Nombre_pair
Variables Nbr, R : entier

Début
  Ecrire ("Entrer un nombre:")
  Lire (Nbr)
  R ← Nbr MOD 2
  Si R=0 Alors
    Ecrire (Nbr," est pair")
  FinSi
Fin
```

A- Les structures conditionnelles (Alternatives)

2. La structure Si ... Alors ... Sinon ... FinSi

Syntaxe:
Si condition **Alors**
 Instruction(s) 1
Sinon
 Instruction(s) 2
FinSi

Règle : La condition est nécessairement une expression ou une variable booléenne.

Évaluation :

1. la condition est évaluée ;
2. si la condition est **vraie**, la série **d'instructions 1** est exécutée et l'ensemble d'instructions 2 est ignoré. puis le contrôle passe à l'instruction qui suit le **FinSi** ;
3. De même, si la condition est **fausse**, la machine saute directement à la première ligne située après le **Sinon** et exécute l'ensemble **d'instruction2**.

A- Les structures conditionnelles (Alternatives)

Exercice 2:

Écrire un algorithme qui lit un nombre entier au clavier et affiche s'il s'agit d'un nombre positif ou négatif.

A- Les structures conditionnelles (Alternatives)

3. La structure Si ... Alors ... SinonSi ... Sinon ... FinSi

Syntaxe:

Si **condition1** **Alors**

Instruction(s) 1

SinonSi **condition2** **Alors**

Instruction(s) 2

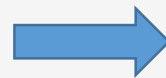
SinonSi **condition3** **Alors**

Instruction(s) 3

Sinon

Instructions

FinSi



L'instruction qui sera exécutée est l'instruction dont la condition est vraie.

Si aucune condition n'a la valeur vraie, c'est l'instruction qui suit la **Sinon** qui sera exécutée.

A- Les structures conditionnelles (Alternatives)

Exercice 3:

Écrire un algorithme qui demande le numéro d'un mois et affiche le nombre de jours que comporte ce mois en utilisant la structure Si ... Alors ... SinonSi ... Sinon ... FinSi.

Janvier, Mars, Mai, Juillet, Août, Octobre, Décembre → **31 jours**

Avril, Juin, Septembre, Novembre → **30 jours**

Février → **28 jours**

A- Les structures conditionnelles (Alternatives)

Algorithme Nombre_de_jours_par_Mois

Variable mois: entier

Début

Ecrire("Entrer une valeur entre 1 et 12")

Lire(mois)

Si(mois=1 ou mois=3 ou mois=5 ou mois=7 ou mois=8 ou mois=10 ou mois=12) **Alors**

Ecrire(« Ce mois comporte 31 jours")

SinonSi (mois=4 ou mois= ou mois=6 ou mois=9) **Alors**

Ecrire("Ce mois comporte 30 jours")

SinonSi (mois=2) **Alors**

Ecrire("Ce mois comporte 28 jours")

Sinon

Ecrire (« Nombre invalide, ce mois n'existe pas")

Fin

A- Les structures conditionnelles (Alternatives)

Exercice 4:

**Ecrire un algorithme qui permet d'afficher le maximum parmi deux nombres saisis au clavier.
(Utiliser l'instruction **Si ... Alors ... SinonSi ... Sinon ... FinSi**).**

A- Les structures conditionnelles (Alternatives)

Algorithme Maximum_de_deux_nombres

Variable A,B: réel

Début

Ecrire("Entrer la valeur de A")

Lire(A)

Ecrire("Entrer la valeur de B")

Lire(B)

Si(A > B) **Alors**

Ecrire("Le maximum est", A)

SinonSi (A = B) **Alors**

Ecrire("Egalité")

Sinon

Ecrire ("Le maximum est", B)

FinSi

Fin

A- Les structures conditionnelles (Alternatives)

Remarque:

Une structure conditionnelle peut contenir à son tour une autre structure conditionnelle, c'est-à-dire l'une incluse dans l'autre

Exemple :

Ecrire un algorithme permettant de calculer le montant des allocations familiales sachant que le montant dépend du nombre d'enfants:

- Si nombre d'enfants est inférieur ou égal à 3 alors les allocations sont de 150 dh par enfant.
- Si le nombre d'enfants est strictement supérieur à trois et inférieur ou égal à 6 alors les allocations sont de:
 - 150 dh par enfant pour les trois premiers enfants
 - 38 dh par enfant pour les suivants
- Si le nombre d'enfants est strictement supérieur à 6 alors les allocations sont:
 - 150 dh par enfant pour les trois premiers enfants
 - 38 dh par enfant pour les suivants
 - 0 dh par enfant pour les suivants

A- Les structures conditionnelles (Alternatives)

Algorithme Allocations_familiales

Variable NbrEnf, Montant: entier

/ NbrEnf: Nombre d'enfants, Montant: Montant des allocations familiales */*

Début

Ecrire("Entrer le nombre d'enfants")

Lire(NbrEnf)

Si (NbrEnf ≤ 3) **Alors**

Montant ← NbrEnf * 150

Sinon

Si (NbrEnf ≤ 6) **Alors**

Montant ← (3*150) + (NbrEnf - 3) * 38

Sinon

Montant ← 564

FinSi

FinSi

Ecrire("Le montant d'allocations familiales est", Montant)

Fin

A- Les structures conditionnelles (Alternatives)

4. La structure à choix multiples

Cette instruction conditionnelle permet de **choisir** le traitement à effectuer **en fonction de la valeur** ou de **l'intervalle de valeurs** d'une variable ou d'une expression.

Syntaxe:

Selon expression **faire**
choix 1: action(s)1
choix 2: action(s)2
choix 3: action(s)3
...
choix n: action(s)n

Sinon
actions
FinSelon

A- Les structures conditionnelles (Alternatives)

4. La structure à choix multiples

- Evaluation:

- ✓ L'expression est évaluée, puis sa valeur est successivement comparée à chacun des ensembles choix i .
- ✓ Dès qu'il y a correspondance, les comparaisons sont arrêtées et la séquence associée est exécutée.
- ✓ À la fin de l'exécution, on continuera la suite de l'exécution du programme à partir de la 1^{ère} instruction qui vient après **FinSelon**.
- ✓ Si aucun choix ne correspondant, on exécutera le traitement **Sinon** du **Selon**.

Syntaxe:

Selon expression **faire**
choix 1: action(s)1
choix 2: action(s)2
choix 3: action(s)3
...
choix n: action(s)n

Sinon
actions
FinSelon

A- Les structures conditionnelles (Alternatives)

4. La structure à choix multiples

- Remarques:

1. Le sélecteur doit être de type **scalaire discret** et doit avoir une valeur avant d'être impliqué dans le **Selon**.
2. Les traitements relatifs aux valeurs peuvent comporter également d'autres structures conditionnelles.
3. Choix i peut être un ensemble des valeurs séparées par des virgules ou intervalle des valeurs ($V_i .. V_f$).
4. Si la valeur du sélecteur est différente des valeurs proposées alors c'est le traitement qui suit la clause **sinon** qui sera exécuté.

Syntaxe:

```
Selon expression faire  
    choix 1: action(s)1  
    choix 2: action(s)2  
    choix 3: action(s)3  
    ...  
    choix n: action(s)n  
Sinon  
    actions  
FinSelon
```


A- Les structures conditionnelles (Alternatives)

4. La structure à choix multiples

Exemple:

Ecrire un algorithme permettant d'afficher le jour en toute lettre selon son numéro saisi au clavier.

Algorithme Jours

Variable J: entier

Début

Ecrire("Entrer le numéro du jours")

Lire(J)

Selon (J) Faire

1: Ecrire("Dimanche")

2: Ecrire("Lundi")

3: Ecrire("Mardi")

...

7: Ecrire("Samedi")

Sinon

Ecrire ("Le numéro saisi ne correspond pas à un jours")

FinSelon

Fin

A- Les structures conditionnelles (Alternatives)

Exercice 5:

Ecrire un algorithme qui vérifie si le caractère saisi par l'utilisateur est une voyelle ou une consonne en utilisant **Selon**.

Les voyelles sont: a, e, i, o, u, y

A- Les structures conditionnelles (Alternatives)

Solution:

Partie 2: Algorithmique

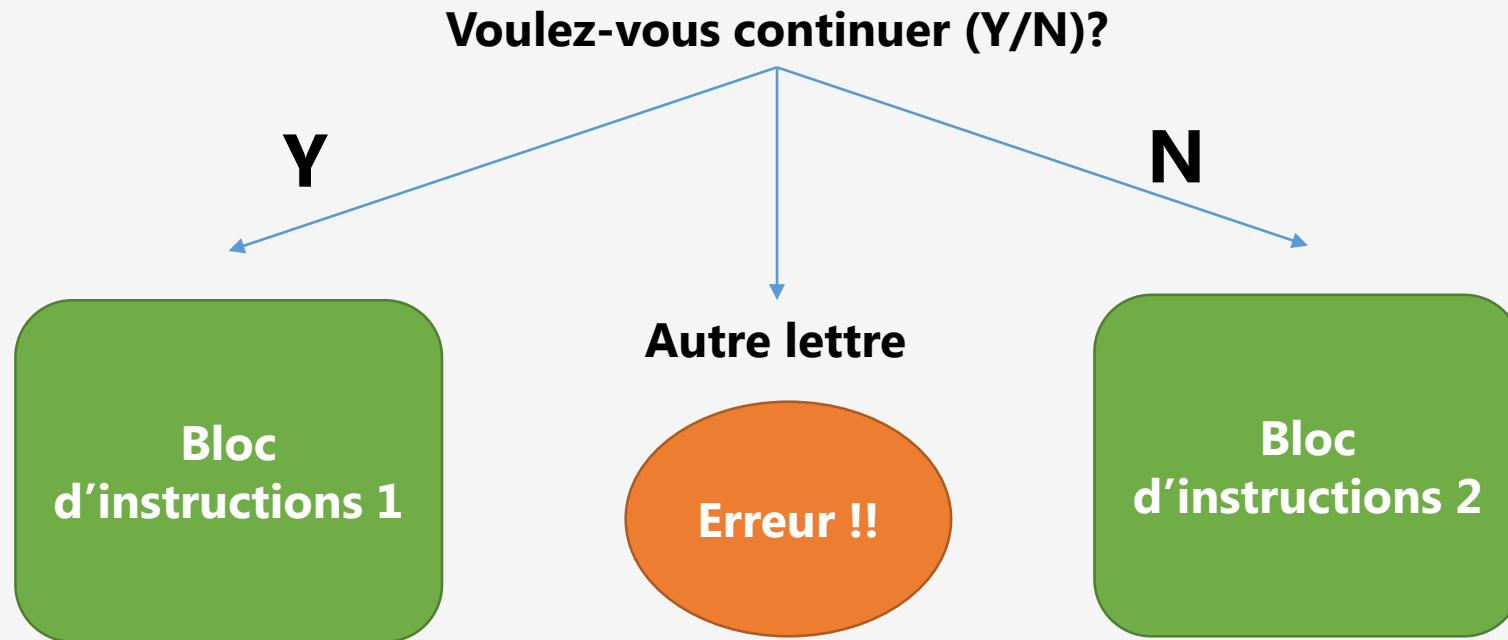
Chapitre 2: Les structures conditionnelles et répétitives

B- Les structures répétitives (Les boucles)

B- Les structures répétitives (itératives)

Introduction

→ Problématique

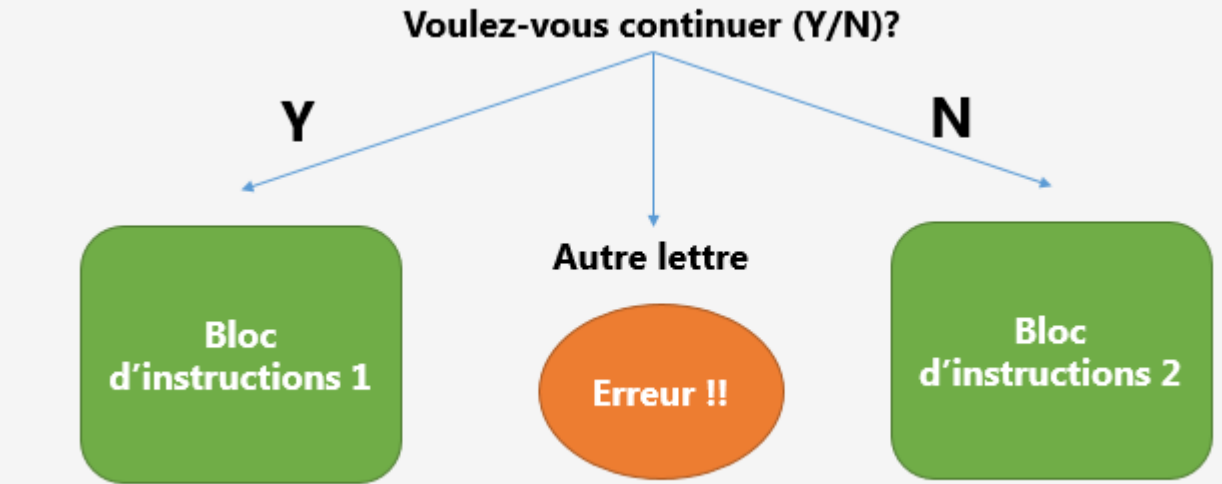


B- Les structures répétitives (itératives)

Introduction

→ Solution 1

```
Algorithme contrôle_saisie
Variabe Rep : caractère
Début
Ecrire ("Voulez-vous continuer Y/N? ")
Lire(Rep)
Si Rep # 'Y' ET Rep # 'N' Alors
Ecrire ("Erreur de saisie, Recommencez")
→    ??
Finsi
Fin
```



Cet algorithme va résoudre le problème si on trompe qu'une seule fois, et on va saisir une valeur correcte à la deuxième demande.

En cas de deuxième erreur, il faudrait rajouter une autre **SI**. Et ainsi de suite, on peut rajouter des centaines de **SI**.

- ✓ **La solution optimale consiste à utiliser une structure répétitive**

B- Les structures répétitives (itératives)

Une structure répétitive (boucle) est utilisée quand une instruction ou un bloc d'instructions, doit être répété plusieurs fois.

NB. La répétition est soumise à une condition.

B- Les structures répétitives (itératives)

1. La boucle TantQue ... Faire

La boucle TantQue permet de répéter un traitement tant que **la condition est vraie**.

Syntaxe:

```
TantQue condition Faire  
    instruction(s)  
FinTantQue
```

L'exécution de la boucle dépend de la valeur de la condition. Si elle est vraie, le programme va exécuter les instructions qui suivent, jusqu'à ce qu'il rencontre la ligne **FinTantQue**. Il retourne ensuite sur la ligne du TantQue pour **tester de nouveau** la condition, le même processus sera ensuite répété.

→ La boucle ne s'arrête que lorsque la condition prend la valeur fausse, et dans ce cas le programme poursuit son exécution après **FinTantQue**.

B- Les structures répétitives (itératives)

1. La boucle TantQue ... Faire

→ Solution optimale avec la boucle TantQue ... Faire

```
Algorithme contrôle_saisie
Variable Rep : caractère
Début
Ecrire ("Voulez-vous continuer Y/N? ")
Lire(Rep)
TantQue Rep# 'Y' ET Rep# 'N' FAIRE
Ecrire ("Erreur de saisie, Merci de répondre par Y Or N")
Lire(Rep)
FinTantQue
Ecrire ("Merci de votre participation")
Fin
```

B- Les structures répétitives (itératives)

1. La boucle TantQue ... Faire

Remarques importantes:

- Etant donnée que la condition est évaluée avant la mise en œuvre des instructions, il est possible que celles-ci **ne soient jamais exécutées**.
- Si une structure **TantQue** dans la quelle la condition ne devient jamais fausse. Le programme tourne dans une boucle infinie et n'en sort plus.

Exemple de boucle infinie

```
I ← 1  
TantQue I <= 3 Faire  
    Ecrire ("Bonjour")  
FinTantQue
```

B- Les structures répétitives (itératives)

1. La boucle TantQue ... Faire

Exercice :

En utilisant la boucle **TantQue**, écrire un algorithme qui permet de lire un entier (N) et qui calcule la somme des nombres compris entre 0 et ce dernier (N).

$$S=0+1+2+3+4+...+N$$

B- Les structures répétitives (itératives)

1. La boucle TantQue ... Faire

Solution :

```
Algorithme somme_nombres_compri_Zero_Nbr
Variabe Nbr, S, I : entier
Début
Ecrire ("Veuillez entrer un nombre ")
Lire(Nbr)
I←0
S←0
TantQue I<=Nbr Faire
    S←S+I
    I←I+1
FinTantQue
Ecrire (" La somme des nombres compris entre 0 et", Nbr, "est: ", S)
Fin
```

B- Les structures répétitives (itératives)

2. La boucle Répéter ... Jusqu'à

La boucle Répéter permet de répéter un bloc d'instructions **jusqu'à la vérification de la condition** (c.à.d jusqu'à ce qu'une condition soit vraie).

Syntaxe:

Répéter

instruction(s)

.....

Jusqu'à condition

- La liste d'instructions est exécutée, puis la condition est évaluée. Si elle est fausse, le corps de la boucle est exécuté à nouveau puis la condition est réévaluée et si la valeur est vraie, le programme sort de la boucle et exécute l'instruction qui suit **Jusqu'à**.

B- Les structures répétitives (itératives)

2. La boucle répéter ... Jusqu'à

Exemple: L'algorithme qui permet l'affichage d'un message plusieurs fois peut être écrit comme suit:

```
Algorithme RépétitionMessage  
Variabe N, I : entier  
Début  
Ecrire ("Entrer la valeur de N")  
Lire(N)  
 $I \leftarrow 1$   
Répéter  
Ecrire ("Welcome to EST FBS")  
 $I \leftarrow I + 1$   
Jusqu'à  $I > N$   
Fin
```

Question: Si $N = 4$ et la condition liée à l'instruction *Répéter* est ($I < N$), combien de fois le message sera affiché??

B- Les structures répétitives (itératives)

2. La boucle Répéter ... Jusqu'à

Exercice :

En utilisant la boucle **Répéter**, écrire un algorithme qui permet de lire deux nombres entiers Nbr1 et Nbr2 et de faire l'opération $\text{Nbr1}/\text{Nbr2}$, tout en vérifiant que le diviseur est différent à zéro.

B- Les structures répétitives (itératives)

2. La boucle répéter ... Jusqu'à

Solution :

```
Algorithme Division
Variable Nbr1, Nbr2: entier
Début
Ecrire ("Entrer le 1er nombre")
Lire(Nbr1)
Répéter
Ecrire (" Veuillez entrer un nombre différent à zéro")
Lire(Nbr2)
Jusqu'à Nbr # 0
Ecrire (" Le résultat de la division est:", Nbr1/Nbr2)
Fin
```


B- Les structures répétitives (itératives)

3. La boucle Pour ... Faire

La boucle Pour permet de répéter un bloc d'instructions **un nombre de fois connu à priori**.

Syntaxe:

```
Pour      compteur ← val_initiale  
jusqu'à val_finale Faire  
           instruction(s)  
FinPour
```

- La variable **compteur** est de type entier. Elle est initialisée à la valeur initiale. Le compteur augmente sa valeur de 1 **automatiquement** à chaque tour de boucle jusqu'à la valeur finale.
- Pour chaque valeur prise par la variable compteur, le bloc d'instructions est exécuté.
→ **La boucle ne s'arrête que lorsque la variable compteur prend sa valeur finale, et dans ce cas le programme poursuit son exécution après FinPour.**

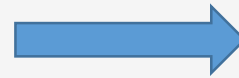
B- Les structures répétitives (itératives)

3. La boucle Pour ... Faire

Exemple: Que donne l'algorithme suivant?

```
Algorithme xxx_5
  Variable I : entier
Début

Pour I←0 jusqu'à 10 Faire
  Ecrire ("5*",I, "=",5*I)
FinPour
Fin
```



```
Résultat
---Tableau de multiplication de 5---
5*0=0
5*1=5
5*2=10
5*3=15
5*4=20
5*5=25
5*6=30
5*7=35
5*8=40
5*9=45
5*10=50
```

B- Les structures répétitives (itératives)

3. La boucle **Pour ... Faire**

Exercice :

En utilisant la boucle **Pour**, écrire un algorithme qui permet de calculer la somme des 20 premiers entiers positifs > 0 .

B- Les structures répétitives (itératives)

3. La boucle Pour ... Faire

Solution :

```
Algorithme somme_20Premier_entier
Variable Somme, I : entier
Début
    Somme ← 0
Pour I ← 1 jusqu'à 20 Faire
    Somme ← Somme + I
FinPour
Ecrire (" La somme est S=", Somme)
Fin
```

B- Les structures répétitives (itératives)

3. La boucle Pour ... Faire

Remarques

- ✓ Par défaut la variable compteur est incrémentée de 1 à chaque tour de boucle.
- ✓ Pour modifier la valeur d'incrémentation, il suffit de rajouter le mot **Pas** et la valeur de ce pas à la boucle Pour.

Syntaxe:

Pour **compteur** ← val_initiale **jusqu'à** val_finale **Pas** Val_Pas **Faire**
instruction(s)
FinPour

B- Les structures répétitives (itératives)

3. La boucle Pour ... Faire

Exercice :

En utilisant la boucle **Pour**, écrire un algorithme qui permet de saisir un nombre entier (Nbr) et qui calcule la somme des entiers pairs jusqu'à ce nombre. (Par exemple, si Nbr=10, l'algorithme doit calculer $0+2+4+6+8+10=30$)

B- Les structures répétitives (itératives)

3. La boucle Pour ... Faire

Solution :

```
Algorithme somme_20Premier_entier
Variabe S, I, Nbr : entier
Début
Ecrire ("Veuillez entrer un nombre entier ")
Lire(Nbr)
    S ← 0
Pour I ← 0 jusqu'à Nbr Pas 2 Faire
    S ← S + I
FinPour
Ecrire (" La somme des nombres pairs est S=", S)
Fin
```

B- Les structures répétitives (itératives)

❑ Remarques importantes

- ✓ Les instructions de la boucle Répéter sont exécutées **au moins une fois**, car l'évaluation de la condition vient après le passage dans la boucle. Cependant, les instructions de la boucle **TantQue peuvent ne jamais être exécutées** si la condition n'est pas vérifiée, lors de la première évaluation.
- ✓ La boucle Répéter s'exécute jusqu'à ce que **la condition soit vérifiée**. Donc la condition permet la sortie de la boucle **Répéter**. La boucle Tant que s'exécute tant que la condition est vérifiée. Donc la condition permet le passage dans la boucle **TantQue**.
- ✓ On utilise la boucle **Pour** quand l'on connaît le nombre d'itération à l'avance.


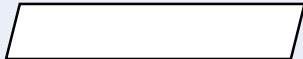
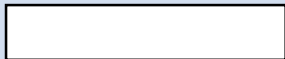


Partie 2: Algorithmique

C- L'organigramme

C- Organigramme

❑ Définition:

- Un organigramme est la schématisation d'un algorithme sous forme géométrique (rectangle, cercle, losange, Parallélogramme ...) et des flèches qui indiquent **l'enchaînement** des instructions à exécuter.
- Chaque instruction est représentée par un symbole normalisé:

| Symbole | Instruction |
|---|---|
|  | Pour le début et la fin de l'algorithme |
|  | Pour la lecture et l'écriture d'une donnée (Entrée/Sortie) |
|  | Pour l'affectation |
|  | Pour les conditions (Test) |
|  | Liaison , avec ou sans flèche, permettent de guider la lecture de l'organigramme |

C- Organigramme

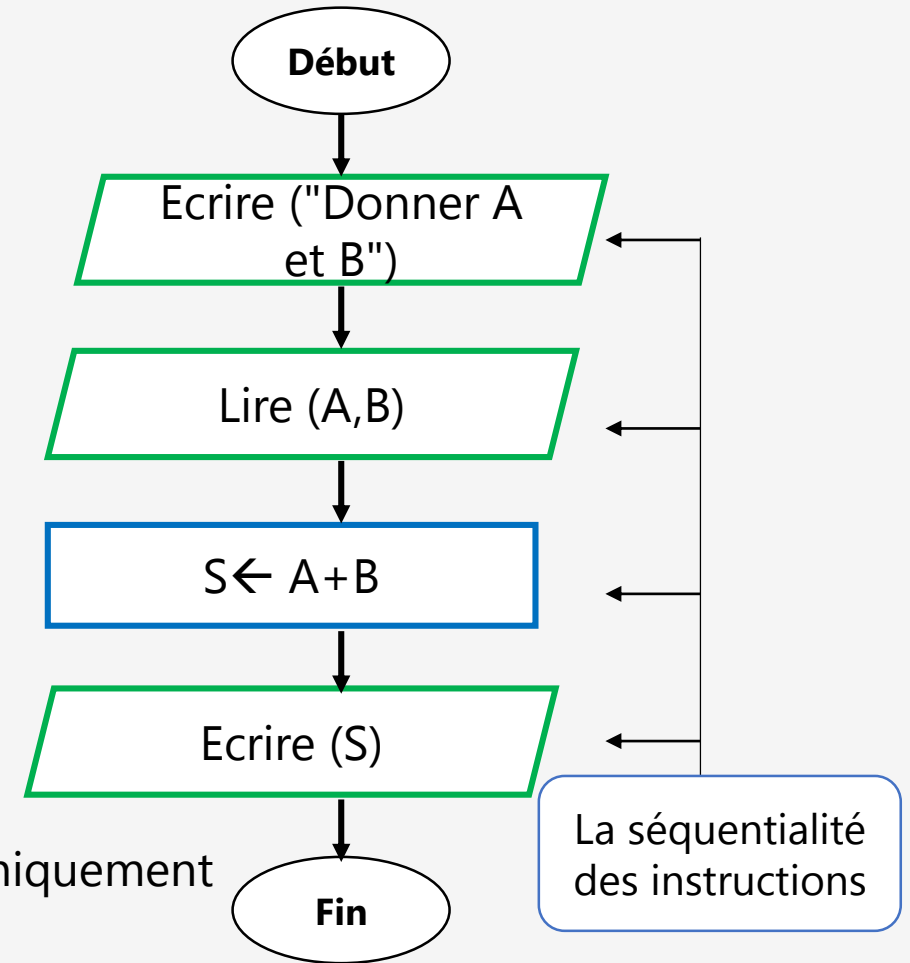
❑ Exemple 1 (traitement séquentiel):

Soit l'algorithme suivant

```
Algorithme somme_deux_nombres  
Variable A, B, S : entier  
Début  
  Ecrire (" Donner A et B ")  
  Lire(A,B)  
   $S \leftarrow A+B$   
  Ecrire (S)  
Fin
```

La partie concernée par
l'organigramme

N.B: Seule la partie **corps de l'algorithme** qu'on doit présenter graphiquement sous forme d'organigramme



C- Organigramme

❑ Exemple 2 (structure conditionnelle simple):

Soit l'algorithme suivant

Algorithme signe

Variabe A : entier

Début

Ecrire (" Donner A ")

Lire(A)

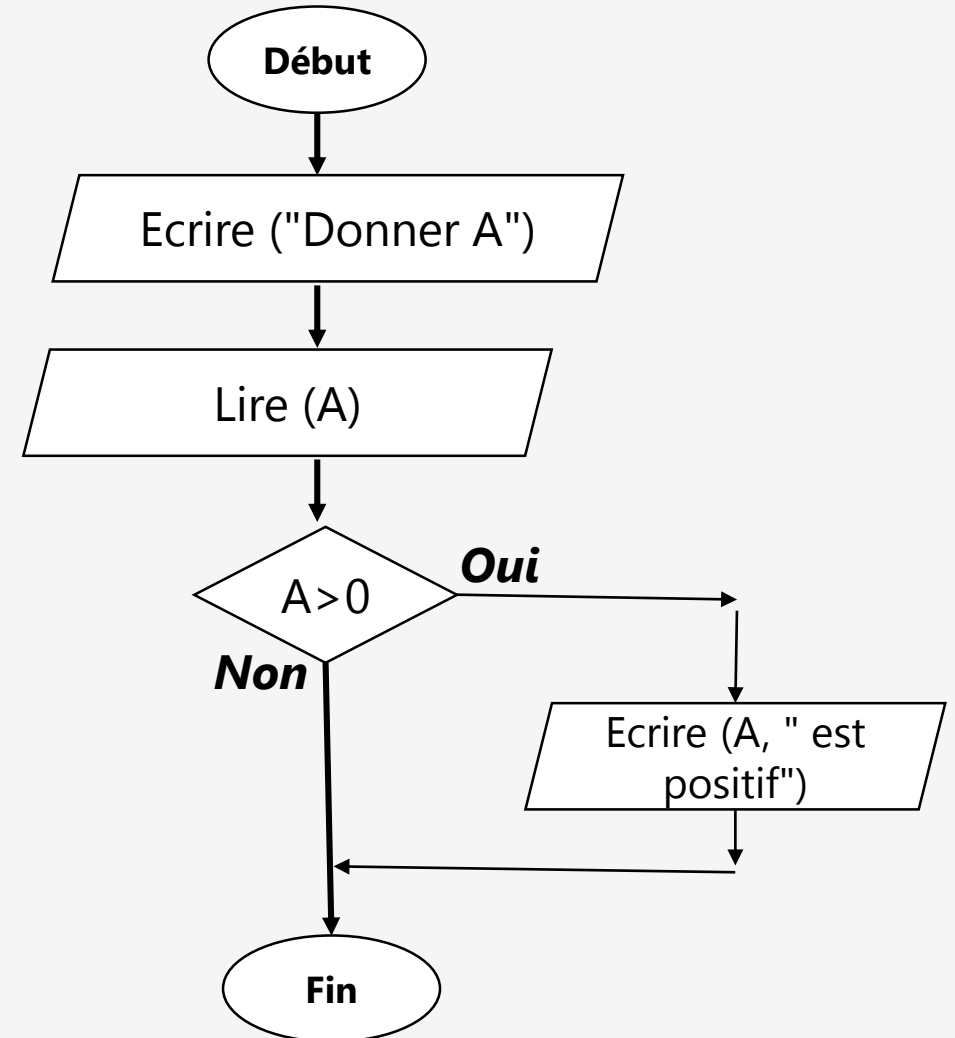
Si A > 0 Alors

Ecrire (A, " est positif")

FinSi

Fin

La partie concernée par
l'organigramme



C- Organigramme

❑ Exemple 3 (structure conditionnelle imbriquée):

Soit l'algorithme suivant

Algorithme maximum_trois_nombres

Variables

Nbr1, Nbr2, Nbr3 : entier

Début

Ecrire ("Veuillez entrer les trois nombres")

Lire(Nbr1, Nbr2, Nbr3)

Si (Nbr1 > Nbr2 ET Nbr1 > Nbr3) Alors

 Ecrire ("Le maximum est", Nbr1)

Sinon

 Si (Nbr2 > Nbr3) Alors

 Ecrire ("Le maximum est", Nbr2)

 Sinon

 Ecrire ("Le maximum est", Nbr3)

 Finsi

Finsi

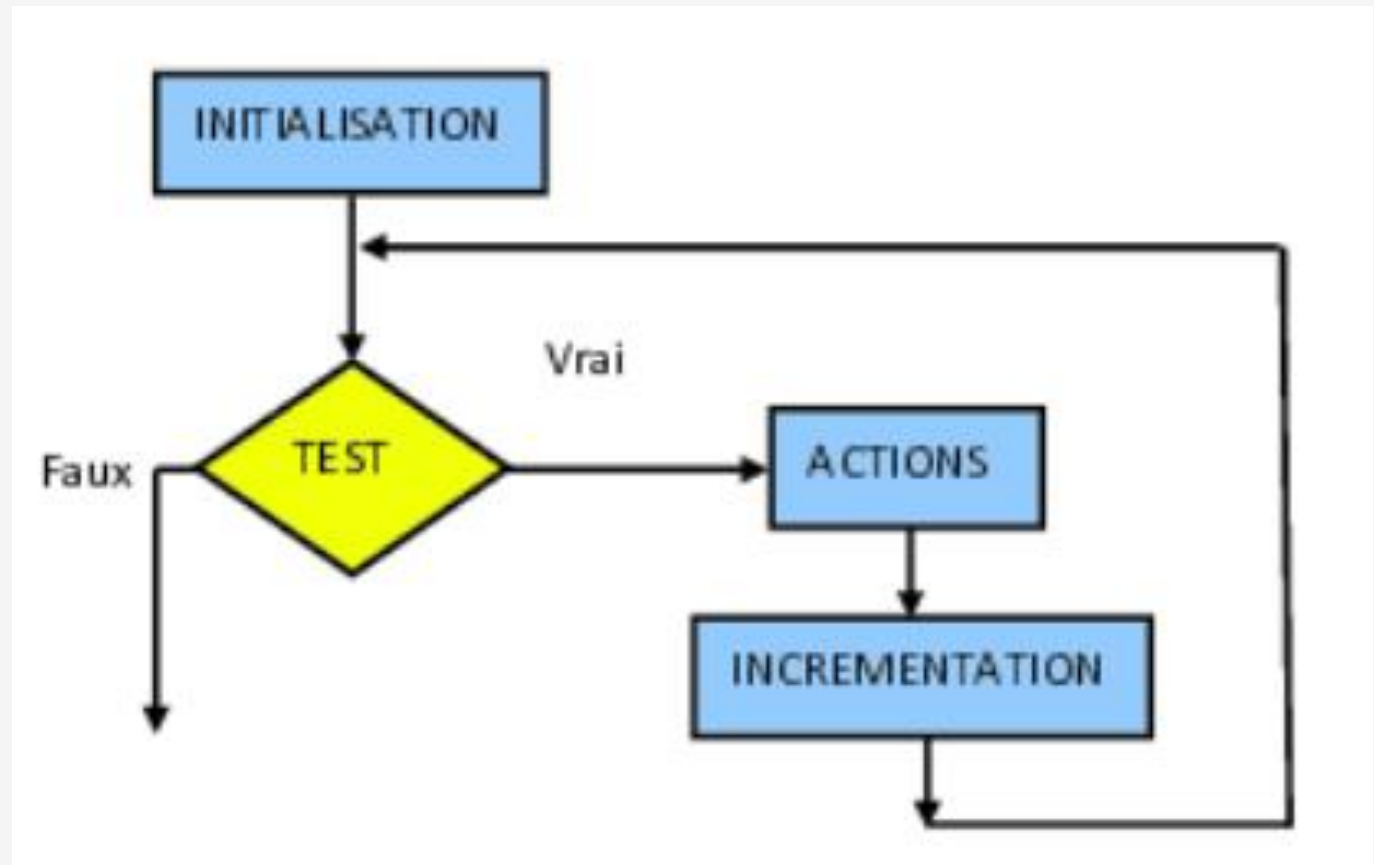
Fin

Question: Construire l'organigramme correspondant à cet algorithme

C- Organigramme

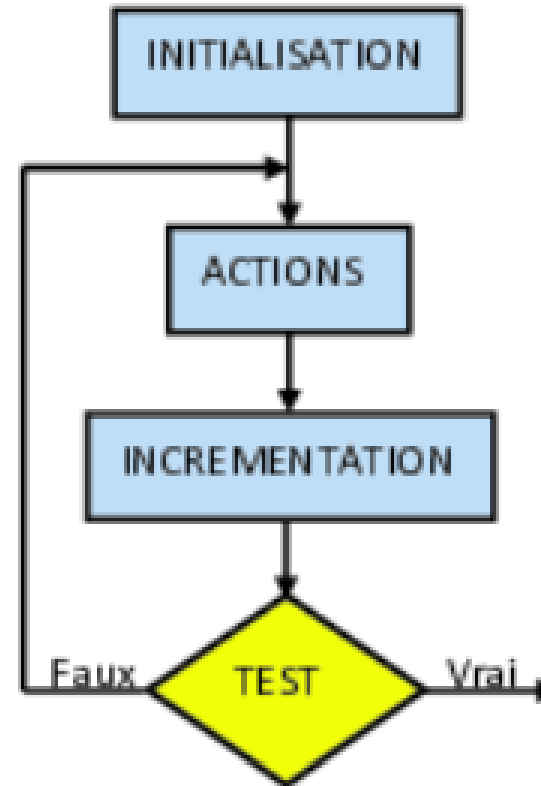
1. La boucle TantQue ... Faire

Organigramme



C- Organigramme

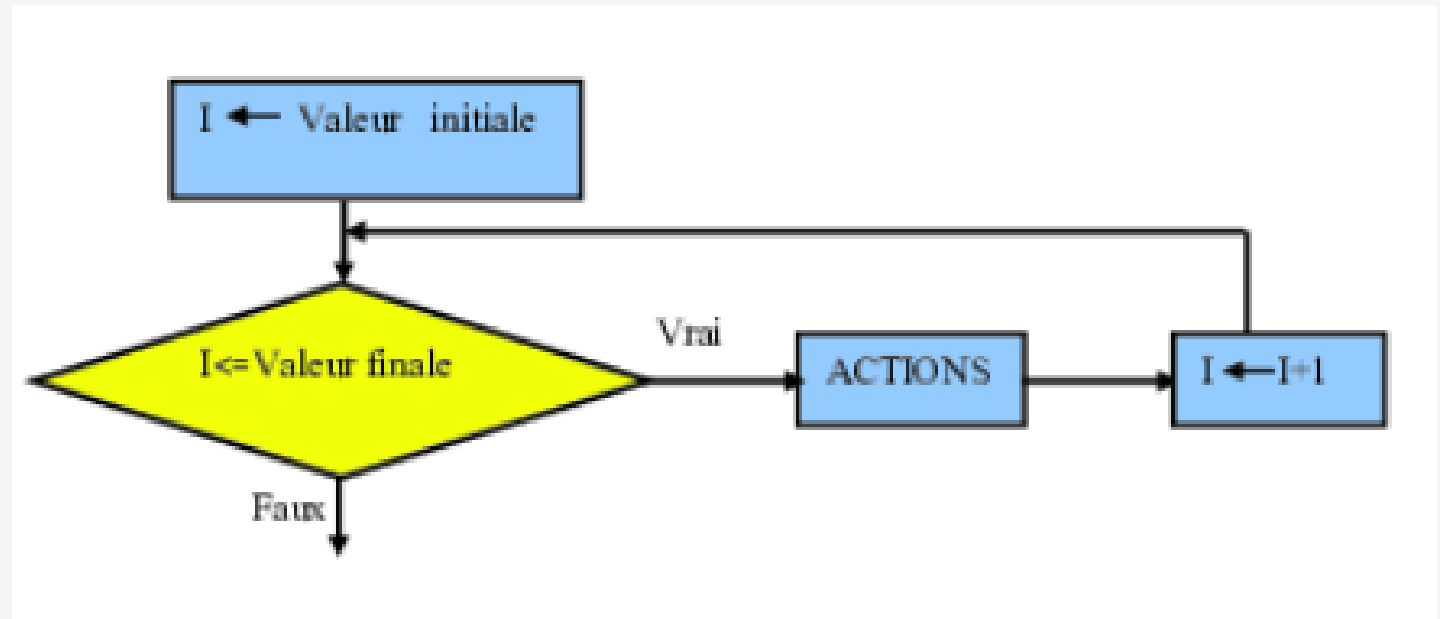
2. La boucle Répéter ... Jusqu'à Organigramme :



C- Organigramme

2. La boucle Pour ... faire

Organigramme :

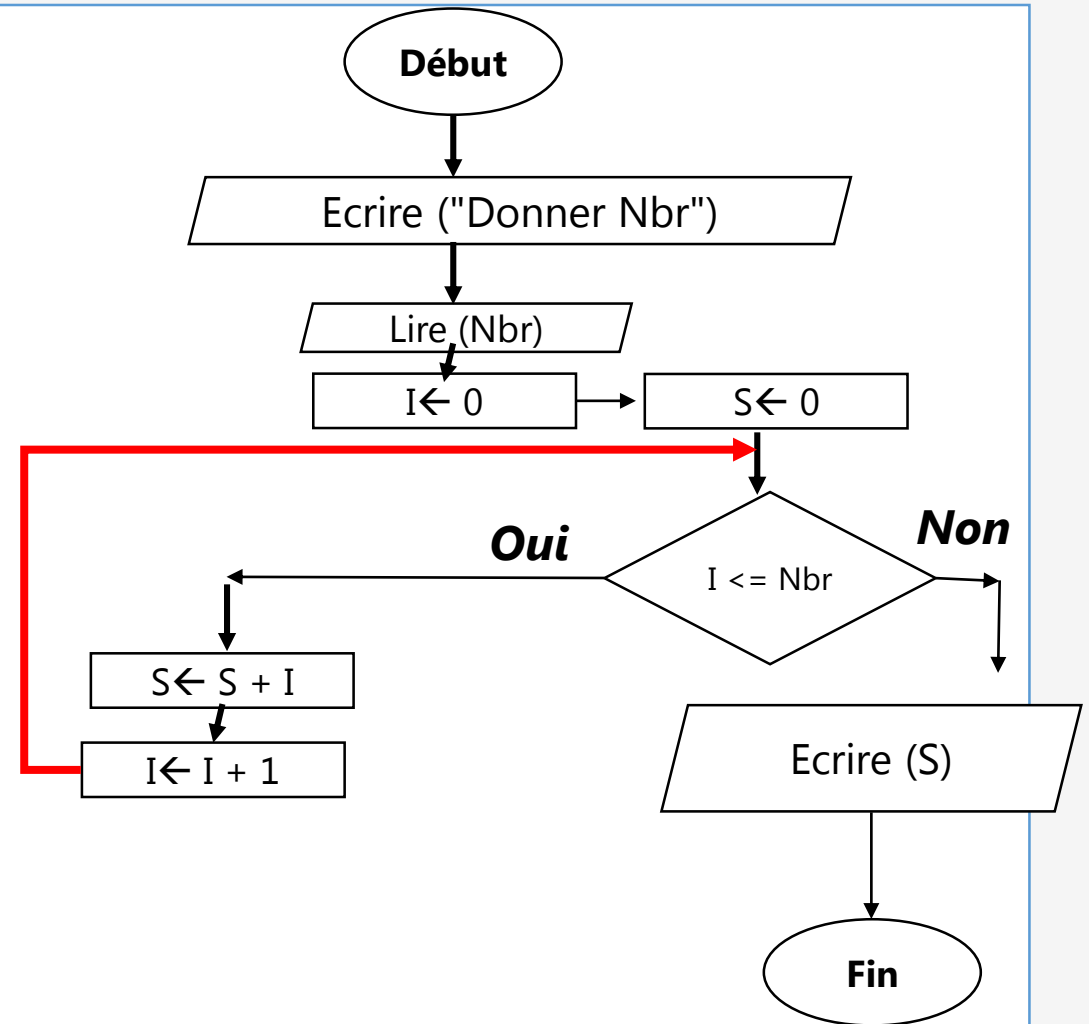


C- Organigramme

❑ Exemple 4 (La boucle TantQue ... Faire):

Soit l'algorithme suivant

```
Algorithme somme_nombres_compri_Zero_Nbr  
Variabe Nbr, S, I : entier  
Début  
Ecrire ("Veuillez entrer un nombre ")  
Lire(Nbr)  
I←0  
S←0  
TantQue I<=Nbr Faire  
    S←S+I  
    I←I+1  
FinTantQue  
Ecrire (" La somme des nombres compris entre 0  
et", Nbr, "est: ", S)  
Fin
```



Partie 2: Algorithmique

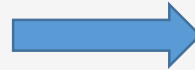
Chapitre 3: Les tableaux

A- Les Tableaux

Introduction

→ Problématique

La nécessité d'un grand
nombre de variables



Difficulté de donner un nom à
chaque variable

Exemple:

Ecrire un algorithme permettant de saisir 7 notes et de les afficher après avoir multiplié toutes les notes par 3.

A- Les Tableaux

Introduction

Algorithme Notes

variables N1,N2,N3,N4,N5,N6,N7: réel

Début

Ecrire ('Entrer la première note')

Lire(N1)

Ecrire ('Entrer la deuxième note')

Lire(N2)

...

...

Ecrire ('La note 1 multipliée par 3 est:',N1*3)

Ecrire ('La note 2 multipliée par 3 est:',N2*3)

...


...

Fin

→ Pour résoudre ce problème, il existe un type de données qui permet de stocker un ensemble de données **de même type** dans une seule variable.

A- Les Tableaux

Variables de type simple prédéfini

- 
- Entier
 - Réel
 - Caractère/Chaine de caractères
 - Booléen

Variable structurée



Variable de type Tableau

A- Les Tableaux

Définition

Un tableau est une suite d'élément de même type. Il utilise plusieurs cases mémoires à l'aide d'un seul nom. Comme toutes les cases portent le même nom, elles se différencient par un indice.

Nous distinguons deux types de tableaux: les tableaux à une dimension (Vecteurs) et les tableaux à plusieurs dimensions (Matrices).

A- Les Tableaux

A.1 Les tableaux à une dimension (vecteur)

On appelle un vecteur V d'ordre n , noté $V(n)$, un ensemble de n éléments de même type disposés dans un tableau à n cases. Un tableau (vecteur) possède un identificateur (nom du vecteur) et un indice qui indique le numéro de la case.

Exemple: Soit T un tableau de 7 cases

| | | | | | | |
|----|---|----|---|---|----|----|
| 13 | 7 | 22 | 4 | 9 | 17 | 23 |
|----|---|----|---|---|----|----|

$T[i]$ est le contenu (l'élément) de la case numéro i .

$T[2]$ est égal à 7, c.à.d l'élément de la case N°2 est 7.

A- Les Tableaux

A.1 Les tableaux à une dimension (vecteur)

A.1.1 Déclaration

La déclaration d'un tableau permet d'associer à un nom une zone mémoire composée d'un certain nombre de cases mémoires de même type.

Syntaxe

Variable **identificateur** : **tableau** [**indice_min ... indice_max**] de **type**

Ou bien

Variable **identificateur** : **tableau** [**taille**] de **type**

Exemple → Variable Note : tableau [1 ... 20] de réels

A- Les Tableaux

A.1 Les tableaux à une dimension (vecteur)

A.1.1 Déclaration

Si on a plusieurs tableaux (vecteurs) de même type et de même taille, on peut les déclarer tous ensemble en les séparant par des virgules

Syntaxe

Variable **identificateur1, identificateur2, identificateur3** : **tableau** [**indice_min** ... **indice_max**] **de** **type**

Exemple → Variable T1, T2, T3 : tableau [1 ... 10] de réels

A- Les Tableaux

A.1 Les tableaux à une dimension (vecteur)

A.1.2 Initialisation d'un élément d'un tableau :

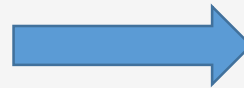
L'initialisation d'un élément du vecteur se fait comme suit:

Syntaxe

Identificateur [indice] \leftarrow expression

Exemple \rightarrow

- Tab [1] \leftarrow -7
- Tab [2] \leftarrow 13
- Tab [3] \leftarrow Tab[2]-Tab[1]
- Tab [4] \leftarrow Tab[1] + 1



| | | | |
|----|----|----|----|
| -7 | 13 | 20 | -6 |
|----|----|----|----|

A- Les Tableaux

A.1 Les tableaux à une dimension (vecteur)

A.1.3 Lecture et affichage des éléments d'un tableau :

la lecture d'un nombre à partir du clavier se fait par l'instruction Lire(x); où x est une variable simple. La même instruction est utilisée pour remplir toutes les cases d'un tableau.

Lire (Tab[1]), Lire (Tab[2]), Lire (Tab[3]), Lire(Tab[n])

Remarque: L'instruction **Lire** est répété n fois. Afin d'éviter la répétition de l'instruction Lire, nous utilisons une des structures itératives que nous avons vu précédemment. Le nombre d'itérations étant connu (n), pour cela nous utilisons la boucle **Pour**.

Remplissage
d'un tableau

Syntaxe

**Pour $i \leftarrow 1$ jusqu'à n Faire
Lire (Tab[i])**

Affichage des
éléments d'un tableau

Syntaxe

**Pour $i \leftarrow 1$ jusqu'à n Faire
Ecrire (Tab[i])**

A- Les Tableaux

A.1 Les tableaux à une dimension (vecteur)

A.1.3 Lecture et affichage des éléments d'un tableau :

- | | | |
|-----------|------------------------------|--|
| Exemple1: | $X \leftarrow \text{Tab}[1]$ | Cette instruction affecte à la variable X la valeur du premier élément du tableau Tab. |
| Exemple2: | Ecrire(Tab[4]) | Cette instruction affiche le contenu du quatrième élément du tableau Tab. |
| Exemple3: | Lire (Tab[2]) | Cette instruction affecte la valeur introduite par l'utilisateur au deuxième élément du tableau Tab. |

A- Les Tableaux

A.1 Les tableaux à une dimension (vecteur)

A.1.3 Lecture et affichage des éléments d'un tableau :

Exercice 1: Ecrire un algorithme permettant de saisir 30 notes et de les afficher après avoir multiplié toutes ces notes par un coefficient fourni par l'utilisateur.

A- Les Tableaux

A.1 Les tableaux à une dimension (vecteur)

A.1.3 Lecture et affichage des éléments d'un tableau :

Exercice 1: Ecrire un algorithme permettant de saisir 30 notes et de les afficher après avoir multiplié toutes ces notes par un coefficient fourni par l'utilisateur.

```
Algorithme TableauNotes
Variable Coef, i : entier
      Notes : [1...30] de réel
Début
Ecrire (" Veuillez entrer le coefficient")
Lire(Coef)
// Remplissage du tableau
Pour i ← 1 jusqu'à 30 Faire
Ecrire (" Entrer la valeur de la note", i)
Lire(Notes[i])
FinPour
// Affichage des notes * coef
Pour i ← 1 jusqu'à 30 Faire
Ecrire (Notes[i]*Coef)
FinPour

Fin
```

A- Les Tableaux

A.1 Les tableaux à une dimension (vecteur)

A.1.3 Lecture et affichage des éléments d'un tableau :

Exercice 2: Ecrire un algorithme qui demande la taille d'un tableau et qui calcule la somme des ses éléments. On suppose que le nombre d'éléments du tableau ne dépasse pas 100.

A- Les Tableaux

A.1 Les tableaux à une dimension (vecteur)

A.1.3 Lecture et affichage des éléments d'un tableau :

Exercice 2: Ecrire un algorithme qui demande la taille d'un tableau et qui calcule la somme de ses éléments. On suppose que le nombre d'éléments du tableau ne dépasse pas 100.

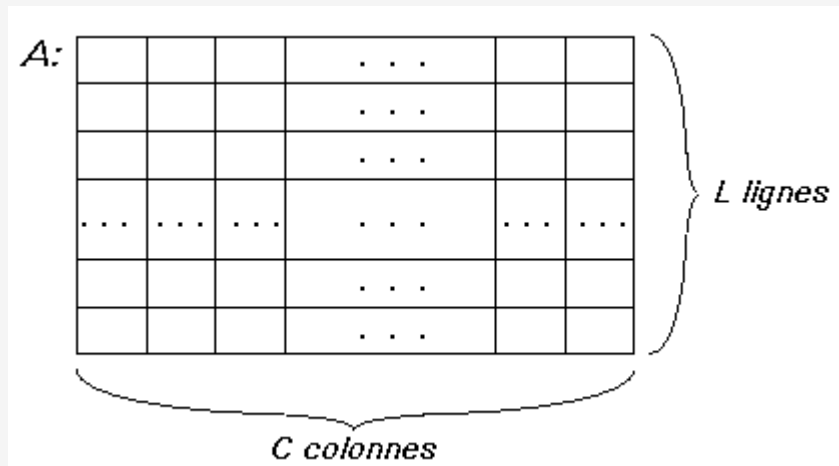
```
Algorithme SommeTableau
Constante M=100
Variable n,i,S : entier
          Tab : tableau [1...M] de réel

Début
Ecrire (" Entrer la taille du tableau")
Lire(n)
Si n>M Alors
    n←M
FinSi
// Remplissage du tableau
Pour i←1 jusqu'à n Faire
Ecrire (" Entrer la valeur de l'élément", i, "
du tableau")
Lire(Tab[i])
FinPour
S ← 0
Pour i←1 jusqu'à n Faire
S ← S+Tab[i]
FinPour
Ecrire (" La somme des éléments du
tableau est:", S)
Fin
```


A- Les Tableaux

A.2 Les tableaux à deux dimensions (Matrice)

- ❑ Un tableau à deux dimensions peut être interprété comme une matrice.
- ❑ Un tableau à deux dimensions contient donc **$L * C$ composantes**.
- ❑ On dit qu'un tableau à deux dimensions est **carré**, si **L est égal à C** .



A- Les Tableaux

A.2 Les tableaux à deux dimensions (Matrice)

Exemple:

Considérons un tableau NOTES à une dimension pour mémoriser les notes de **20 élèves** d'une classe dans un devoir:

Notes : tableau[1...20] de réel

| | | | | | | | |
|--------------|----------|----|----------|-----|-----|-----------|-----------|
| Notes | 15 | 14 | 17 | ... | ... | 18 | 17.50 |
| | Notes[1] | | Notes[3] | | | Notes[19] | Notes[20] |

A- Les Tableaux

A.2 Les tableaux à deux dimensions (Matrice)

Pour mémoriser les notes des élèves dans les 10 devoirs d'un trimestre, nous pouvons rassembler plusieurs de ces tableaux uni-dimensionnels dans un tableau NOTES à deux dimensions :

Dans une ligne nous retrouvons **les notes de tous les élèves dans un devoir**. Dans une colonne, nous retrouvons **toutes les notes d'un élève**.

NOTE :

| | | | | |
|-----|-----|-----|-----|-----|
| 45 | 34 | ... | 50 | 48 |
| 39 | 24 | ... | 49 | 45 |
| ... | ... | ... | ... | ... |
| 40 | 40 | ... | 54 | 44 |

10 lignes

20 colonnes

A- Les Tableaux

A.2 Les tableaux à deux dimensions (matrice)

A.2.1 Déclaration

La déclaration d'un tableau permet d'associer à un nom une zone mémoire composée **d'un certain nombres de cases** mémoires de même type.

Syntaxe

Variable **identificateur** : **tableau** [**1** ... **nbr_lignes**, **1** ... **nbr_colonnes**] de **type**

Ou bien

Variable **identificateur** : **tableau** [**nbr_lignes**, **nbr_colonnes**] de **type**

Exemple → Variable Note : **tableau** [1...3, 1...4] de réels

Note est un tableau de type réel à deux dimensions composé de 3 lignes et de 4 colonnes

M[i, j] est le contenu de la case qui se trouve à la ligne i et la colonne j.

A- Les Tableaux

A.2 Les tableaux à deux dimensions (matrice)

A.2.2 Initialisation d'un élément d'un tableau à deux dimensions :

Comme dans le cas d'un vecteur, l'initialisation d'un élément d'une matrice se fait par l'affectation selon la syntaxe suivante

Syntaxe

Identificateur [indice_ligne, indice_colonne] ← expression

Exemple Soit Tab une matrice d'ordre 2×3 →

Tab [1,1] ← 5

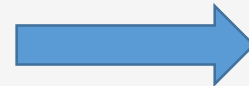
Tab [1,2] ← 13

Tab [1,3] ← Tab[1,2]-Tab[1,1]

Tab [2,1] ← 7

Tab [2,2] ← 0

Tab [2,3] ← Tab[2,1]+Tab[2,2]



| | | |
|---|----|---|
| 5 | 13 | 8 |
| 7 | 0 | 7 |

A- Les Tableaux

A.2 Les tableaux à deux dimensions (matrice)

A.2.3 Lecture et affichage des éléments d'un tableau :

Pour parcourir une matrice nous avons besoin **de deux boucles**, l'une au sein de l'autre (boucles imbriquées). Le remplissage d'une matrice se fait, ligne par ligne ou colonne par colonne, par l'instruction suivante:

**Remplissage
d'un tableau**

Syntaxe

```
Pour i←1 jusqu'à L Faire  
    Pour j←1 jusqu'à C Faire  
        Lire (Tab[i,j])  
    FinPour  
FinPour
```

La première boucle est conçue pour parcourir les lignes, tandis que la deuxième boucle est utilisée pour parcourir les éléments de la ligne précisée par la boucle principale.

A- Les Tableaux

A.2 Les tableaux à deux dimensions (matrice)

Exercice 3: Ecrire un algorithme permettant la saisie des notes d'une classe de **40** étudiants en **5** modules

| N° des étudiants | N° des modules | | | | | |
|------------------|----------------|-----------|-----------|-----------|-----------|-----------|
| | | ModuleN°1 | ModuleN°2 | ModuleN°3 | ModuleN°4 | ModuleN°5 |
| | EtudN°1 | 17.50 | 09.75 | 11.50 | 15.50 | 14.75 |
| | EtudN°2 | 16.00 | 14.25 | 15.50 | 09.50 | 08.00 |
| | | ... | ... | ... | ... | ... |
| | | ... | ... | ... | ... | ... |
| | | ... | ... | ... | ... | ... |
| | EtudN°40 | 15.50 | 12.75 | 10.50 | 13.50 | 19.50 |

A- Les Tableaux

A.2 Les tableaux à deux dimensions (matrice)

Exercice 3: Ecrire un algorithme permettant la saisie des notes d'une classe de 40 étudiants en 5 modules

```
Algorithme NotesModules
Constante Nbr_Etud=40
          Nbr_Mod=5
Variable  Tab : [1..Nbr_Etud, 1...Nbr_Mod] de réel
          i,j : entier

Début
// Remplissage du tableau
Pour i←1 jusqu'à Nbr_Etud Faire
    Pour j←1 jusqu'à Nbr_Mod Faire
        Ecrire (" Entrer la note de l'étudiant numéro", i, " dans
le module", j )
        Lire(Tab[i,j])
    FinPour
FinPour
Fin
```


B- Les Structures

Introduction

Les Tableaux:

- Permettent de stocker sous un **même nom** un ensemble de valeurs de **même type**.
- Chaque valeur est repérée ensuite par un indice.

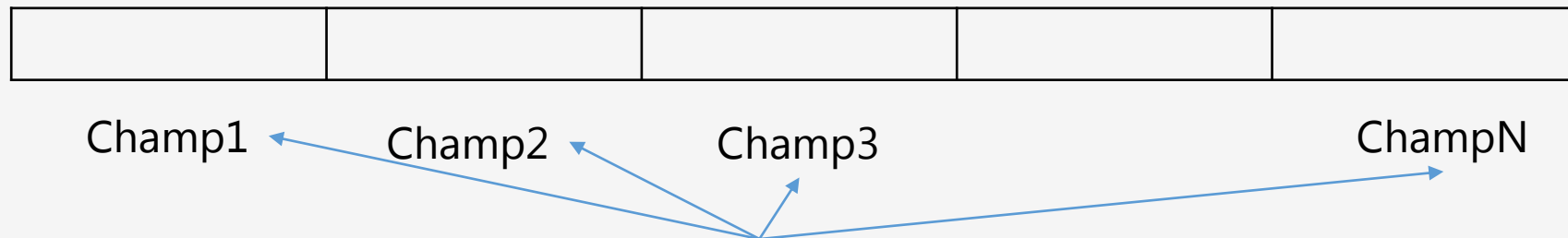
Besoin: Possibilité de stocker des valeurs avec des types différents?

→ Solution: L'utilisation des structures

B- Les Structures

Définition

Les structures permettent de désigner sous un seul nom un ensemble de valeurs pouvant être **de types différents**. L'accès à chaque élément de la structure nommé **champ** se fera, cette fois, **non plus par un indice**, mais par **son nom** au sein de la structure.



Une structure est composée de plusieurs champs

B- Les Structures

B.1. Déclaration d'une structure

Contrairement à la déclaration d'une variable simple ou de type tableau, la **déclaration** d'une structure **ne définit aucune variable**, elle permet de définir **un modèle** de structure. En d'autres termes, déclarer une structure c'est **définir un nouveau type**.

Syntaxe:

```
Type Structure nom_structure  
    nom_champ1: type_champ1  
    ...  
    nom_champN: type_champN  
FinStruct
```

Exemple: Déclaration d'une structure nommée clients

```
Type Structure clients  
    Nom: chaine  
    Prénom: chaine  
    Age: entier  
    Genre: chaine  
FinStruct
```

B- Les Structures

B.1. Déclaration d'une structure

Note importante: La déclaration des structures se fait dans une section spéciale des algorithmes appelée **Type**, qui précède la section des variables.

```
Algorithm STRUCT
Type Structure Etudiant
    champ1: chaine
    champ2: entier
FinStructure
Variables .....
Début
...
Fin
```

B- Les Structures

B.2. Déclaration d'une variable de type structure

Après avoir défini la structure, on peut l'utiliser comme un type normal tel que les types prédéfinis (réel, entier, caractère,...) en déclarant une ou plusieurs variable de ce type.

Syntaxe:

Variable **nom_variable** : **nom_structure**

Exemple:

Variable client1, client2 : clients

client1 est une variable complexe composée de champs qui sont des variables simples ou **complexes** ou des tableaux

| | | | | |
|---------|------------|---------------|------------|--------------|
| client1 | Faridi | Ahmed | 20 | Homme |
| | Nom | Prénom | Age | Genre |

B- Les Structures

B.3. L'accès à un champ d'une structure

Chaque champ d'une structure peut être manipulé comme n'importe quelle variable du type correspondant. L'accès à un champ se fait en faisant suivre le nom de la variable de type structure du nom de champ séparé par un **point**.

Syntaxe:

Variable nom_var.nom_champ

Exemple:

Pour donner un nom au client1:

client1.Nom ← "Faridi"

B- Les Structures

B.3. L'accès à un champ d'une structure

Exercice 1: Ecrire un algorithme qui permet de:

- Définir une structure **Personnel** qui contient trois champs: Code, Nom, Age.
- Déclarer deux variables Fonctionnaire1 et Fonctionnaire2 de type Personnel.
- Remplir tous les champs de ces deux variables
- Calculer la différence d'âge entre les deux

B- Les Structures

B.3. L'accès à un champ d'une structure

```
Algorithme Dif_Age
//Déclaration de la structure Personnel
Type Structure Personnel
    Code : entier
    Nom : chaine
    Age : entier
FinStruct
//Déclaration des variables de type Personnel
Variable Fonctionnaire1, Fonctionnaire2 : Personnel
Début
Ecrire("Taper le code, le nom puis l'age du premier fonctionnaire")
Lire(Fonctionnaire1.Code, Fonctionnaire1.Nom, Fonctionnaire1.Age)
Ecrire("Taper le code, le nom puis l'age du deuxième fonctionnaire")
Lire(Fonctionnaire2.Code, Fonctionnaire2.Nom, Fonctionnaire2.Age)
Ecrire ("La différence d'age entre", Fonctionnaire1.Nom, "et", Fonctionnaire2.Nom, "est:")
Si Fonctionnaire1.Age > Fonctionnaire2.Age
    Ecrire(Fonctionnaire1.Age-Fonctionnaire2.Age, "ans")
Sinon
    Ecrire(Fonctionnaire2.Age-Fonctionnaire1.Age, "ans")
Fin Si
Fin
```


B- Les Structures

B.3. L'accès à un champ d'une structure

Exercice 2:

Personnel est une structure composée de trois champs: Nom, Prénom et date_de_recrutement. Nom et Prénom sont de type chaîne. Date_de_recrutement est de type ***R_date***.

R_date est structure composée de trois champs: jour, mois et année.

Jour, année et mois sont de type entiers.

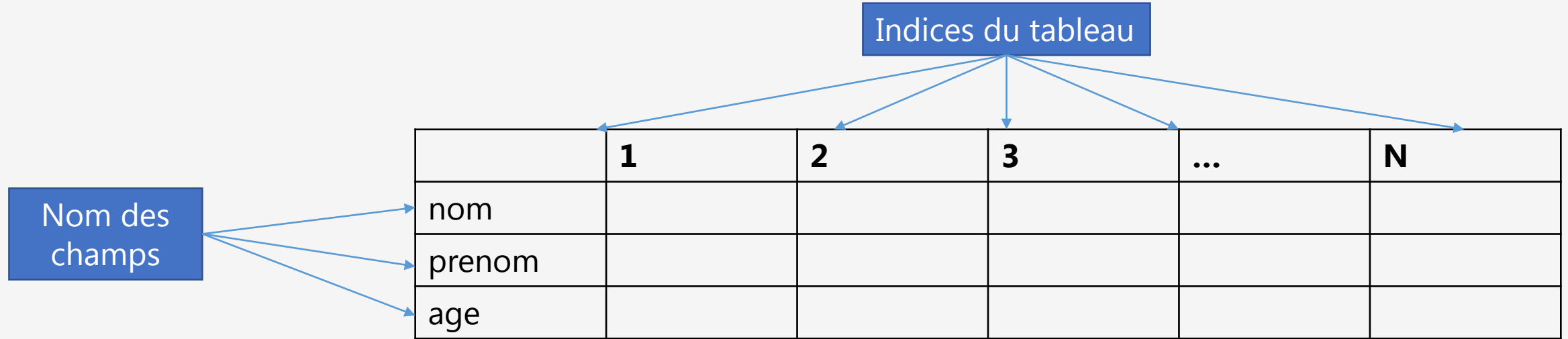
- Ecrire un algorithme qui permet de saisir et d'afficher l'année de recrutement d'un fonctionnaire.
- Donner la taille d'une variable de type *R_date*

B- Les Structures

B.3. L'accès à un champ d'une structure

Exercice 3: Ecrire un algorithme qui demande le nombre de clients d'une entreprise et qui permet de remplir la fiche de tous ces clients (Nom, Prénom, Age, Email)

B- Les Structures



B- Les Structures

Algorithme Fiche

Type Structure client

nom, prenom, email:chaîne

age:entier

Finstructure

Variable info_clt : tableau [1...100] de **client**

i,n: entier

Début

Ecrire (" Entrer le nombre des clients dans l'entreprise")

Lire(n)

Pour i <= n jusqu'à n Faire

Ecrire ("Entrer le nom du client N°", i)

Lire(info_clt[i].nom)

Ecrire ("Entrer le prénom du client N°", i)

Lire(info_clt[i].prenom)

Ecrire ("Entrer l'âge du client N°", i)

Lire(info_clt[i].age)

FinPour

Fin

Chapitre 4: Les fonctions et les procédures

Introduction:

→ Problématique

Dès qu'on commence à écrire des programmes importants, il devient difficile d'avoir une vision globale sur son **fonctionnement** et d'**identifier les erreurs**.

Exemple:

Il nous est demandé d'écrire un algorithme qui permet de vérifier la validité d'un numéro de compte bancaire puis de se connecter à son compte, de choisir l'opération souhaitée et d'effectuer les traitements demandés par l'utilisateur.

Que faire ?

- La solution consiste à découper l'algorithme en plusieurs parties plus petites.
- Ces parties sont appelées des sous-algorithmes ou sous-programmes.

Fonctions

Procédures

Introduction:

→ Définition

- Un sous-algorithme est une suite d'instructions **faisant partie d'un algorithme**. Il est **déclaré** dans la partie entête (avant le début de l'algorithme) puis **appelé** dans le corps de l'algorithme.
- Étant donné qu'il s'agit d'un bloc à part entière, il possède éventuellement un en-tête, une série de traitements, et une gestion des résultats tout comme l'algorithme qui le contient.

Introduction:

But

- ✓ Découper l'algorithme (action globale) en sous-algorithmes (sous-actions) plus simples.
- ✓ Mettre en commun les parties qui se répètent.
- ✓ Avoir un algorithme beaucoup plus petit et bien lisible.
- ✓ Simplifier et faciliter la maintenance du code.
- ✓ Réutiliser dans d'autres algorithmes/programmes.

Introduction:

Types de sous-algorithme

- ✓ Il existe deux sortes de sous algorithmes: les **procédures** et les **fonctions**.

A- Les procédures

A.1 Notion de procédure

A.1.1 Définition

Une procédure est une série d'instructions regroupées sous un nom et **déclarée** dans l'entête de l'algorithme et **appelée** dans son corps (ou dans un autre sous-algorithme) à chaque fois que le programmeur en a besoin.

A- Les procédures

A.2 Déclaration d'une procédure

Syntaxe:

Procédure nom_proc (paramètres : type)

Variables identificateurs : type

Début

Instruction(s) (Corps de la procédure)

FinProc

Remarque :

- 1- Une procédure s'écrit en dehors du programme principal.
- 2- Après le nom de la procédure il faut donner la liste des paramètres avec leur type respectif.
- 3- Une procédure peut ne pas avoir de paramètres.
- 4- Les valeurs des paramètres formels ne sont pas connues lors de la création de la procédure
- 5- **Une procédure ne renvoie aucune valeur.**

A- Les procédures

A.2 Déclaration d'une procédure

Exemple 1 (sans paramètres formels):
Ecrire une procédure qui affiche à l'écran une ligne de 20 étoiles puis passe à la ligne suivante.

Solution:

```
//Déclaration de la procédure étoile  
Procédure Etoiles()  
Variables i : entier  
Pour i ← 1 jusqu'à 20 Faire  
    Ecrire ("*")  
FinPour  
Ecrire ("\n")  
FinProc
```

Exemple 2 (avec paramètres formels):
Ecrire une procédure qui détermine le minimum de deux réels.

Solution:

```
//Déclaration de la procédure minimum  
Procédure minimum(a,b: réels)  
Variables min : réel  
Début  
    si a < b alors  
        min ← a  
    sinon min ← b  
    Ecrire ("Le minimum est", min)  
Finsi  
FinProc
```

A- Les procédures

A.3 L'appel d'une procédure

Pour déclencher l'exécution d'une procédure dans un algorithme, il suffit de **l'appeler**.
l'appel d'une procédure se réalise, à tout endroit de **l'algorithme principal** ou dans **une autre procédure** par une instruction indiquant le nom de la procédure suivi par la liste des paramètres **effectifs** séparés par des virgules.

Syntaxe: nom_procedure (liste des paramètres effectifs)

A- Les procédures

A.3 L'appel d'une procédure

Exemple 3: En utilisant la procédure Etoiles déclarée dans l'exemple 1, écrire un algorithme permettant de dessiner un carré d'étoiles de 20 lignes et de 20 colonnes.

A- Les procédures

A.3 L'appel d'une procédure

Solution →

```
Algorithme carré_étoile
Variable j:entier
//déclaration de la procédure Etoile()
Procédure Etoile()
Variable i: entier
Début
Pour i←1 jusqu'à 20 Faire
    Ecrire("*")
FinPour
Ecrire("\n")
FinProc
//Algorithme principal
Début
Pour j←1 jusqu'à 20 Faire
    Etoile()    //Appel de la procédure
FinPour
Fin
```

A- Les procédures

A.3 L'appel d'une procédure

Exemple 4: En utilisant la procédure minimum déclarée dans l'exemple 2, écrire un algorithme permettant de calculer le plus petit de deux nombres réels différents.

A- Les procédures

A.3 L'appel d'une procédure

Solution →

Algorithme minimum

Variables x,y: réels

//déclaration de la procédure minimum()

Procédure minimum(a,b: réels)

Variables min : réel

Début

si $a < b$ alors min \leftarrow a

sinon min \leftarrow b

Ecrire ("Le minimum est", min)

Finsi

FinProc

//Algorithme principal

Début

Ecrire("Entrer les deux nombres")

Lire(x,y)

//Appel de la procédure minimum

minimum(x,y)

Fin

B- Les fonctions

B.1 Notion de fonction

B.1.1 Définition

- ✓ Une fonction est un bloc d'instructions qui, à partir de donnée(s), calcule et **retourne obligatoirement** à l'algorithme appelant une et une seule valeur résultat de **type simple**.
- ✓ Le résultat d'une fonction peut apparaître dans une expression, dans une comparaison, à la droite d'une affectation.
- ✓ Une fonction n'affiche jamais la réponse à l'écran car elle la renvoie simplement à l'algorithme appelant, alors qu'en général, une **procédure** affiche les résultats demandés.

B- Les fonctions

B.2 Déclaration d'une fonction

Syntaxe:

Fonction nom_fonction (paramètres : type) : **type_fonction**

Variables identificateurs : type

Début

Instruction(s)

Retourner Expression

FinFonction

type de la valeur retournée
par la fonction

renvoie la valeur du résultat

B- Les fonctions

B.2 Déclaration d'une fonction

Remarques:

- 1- Les paramètres sont facultatifs, mais s'il n'y a pas de paramètres, les parenthèses doivent rester présentes.
- 2- Pour le choix d'un nom de fonction il faut respecter les mêmes règles que celles pour les noms de variables.
- 3- **type_fonction** est le type du résultat retourné par la fonction.
- 4- L'instruction **Retourner** sert à retourner la valeur du résultat.

B- Les fonctions

B.2 Déclaration d'une fonction

Exemple 1: Fonction qui calcule la somme des carrées de deux réels A et B.

// Déclaration de la fonction somme carées

Fonction SommeCarre (A: réel, B:réel) : **réel**

Variable S : réel

Début

$S \leftarrow A^2 + B^2$

Retourner S

FinFonction

Exemple 2: Ecrire une fonction qui détermine si un entier est pair?

// Déclaration de la fonction Pair

Fonction Pair (N: entier) : **booléen**

Début

Retourner (N%2=0)

FinFonction

B- Les fonctions

B.2 Déclaration d'une fonction

Exemple 3: Définir une fonction qui renvoie le plus grand de deux nombres différents.

// Déclaration de la fonction Max

Fonction Max (A: entier, B:entier) : **entier**

Début

 Si $A > B$ Alors

 Retourner A

 Sinon

 Retourner B

 Finsi

FinFonction

B- Les fonctions

B.3 L'appel d'une fonction

- ✓ L'exécution d'une fonction se fera par simple écriture de son nom suivie des paramètres effectifs dans le programme principal ou dans une autre fonction. C'est la même syntaxe qu'une procédure.
- ✓ A la différence d'une procédure, la fonction retourne une valeur. L'appel donc d'une fonction pourra donc être utilisé dans une instruction (affichage, affectation, comparaison, ...) qui utilise sa valeur.

Syntaxe:

Nom_fonction (liste de paramètres effectifs)

B- Les fonctions

B.3 L'appel d'une fonction

Exemple 4: Ecrire un algorithme appelant, utilisant la fonction Max de l'exemple 3.

Algorithme Max

// Déclaration de la fonction Max

Fonction Max (A: entier, B:entier) : entier

Début

Si A > B Alors

Retourner A

Sinon

Retourner B

Finsi

FinFonction

// Algorithme principal

Variable X,Y,R : entier

Début

Ecrire("Donner la valeur de deux nombres")

Lire(X,Y)

// Appel de la fonction Max

$R \leftarrow \text{Max}(X,Y)$

Ecrire("Le plus grand de ces deux nombres est",R)

Fin

B- Les fonctions

Remarque importante:

- ✓ **Contrairement à l'appel d'une fonction, on ne peut pas affecter la procédure appelée ou l'utiliser dans une expression.**

C- Passage de paramètres

C.1 Les paramètres

Les paramètres servent à échanger (passer) des données entre le programme principal (ou la procédure/fonction appelante) et la procédure/fonction appelée.

Il existe deux types de paramètres à savoir:

1. Les paramètres placés dans la déclaration (définition) d'une procédure/fonction sont appelés **paramètres formels**.

→ Ces paramètres peuvent prendre toutes les valeurs possibles mais ils sont abstraits (n'existent pas réellement).

2. Les paramètres placés dans l'appel d'une procédure/fonction sont appelés **paramètres effectifs**.

→ Ils contiennent les valeurs pour effectuer le traitement.

Note: Le nombre de paramètres effectifs doit être égal au nombre de paramètres formels. L'ordre et le type des paramètres doivent aussi correspondre.

C- Passage de paramètres

C.2 Types de passage

❑ Passage par valeur:

Dans ce type de passage, le paramètre **formel** reçoit uniquement une copie de la valeur du paramètre **effectif**. La valeur du paramètre effectif ne sera jamais modifiée.

Exemple:

```
Algorithme Passage_par_Valeur
Variable Nbr:entier
Procédure Proc1 (N: entier)
Début
    N ← N - N
Ecrire(N)
FinProc
```

```
//Algorithme principal
Début
Nbr ← 7
Proc1(Nbr)
Ecrire(Nbr)
Fin
```

Résultat →

0
7

→ La procédure n'a pas modifié le paramètre qui est passé par valeur.

C- Passage de paramètres

C.2 Types de passage

❑ Passage par référence ou par adresse:

Dans ce type de passage, la procédure utilise **l'adresse** du paramètre effectif. Lorsqu'on utilise l'adresse du paramètre, **on accède directement à son contenu**. La valeur de la variable effectif pourra donc être modifiée.

→ Les paramètres passés par adresse sont précédés du mot clé **Var**.

Exemple:

```
Algorithme Passage_par_Valeur  
Variable Nbr:entier
```

```
Procédure Proc1 (Var N: entier)
```

```
Début
```

```
    N ← N - N
```

```
Ecrire(N)
```

```
FinProc
```

```
//Algorithme principal
```

```
Début
```

```
Nbr ← 7
```

```
Proc1(Nbr)
```

```
Ecrire(Nbr)
```

```
Fin
```

Résultat →

0
0

→ La procédure Proc1 modifie la valeur de paramètre effectif.

D- Portée des variables

La portée d'une variable désigne le domaine de visibilité de cette variable.
Une variable peut être déclarée dans **deux** emplacements différents.

→ Selon l'emplacement dans lequel une variable est déclarée, on distingue deux types de variables:

Variable locale: Une variable est dite locale à une fonction/procédure si elle est déclarée à l'intérieur de cette dernière. Elle ne peut être utilisée que par des instructions qui se trouvent à l'intérieur de cette fonction/procédure, les autres fonctions n'y ont pas accès.

Variable globale: Les variables globales sont définies en dehors d'une fonction, généralement au-dessus du programme. Les variables globales gardent leurs valeurs tout au long de la durée de vie de votre programme et peuvent être consultées dans n'importe quelle fonction définie pour le programme.

D- Portée des variables

Exemple:

```
Algorithme portée
Variable Nbr : entier
Procédure Proc1 (N: entier)
Variable q : entier
Début
    q ← 0.5
    N ← N + (N*q)
Ecrire(N)
FinProc
//Algorithme principal
Début
Nbr ← 7
Proc1(Nbr)
Ecrire(Nbr)
Fin
```

| | | |
|------------------------|---|-------------------------|
| Nbr est une ... | → | Variable globale |
| q est une ... | → | Variable locale |
| N est ... | → | Paramètre formel |

Note: Les variables globale sont à éviter pour la maintenance des programmes

Exercices

Exercice 1: Soit un exemple de question à laquelle l'utilisateur doit répondre par oui ou par non.

Mauvaise Structure :

```
...  
Ecrire "Etes-vous marié ?"  
Rep1 ← ""  
TantQue Rep1 <> "O" et Rep1 <> "N"  
    Ecrire "Tapez O ou N"  
    Lire Rep1  
FinTantQue  
  
...  
Ecrire "Avez-vous des enfants ?"  
Rep2 ← ""  
TantQue Rep2 <> "O" et Rep2 <> "N"  
    Ecrire "Tapez O ou N"  
    Lire Rep2  
FinTantQue  
  
...
```

Questions:

1. Améliorer cet algorithme en créant une fonction dans laquelle on doit mettre les instructions qui se répètent .
2. Donner l'algorithme principal dans lequel vous devez appeler la fonction.

Exercices

Solution

1

Fonction

```
Fonction RepOuiNon() : caractère  
  Truc ← ""  
  TantQue Truc <> "O" et Truc <> "N"  
    Ecrire "Tapez O ou N"  
    Lire Truc  
  FinTantQue  
  Retourner Truc  
Fin
```

2

Algorithme principal

```
...  
Ecrire "Etes-vous marié ?"  
Rep1 ← RepOuiNon()  
...  
Ecrire "Avez-vous des enfants ?"  
Rep2 ← RepOuiNon()  
...
```


Exercices

Exercice 2:

Ecrire une fonction calculant le périmètre d'un rectangle dont on lui donne la longueur et la largeur.

Longueur: réel

Largeur: réel

Périmètre: réel

Exercice 3:

En utilisant une procédure, Ecrire l'algorithme qui demande un nombre N, calcule et affiche la somme $\sum_{i=1}^N i^3$

Exercices

Exercice 4:

On veut écrire une fonction permettant de calculer le salaire d'un employé payé à l'heure à partir de son salaire horaire et du nombre d'heures de travail.

Les règles de calcul sont les suivantes :

le taux horaire est majoré pour les heures supplémentaires : 25% au-delà de 160 heures et 50% au-delà de 200 heures

Lexique

- sh : réel, salaire horaire
- nbh : entier, nombre d'heures de l'employé
- salaire : réel, salaire de l'employé

Exercices

Solution 2:

```
Fonction périmètreRectangle (Longueur, Largeur : réel) : réel  
Variable périmètre : réel  
Début  
périmètre <-- 2*(Longueur+Largeur)  
Retourner périmètre  
FinFonc
```

Exercices

Solution 3:

```
Algorithme SommeCube
//Déclaration de la procédure
Procédure SomCub (Nbr: entier)
Variable i,S:entier
Début
    S<-- 0
    Pour i<-- 1 jusqu'à Nbr Faire
        S<-- S+(i*i*i)
    Fin Pour
Ecrire ("La somme cubiques des n entiers est:",S)
Fin procédure
//Algorithme principal
Variable N : entier
Début
Ecrire ("Donner un nombre")
Lire (N)
SomCub (N)
Fin
```

Exercices

Solution 4:

```
//Déclaration fonction
Fonction CalculSalaire (Nbr_heure:entier, Sh:réel) : réel
Variables Salaire : réel
Début
    Si Nbr_heure < 160
        Salaire <-- Nbr_heure*Sh
    Sinon Si Nbr_heure < 200
        Salaire <-- 160*Sh + (Nbr_heure-160)*0.25*Sh
    Sinon Salaire <-- 160*Sh + 40*0.25*Sh + (Nbr_heure-200)*0.5*Sh
    Fin Si
Fin Si
Retourner Salaire
Fin Fonc
```