Algorithme

Cours Magistral

Licence Informatique

Dr Edouard Ngor SARR
Enseignant-Chercheur
UFR SES-Université Assane SECK de Ziguinchor (UASZ)
Ziguinchor-Sénégal
Email : edouard-ngor.sarr@univ-zig.sn
Site : www.edouardngorsarr.com

Avril 2025

Sommaire

Chapitre 1
Introduction à l'algorithmique

Algorithmique

- **Informatique** = Sciences du **traitement** automatique de l'information
- **Rôle de l'ordinateur**
  - Le **traitement** d'informations : Algorithmes (Programmes)
  - le stockage d'informations : Fichiers et bases de données
- **Traitement** renvoie à l'exécution d'un ensemble d'Operations pour résoudre des taches (les algorithmes)
- **Un algorithme** est une suite ordonnée d'instructions qui indique la démarche à suivre pour résoudre une série de problèmes équivalents.
- Exemple : Extrait d'un dialogue entre un touriste égaré et un autochtone.
  - Pourriez-vous m'indiquer le chemin du marché Boucote, s'il vous plaît ?
    - Oui bien sur : vous allez tout droit jusqu'au prochain carrefour,
    - vous prenez à gauche au carrefour et ensuite la troisième à droite,
    - et vous verrez le marché juste en face de vous.
- **L'algorithmique** est la science de la conception, de l'analyse et de l'optimisation des algorithmes.
- Le mot algorithme est dérivé du nom du mathématicien qui a vécu au 9ème siècle **Al Khwarizmi**, et qui était membre de l'académie des sciences à Bagdad.

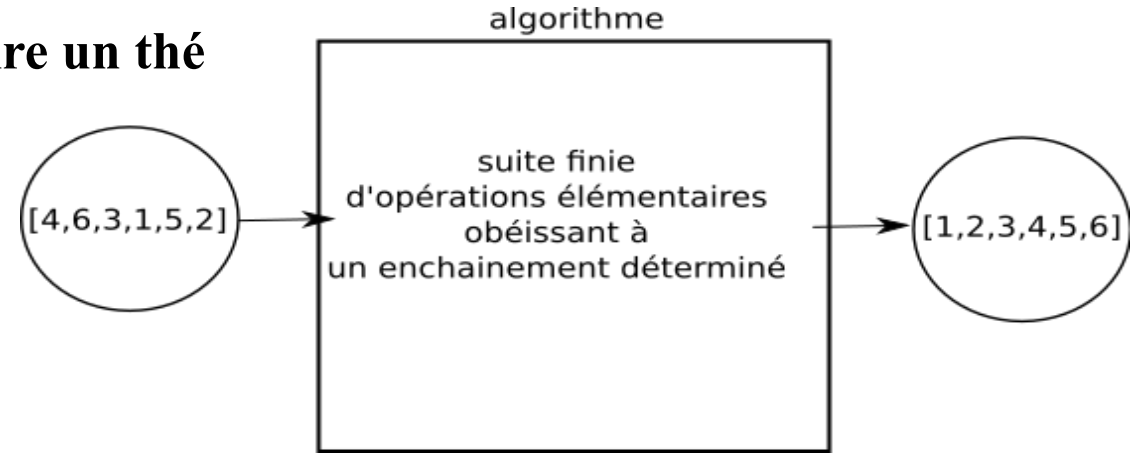
Algorithme : Définitions

1. Un ensemble d'opérations **ordonnées** et **finies** pour résoudre un problème.
2. Suite ordonné d'instructions (Ordres) écrite en langage humain (pseudocode) mais formel (bien codifié) pour résoudre un problème bien défini.
3. Une suite d'instructions ou d'étapes logiques permettant de résoudre un problème ou d'accomplir une tâche.
4. **Un algorithme, c'est une suite d'instructions, qui une fois exécutée correctement, conduit à un résultat donné**.

- Un algorithme est écrit en langage humain : Wolof, Sérère, Français, Anglais
- Un algorithme
  - Prend toujours un ensemble de données entrée (Nombres, Textes, Images, Vidéos …)
  - Effectue un traitement (suite d'opérations) sur ces données (Nombres, Textes, Images …)
  - Génère une ou des données en sorties (L'information / le résultat / le but de l'algorithme)

Exemple 1 : Algorithme pour faire un thé

1.Faire bouillir de l'eau.
2.Mettre un sachet de thé dans une tasse.
3.Verser l'eau chaude dans la tasse.
4.Laisser infuser 3 minutes.
5.Retirer le sachet de thé.
6.Ajouter du sucre si besoin.
7.Boire le thé.

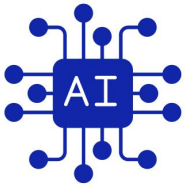


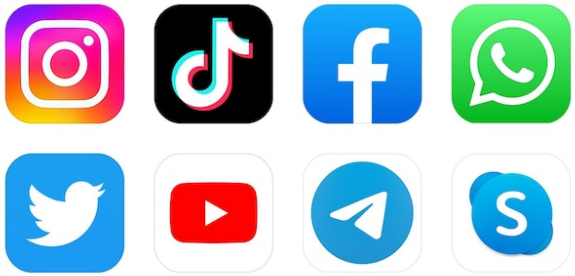
Algorithme, Programme, Application et Logiciel

- **Un algorithme** est un ensemble d'opérations écrites en langage humain (pseudocode) mais formel (bien codifié) pour résoudre un problème bien défini.
- **Un programme** est un algorithme écrit en langage machine (langage de programmation) et destiné à être exécuté par un ordinateur.
  - Exemple : Addition(X entier, Y Entier)
- **Une application** est un ensemble de programmes destinée à un usage particulier. Selon le format et la support nous avons :
  - Une application mobile (ex: WhatsApp, Wave …),
  - Une application web (ex: Gmail; Campusen …),
  - Une application bureautique (ex: Microsoft Word, Visual Code…).
- **Un logiciel** est un terme **générique** pour désigner **les programmes** et les applications). On parle souvent de
  - **Logiciel système** (comme Windows, Linux)
  - **Logiciel applicatif** (comme VLC, Photoshop) :
  - Toutes les applications sont des logiciels mais tous les logiciels ne sont pas des applications (Exemple des logiciels système)

Cas d'usage des algorithmes

- Aujourd'hui tout est algorithme surtout l'IA (Machine Learning)
- Sont omniprésents dans tous les secteurs d'activités
  - Santé : Epidémiologie, Consultance, Prise de RDV etc
  - Marketing : Segmentation, Personnalisation, Recommandation etc
  - Agriculture : Irrigation, Commercialisation, Détection des intrusions etc
  - Education : Orientation, Evaluation
  - Jeux : 1xBet
  - Loisir : site de rencontre, Réseaux sociaux
  - Météo
  - Documentation
    - Moteur de recherche
    - IA générative





Google

ChatGPT

1XBET

deepseek

Objectif et problèmes

- Un algorithme est utilisé pour Apprendre à l'ordinateur à résoudre un problème
- **Défis à surmontés**
  - **Comment résoudre le problème ?**
    1. Connaitre les différentes étapes
    2. Connaitrais les données nécessaires en entrée et le résultats attendus en sortie
  - **Comment transmettre à l'ordinateur ce savoir faire ?**
    - Communication avec l'ordinateur : Input/Output
    - Gestion des erreurs et des exceptions par l'ordinateur
  - **Comment évaluer le traitement et le résultat de l'ordinateur ?**

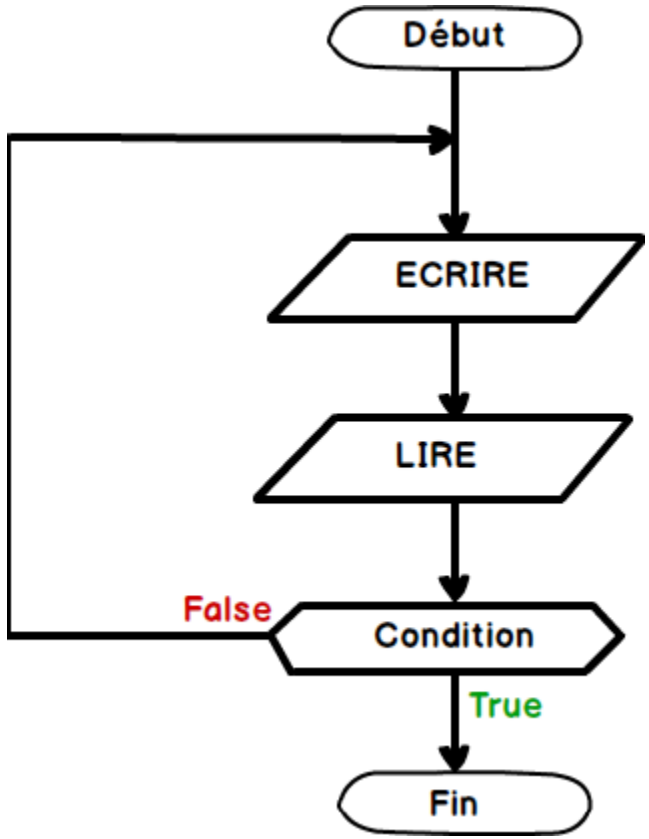
Problèmes des algorithmes

- **La Calculabilité d'un algorithme**
  - Dans ma démarche de résolution, existe-t-il des tâches pour lesquelles il n'existe aucun solution ?
  - C'est l'étude de ce qui peut être calculé ou non par une machine
  - Un problème non calculable signifie qu'aucun algorithme ne pourra jamais le résoudre dans.
  - Objectif : Savoir si un problème peut être résolu par un algorithme, peu importe le temps ou les ressources nécessaires.
- **La Correction d'un algorithme**
  - La validité : est-ce qu'il donne le bon résultat pour toutes les entrées ?
  - Peut-on être sûr que notre algorithme trouvera la solution attendue ?
  - Deux formes :
    - Correction partielle : L'algorithme donne le bon résultat s'il termine.
    - Correction totale : L'algorithme termine toujours et donne le bon résultat.
  - Prouver la correction, c'est démontrer que pour toute entrée valide, la sortie est correcte.
- **La Complexité d'un algorithme**
  - C'est l'étude des ressources nécessaires à l'exécution d'un algorithme : temps (temps d'exécution) et espace (mémoire utilisée).
  - Types :
    - Complexité temporelle : Combien d'opérations sont nécessaires ?
    - Complexité spatiale : Combien de mémoire est nécessaire ?
  - On cherche souvent les algorithmes les plus efficaces, surtout pour les grands volumes de données.

Représentation des algorithmes

- Les algorithmes sont présentés
  - Texte structuré en langue Humaine
  - **Pseudocode (langage simplifié proche du code)**
  - Organigramme (représentation graphique en blocs)

```
DÉBUT
    Afficher : « Quel est votre age ? »
    Entrer : age
    SI ( age < 18 )
        Afficher : « Vous êtes mineur »
    SINON
        Afficher : « Vous êtes majeur »
    FIN SI
FIN
```



Chapitre 2
Fondamentaux de l'algorithme

Structure générale d'un algorithme

- **Un algorithme est composé de 03 Parties dont deux obligatoires**
  - L'Entête de l'algorithme (Obligatoire)
  - La Zone de déclaration des variables et des constantes
    - Toutes les variables et constantes doivent être déclarer ici avant leur utilisation
  - La Zone d'appel de fonctions et procédures externes
  - Le corps ou la zone à exécuter (Obligatoire)
    - Cette zone commence par DEBUT et se termine par FIN

L'en-tête — *Algorithme* NomAlgorithme ;

La partie déclarative
- **Constantes** Identificateur=valeur ;
- **Variables** Identificateur : type ;
- **Structures** Identificateur : type ;

Les fonctions et les procédures
- **fonctions**
  Liste des fonctions
- **Procédure**
  Liste des procédures

Le corps de l'algorithme
- **Début**
  Instruction1 ;
  Instruction2 ;
  ……;
  Instruction2 ;
- **Fin Algoritme**

Algorithme en pseudo-langage

Algorithme du calcul de la distance

```
Variables
    a (réel), b (réel)
Début
1   | a ← Saisir("Valeur de a ? ")
2   | b ← Saisir("Valeur de b ? ")
3   | Si a > b alors :
4   |     | d ← a − b
5   | Sinon :
6   |     | d ← b − a
7   | Fin Si
8   | Afficher("La distance vaut : ", d)
9   Fin
```

Les variables & les constantes

- Ont des structures de données utilisées en informatique pour stocker temporairement en mémoire RAM des données.
- Sont des sortes de « boites » dans lesquelles on met une valeur (un nombre, un mot…)
- Sont toujours caractérisées par :
  - **Un identificateur :** Le nom qui est attribué
    - Composé de lettres seulement ou des lettres et de chiffres sans des espaces ou des caractères spéciaux (que -@,/.+?¨*£°ë‴«Çøø}ë‟{¶«) sauf le (_)
    - Commence toujours par une lettre. Par exemple :
      - » **Prenom1** est valide
      - » **Mon_nom** est valide
      - **1prenom** n'est pas valide
      - **Mon Prenom** n'est pas valide
  - **Un type:** son domaine d'existence des valeurs qu'il va stocker. Les types primitifs sont :
    - ENTIER: Nombre entiers
    - REEL : (Nombre à virgule)
    - CHAINE: Chaine de caractere, Texte
    - DATE : une date (jj/mm/AAAA)
    - BOUL (Oui/Non) : Ne peuvent prendre que deux valeurs
  - **Une valeur :** toujours de même type
- **Différence entre variable et constante**
  - Lors de l'exécution du programme la valeur de la variable **peut changer** alors que celle de la Constantes **ne peut pas changer**

Les variables & les constantes: Déclaration

- La déclaration consiste à réserver une place en mémoire VIVE (la RAM)
- Elle est Obligatoire et consiste:
  - Attribuer l'identificateur : un nom
  - Attribuer un type à cette place
  - Attribuer une valeur (**Initialisation ou le cas des constantes**)
- Possibilité de faire une déclaration multiple : Déclarer plusieurs variables en même temps.
- Attention: Une constante est toujours initialisée (déclarée avec sa valeur)
- **Exemple**

  - **Variables**

    X : INT

    i , j , k INT

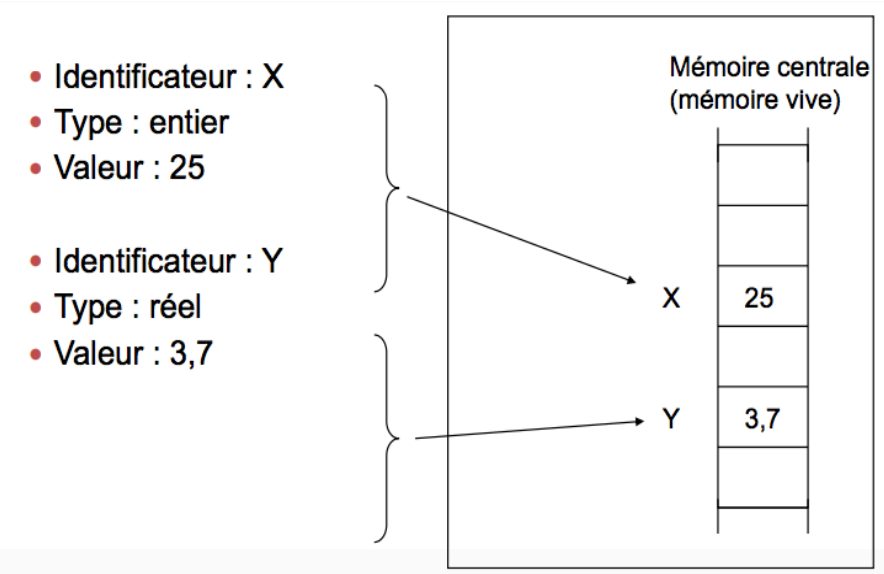
    Min ← 12,5 : DOUBLE

  - **Constantes**

    Pays ← 'SENEGAL' : STRING

    Pie ← 3,14: DOUBLE

    Sexe ← 'M' : BOOL ('M','F')



Les opérateurs

- **les opérateurs arithmétiques**

+	(addition)	*	(multiplication)
%	(Modulo)	-	(soustraction)
/	(division)	^	(exponentiation)

- **Les opérateurs logiques**

  ET (AND)

  OU (OR)

  NON (!)

A	B	A ET B
FAUX	FAUX	FAUX
FAUX	VRAI	FAUX
VRAI	FAUX	FAUX
VRAI	VRAI	VRAI

A	NON A
FAUX	VRAI
VRAI	FAUX

A	B	A OU B
FAUX	FAUX	FAUX
FAUX	VRAI	VRAI
VRAI	FAUX	VRAI
VRAI	VRAI	VRAI

- **L'opérateur concaténation**

  + (réalisant la concaténation de deux chaînes de caractères)

- **Operateurs d'incrémentation et de décrémentation**

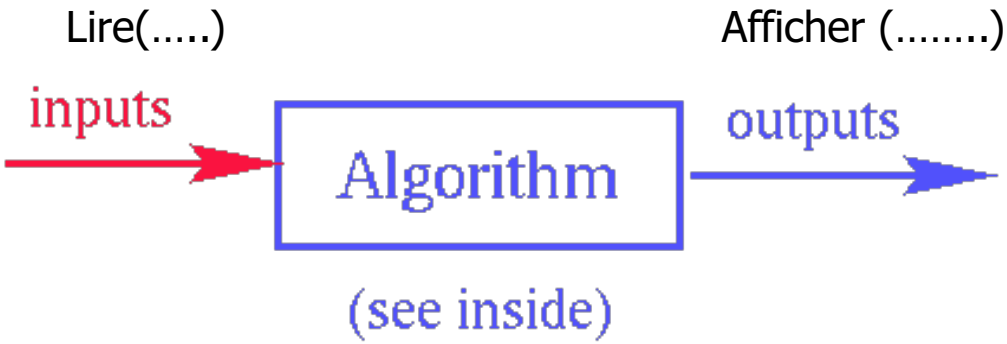
  ++ -- +<-- -<-- *<-- /<--

- **Affectation (←) :** permet d'affecter, à une variable, une nouvelle valeur.
- **Les operateurs de comparaison**

= (égal).	!= different	< (inférieur)	<= (inférieur ou égal)
<> (différent)		> (supérieur)	>= (supérieur ou égal)

Les Entrées / Sorties : Lire() et Afficher()

- Au cours de l'exécution d'un programme, il est indispensable que l'utilisateur puisse communiquer avec le programme :
  - Entrer des données au programme grâce au clavier (Lecture / Input)
  - Recevoir des données ou message du programme grâce à l'écran (Afficher/ Output)



- NB: Chaines de caractères et dates sont mises entre cotes ' ' lors de l'affichage
- Exemples
  - Afficher('Bonjour LMIO')
  - Afficher(100)
  - Afficher('le plus grand est ' , max)
  - Afficher ('la racine carrée de ' , X , ' est ' , R)
  - Afficher (x+y)
  - Lire (x)
  - Lire (x , y) *lecture de deux valeurs saisies par l'utilisateur*

Exemple 1

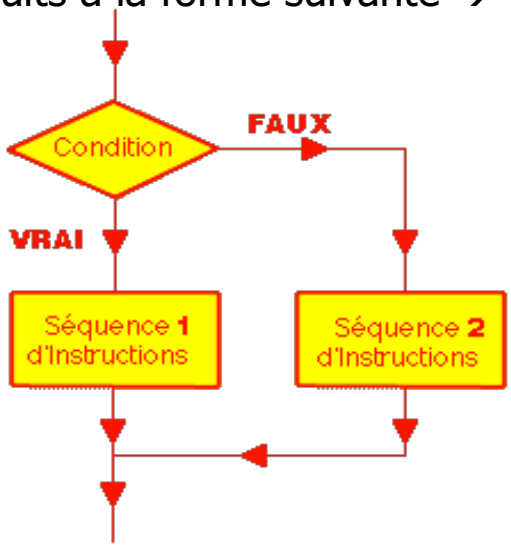
```
VARIABLE
    X : INT
    Y , Z , k INT
DEBUT
    X ← 10
    Z ← 2
    RES ← X*Y
FIN
```

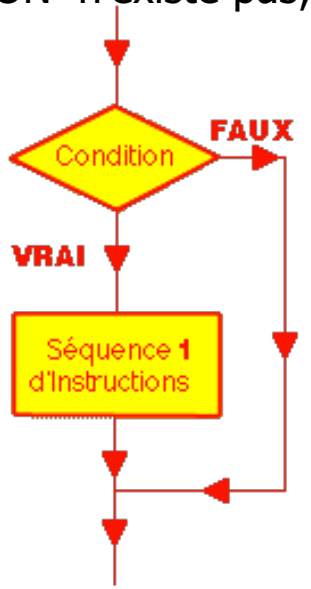
```
ALGORITHME Epicier;
VAR prix_total , prix_du_kg : REEL;
    nb_kg : ENTIER;
DEBUT
    Ecrire('Entrez le prix d'un kilogramme de choux : ');
    Lire(prix_du_kg);
    Ecrire ('Entrez le nombre de kilogramme de choux : ');
    Lire (nb_kg);
    prix_total <- prix_du_kg * nb_kg;
    Ecrire ('Le prix total de l''achat est :',prix_total);
FIN .
```

```
X <-- 1        ( X prend la valeur 1 )
X <-- 2*3+5    ( X prend la valeur du résultat de l'opération 2*3+5)
D <-- D+1      ( D augmente de 1)
prix_total <- nb_kg * prix_du_kg  (Si nb_kg est de type entier et
                                   prix_du_kg est de type  réel alors
                                   prix_total doit être de type réel)
```

Instructions conditionnelles: SI

- Structure de contrôle SI permettant de gérer des cas et de rendre conditionnelle l'exécution de séquences d'instructions.
- Si la condition est vraie le programme exécute la séquence1 d'instructions puis il se poursuit après l'instruction finsi
- Sinon (condition fausse) il exécute la séquence2 d'instructions.
- **Remarque**: Dans certain cas le SINON n'existe pas; la syntaxe et le schéma sont alors réduits à la forme suivante →






```
Si ( condition ) Alors
    bloc d'instructions n°1
Sinon
    bloc d'instructions n°2
Fin de Si
```

```
1  Si ( ( x > −1 ) et ( x < 1 ) ) Alors
2      Afficher ( "Bonjour !" )
3  Fin de Si
```

```
Si ( condition ) Alors
    bloc d'instructions n°1
Fin de Si
```

Instructions conditionnelles: SELON

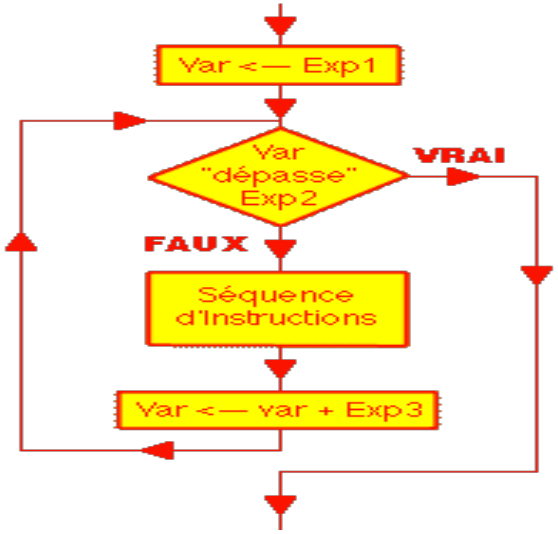
- SELON indique le traitement à faire selon la valeur d'une variable.
- Gere aussi des cas possibles d'une variable
- La **condition 1 est évaluée**
  - Si la condition 1 est vraie, alors on exécute l'action correspondante et on quitte la structure selon-que
  - Si la condition 1 est fausse, on évalue la condition 2...et ainsi de suite.
- Si **aucune n'est vraie** on effectue l'**action sinon** (au cas où l'action sinon n'existe pas alors aucune action n'est exécutée !).

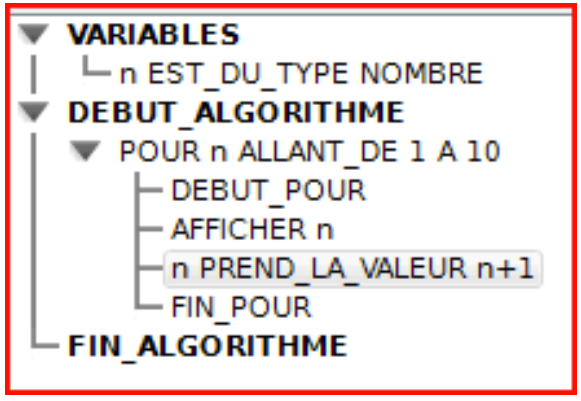
SELON abréviation
"M" : afficher(" Monsieur ")
"Mme" :afficher(" Madame ")
"Mlle" : afficher(" Mademoiselle ")
autres :afficher(" Monsieur, Madame ")
FIN SELON

SELON
Note ≥ 16 : ECRIRE ("TB")
Note ≥ 14 : ECRIRE ("B")
Note ≥ 12 : ECRIRE ("AB")
Note ≥ 10 : ECRIRE ("Passable")
SINON : ECRIRE ("ajourné")
FIN SELON

Instructions Itératives : POUR

- Une boucle est un instruction qui permet de repeter une ou plusieurs instructions N fois.
- Trois Types de boucles : POUR, TANT QUE et FAIRE TANT QUE
- **La boucle POUR**
  - Elle tourne entre deux bornes (MIN et MAX) et dispose d'une variable de contrôle ou indice i et d'un pas P
    - A l'origine i ← MIN
    - Tant que i <= MAX alors la séquence d'instructions continue a être exécutée et à chaque tour i est incrémenté de la valeur décrite par le pas P (P←1 le plus souvent) ou décrémentée (P←-1) et est à nouveau testé avec MAX.
    - Si i > MAX alors on sort de la boucle

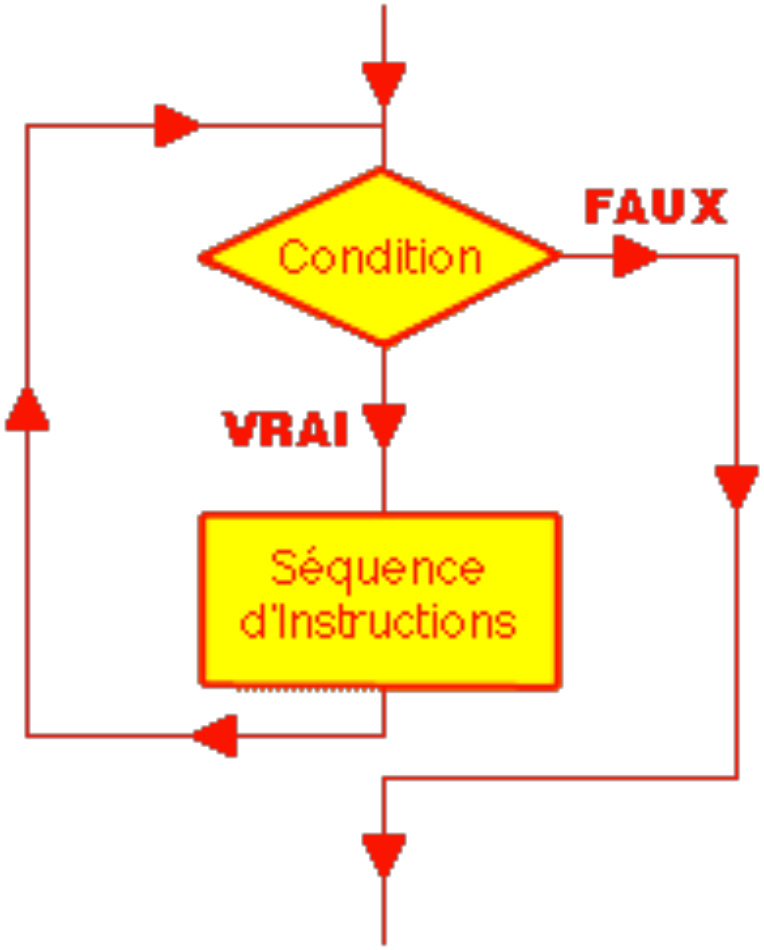





```
Algorithme DessineEtoiles (n : entier)
Variable i : entier
Début
  Pour i ← 1 à n faire
    Écrire('*')
  Fin pour
Fin
```

Instructions Itératives : TANT QUE

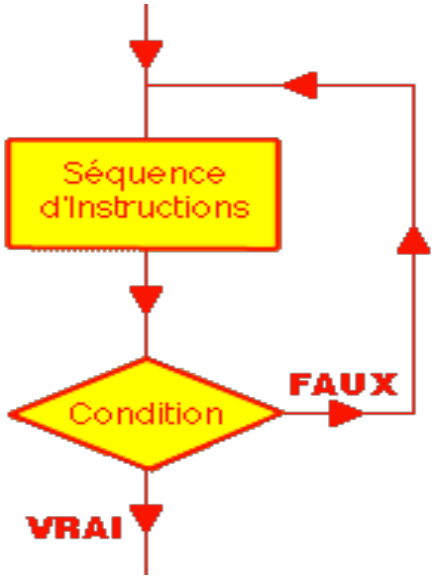
- **La boucle TANT QUE**
  - Le test est fait en début de boucle (on peut donc simuler une boucle "pour" à l'aide d'une boucle "tant que").




```
VARIABLES
    N EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
    N PREND_LA_VALEUR 1
    TANT_QUE (pow(2,N)<10000) FAIRE
        DEBUT_TANT_QUE
        N PREND_LA_VALEUR N+1
        FIN_TANT_QUE
    AFFICHER N
FIN_ALGORITHME
```

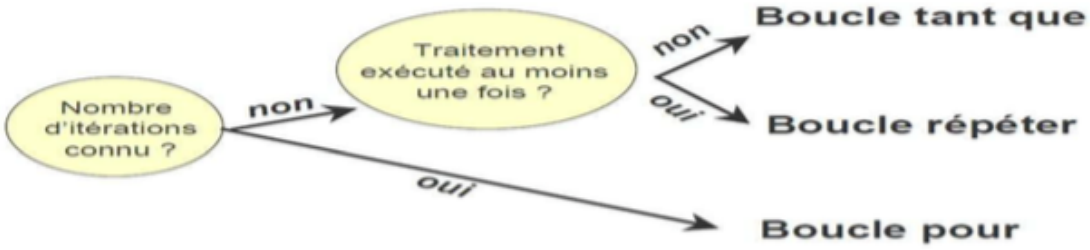
Instructions Itératives : FAIRE - TANT QUE

- Encore Appeler : REPETER - TANT QUE
- Le test est fait en fin de boucle (on peut aussi simuler une boucle "pour" à l'aide d'une boucle »Faire tant que").



- La Condition est évaluée après chaque itération
- Les instructions entre Répéter et jusqu'à sont exécutées au moins une fois
- Leur exécution est répétée jusqu'à ce que condition soit vrai (tant qu'elle est fausse)

```
Algorithme CompteJusqueCentVersionRepeter
Variable i : entier
Début
  i ← 1
  Répéter
    Écrire(i)
    ALaLigne
    i ← i+1
  Jusqu'à (i > 100)
Fin
```



Chapitre 3
Tableaux et fonctions de Tri

Les tableaux

- Supposons que nous souhaitons conserver les notes des 05 étudiants et calculer la moyenne de la classe
- **Solution 1: Itiliser des variables.**

Algorithme Note

Variables

N1, N2, N3, N4, N5 : Réel

Moy: Réel

Début

Ecrire ("Entrer la valeur de la 1er note")

Lire (N1)

Ecrire ("Entrer la valeur de la 2ème note")

Lire (N2)

Ecrire ("Entrer la valeur de la 3ème note")

Lire (N3)

Ecrire ("Entrer la valeur de la 4ème note")

Lire (N4)

Ecrire ("Entrer la valeur de la 5ème note")

Lire (N5)

Moy ← (N1+N2+N3+N4+N5)/5

Fin

Supposons maintenant que le nombre d'étudiants est de 300

Les tableaux

- Solution 2: Regrouper toutes ces notes (N1, N2, … N30) dans une seule variable structurée composée de 30 cases juxtaposées de même nom et de même type.
  - Attribuer un seul nom à l'ensemble des 300 notes, par exemple N.
  - Repérer chaque note par ce nom (N) suivi entre crochets d'un numéro entre 1 et 300 : N[1], N[2], … N[30]
- C'est type de structure de données sont appelé **des tableaux**
- Un Tableau
  - **Une structure de données capable de contenir en même temps plusieurs valeurs de même type.**
  - **Une suite de cases (espace mémoire) de même taille destinée à stocker et manipuler un ensemble de valeurs de même type**
- Deux types de tableux en programmation

Tableau à une dimension

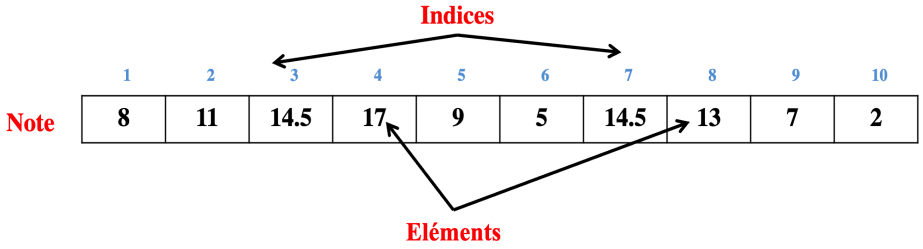


Tableau à deux dimensions

	colonne 0	colonne 1	colonne 2	colonne 3
ligne 0	tab[0][0]	tab[0][1]	tab[0][2]	tab[0][3]
ligne 1	tab[1][0]	tab[1][1]	tab[1][2]	tab[1][3]
ligne 2	tab[2][0]	tab[2][1]	tab[2][2]	tab[2][3]

Tableau à une dimension

- **Une variable structurée formée d'un nombre entier N de variables simples du même type :** 01 seule ligne + N colonnes ou vis versa (01 colonne + N lignes)
- **La dimension ou la taille du tableau (N) =** Le nombre de cellules
- **Indice i** est numéro de la cellule.
  - Indice commence par 0 et s'arrête à (N-1).

Pour un tableau de 10 éléments (N=10), Premier indice 0 et dernier indice 9

- **Déclaration**

  Nom_Tableau[Taile] : Type

  Tab[10] : Entier

  Tab_Prenom [20]: Chaines
- **Tab[1]** désigne le contenu de la cellule numéro 1, la deuxième cellule du tableau

  X= Tab[5]

  Tab[5] = 23

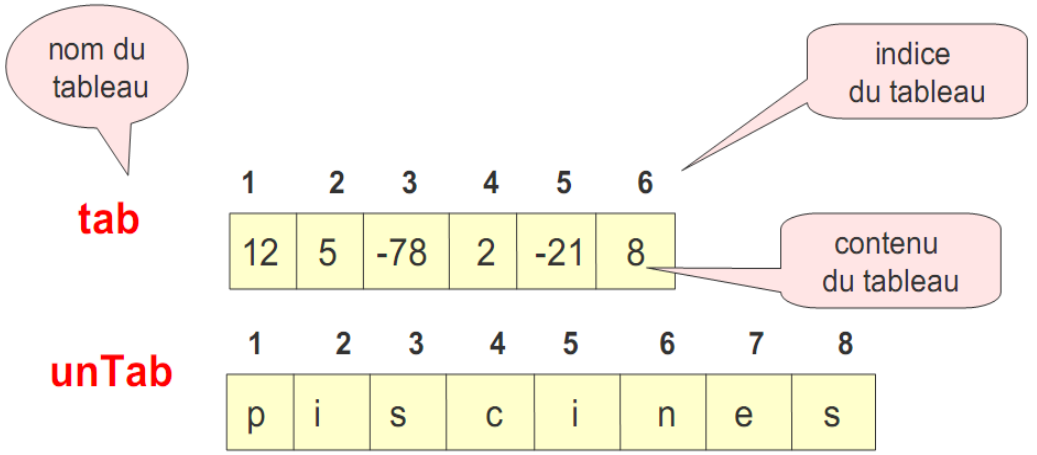


Tableau à une dimension : Initialisation

- La définition d'un tableau nécessite trois informations :
  - le **type** des éléments du tableau (rappelez-vous : un tableau est une suite de données de même type) ;
  - le **nom** du tableau (en d'autres mots, son identificateur) ;
  - la **longueur** du tableau (autrement dit, le nombre d'éléments qui le composent). Cette dernière doit être une expression entière

identificateur[longueur]: TYPE

tab[20]: ENTIER

TabP[10] : CHAINE

- **Initialisation :** Action de déclarer un tableau et de lui donner ses premières valeurs

Tab[3]: ENTIER ← { 12, 24, 73 };

Il est possible d'initialiser sans donner la taille du tableau

Tab[] ← { 1, 2, 3 };

Tableau à une dimension : Parcours

- **Plusieurs opérations peuvent être effectuer sur les tableaux 1D**
  - Enregistrement ou chargement
  - Mise à jour
  - Affichage
  - Statistiques : MIN, MAX, AVG, SOMME, MOYENNE …
  - Recherche de valeur
  - Ordonnancement ou Tri
- **Pour se faire il faut toujours parcourir le tableau**
- **La meilleure boucle est le POUR**

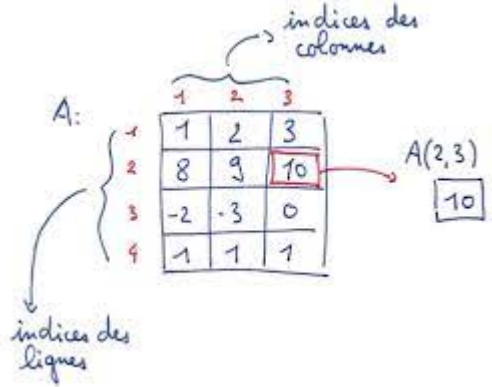
```
pour i ←1  à N faire
        Lire(T[i]) ;
Fpour
```

```
pour i ←1  à N faire
        Ecrire(T[i]) ;
Fpour
```

```
Algorithme affectation2;
Var
      T: tableau[7] de Entier;
      i : entier;
Début
/* lecture des éléments du tableau
Pour i ← 1 à 7  Faire
  Ecrire("entrer l'élément N° ", i);
  Lire(T[i]);
FinPour
Fin
```

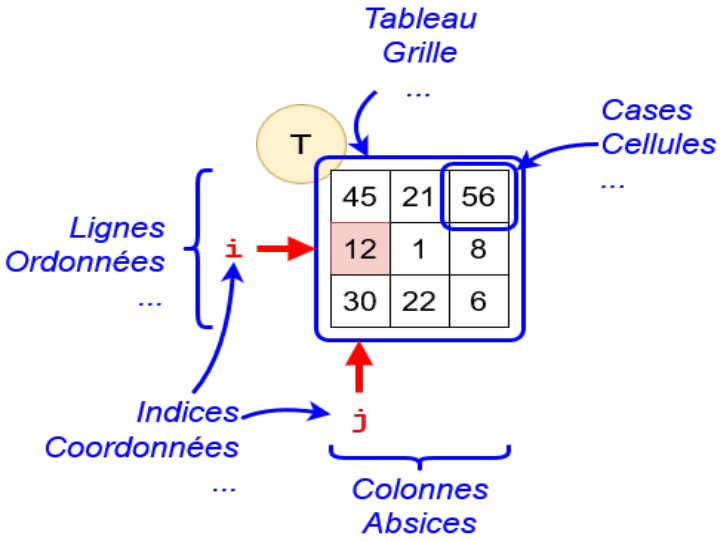
Tableau à 02 dimensions

- **Utiliser les deux indices i et j**
  - i pour les lignes 0 à L
  - j pour les colonnes 0 à C
- **Déclaration nécessite trois informations :**
  - Le type des éléments du tableau ;
  - le nom du tableau (son identificateur) ;
  - La taille des différentes dimensions (Nombre de lignes et Nombre de colonnes).



Type NomduTableau [Nb lignes] [Nb colonnes];

- **Exemples** **Tab[3][4] : Entiers**
  - On déclare un tableau d'entiers de 03 lignes et 04 colonnes



Pape Ousmane	DIOP	77 876 76 76	Dakar
Aliou	FAYE	76 555 44 22	Ziguinchor
Fatou	FALL	78 763 44 22	Touba
Jean Alain	DIATTA	33 875 55 11	THIES
Amie	Ka	77 765 77 11	Matam
Paul	SARR	76 776 32 44	Mbour

Tableau à 02 dimensions

- **Initialiser un tableau 2D**
  - int Tab[3][2] <-- {{ 51, 23 }, { 11, 32 }, { 83, 14 } };
  - Tab[0][0] <--81;
  - Tab[0][1] <--71;
  - Tab[0][1] <--12;
  - Tab[1][1] <-- Tab[2][1] ;
  - Tab[2][1] <--31;

51	23
11	32
83	**14**

81	12
11	14
83	**31**

  - Char tabE[3][2] <-- {{ 'Fatou', 'BA' }, { 'Amie', 'FAYE' }, { 'Jean", 'SARR' } };
  - tabE[0][1]= 'SARR'
  - tabE[2][0]= 'Ngor'

Fatou	BA
Amie	FAYE
Jean	SARR

Fatou	**SARR**
Amie	FAYE
Ngor	SARR

Tableau à 02 dimensions : Parcours

- **Pour parcourir un tableau 2D**
  - On utilise deux boucles FOR imbriquées
  - La première boucle i pour les lignes POUR i ← 0 à 3 par pas de 1
  - La seconde boucle j pour les colonnes POUR j← 0 à 2 par pas de 1
  - Principe: On se position à i<--0 puis on parcourt tous les j (0 à C) puis on passe à i<--1 ainsi de suite jusqu'à i<L

```
VARIABLES
    Tab[3][2] ← {{ 51, 23 }, { 11, 32 }, { 83, 14 } }
    i, j:Entiers
DEBUT
    i←0, j←0;
    POUR i ← 0 à 3 par pas de 1
        POUR j← 0 à 2 par pas de 1
            AFFICHER (tab[i][j]);
        FIN POUR
    FIN POUR
FIN
```

51	23
11	32
83	14

Tri d'un tableau

- Comparer un élément au suivant de la liste si cet élément à une valeur inférieur, ilreste en place sinon il est échangé avec celui qui est de valeur inférieur.
- Soit le tableau suivant a classé dans l'ordre croissant:

Processus de classement	5	12	2	31	7
i=1	2	12	5	31	7
i=2	2	5	12	31	7
i=3	2	5	7	31	12
i=4	2	5	7	12	31
i=5	2	5	7	12	31

```
Pour i←1 à n Faire
      Lire(tab[i]);
Fpour;
Pour i←1  à n-1 Faire
     Pour  j←i+1 jusqu'a n Faire
               Si (tab[j]<tab[i]) Alors
                         temp ← tab[i]
                         tab[i] ←tab[j]
                         tab[j] ←temp
               Finsi;
      Fpour
Fpour;
Pour i ←1 à  n Faire
        Ecrire(tab[i]);
Fin pour;
Fin.
```

Chapitre 4
Les sous programmes

Les sous programmes

- **Problème** : Lorsqu'un programme est long on ne peut le coder enentier dans la fonction principale.
- **Idée** : Décomposer le programme en plusieurs sous-programmes plus petits.
  - Les sous-programmes (procédures ou fonctions) permettent de découper un gros programme en morceaux plus petits.
- Un sous-programme possède un nom, des variables, des instructions, un début et une Fin.
- L'exécution d'un sous-programme est invoquée dans la fonction principale.
- Un sous-programme est un élément de programme nommé et éventuellement paramétré, que l'on définit afin de pouvoir ensuite l'appeler par son nom en affectant, s'il y a lieu, des valeurs aux paramètres
- **Les intérêts :**
  - Le gain de place en mémoire pour le code du programme : lorsqu'un sous-programme est appelé plusieurs fois, dans une boucle,
  - Les notions d'abstraction et de modularité : utiliser un appel de sous-programme permet de d'écrire une application en faisant abstraction (en dissimulant) les détails de la fonction que réalise ces ous-programme.
- **Il existe deux types de sous-programmes :**
  - Les procédures : sous-programme nommé, ne renvoyant pas de résultat.
  - Les fonctions : sous-programme nommé, renvoyant une valeur.

Les sous programmes : Les procédures

- **Définition**
  - Une procédure a la même structure qu'un programme. Après le nom de la procédure, il faut donner la liste des paramètres (s'il y en a) avec leur type respectif. Ces paramètres sont appelés paramètres formels. Leur valeur n'est pas connue lors de la création de la procédure.
- **Syntaxe**

```
Procédure nom_proc(liste et déclaration de paramètres);
Variables identificateurs : type ;

 Début
   Instruction(s) ;

 FIN
```

- **Appel**
  - L'appel de procédure s'écrit en mettant le nom de la procédure, puis la liste des paramètres, séparés par des virgules.
  - A l'appel d'une procédure, le programme interrompt son déroulement normal, exécute les instructions de la procédure, puis retourne au programme appelant et exécute l'instruction suivante.

```
Nom_proc (liste de paramètres) ;
```

Les sous programmes : Les fonctions

- **Définition**
  - Les Fonctions sont des sous algorithmes admettant des paramètres et retournant un seul résultat(une seule valeur) de type simple qui peut apparaître dans une expression, dans une comparaison, à ladroite d'une affectation, etc.
- **Déclaration**

```
Fonction nom-de-fonction (Déclaration Des Paramètres):Type du résultat;
```

- **Syntaxe**

```
//*Déclaration de la fonction Max
Fonction Max(X: réel, Y:réel) : réel ;
Début
    Si X > Y Alors
        ecrire(X)
    Sinon
        ecrire(Y)
    FinSi
Fin
```

- **Appel**

```
NomDeFonction ← Valeur Du Resultat;
```

Les sous programmes : Les variables

- La portée d'une variable désigne le domaine de visibilité de cette variable.
- Une variable peut être déclarée dans deux emplacements distincts.
  - Une variable déclarée dans la partie déclaration de l'algorithme principale est appelée variable **globale** :
    - Accessible de n'importe où dans l'algorithme, même depuis les procédures et les fonctions.
    - Elle existe pendant toute la durée de vie du programme.
  - Une variable déclarée à l'intérieur d'une procédure (ou une fonction) est dite **locale**.
    - N'est accessible qu'à la procédure au sein de laquelle elle définie, les autres procédures n'y ont pas accès.
    - La durée de vie d'une variable locale est limitée à la durée d'exécution de sa procédure ou sa fonction.
- La variable locale s'oppose à la variable globale qui peut être utilisée dans tout le programme.

```
Algorithme portée
 Variable x,y :entier ;
Procedure p1() :
  Variables
    A :entier
Debut
….
Finproc
//Algorithme principal
Début
  …..
Fin
```

```
X et Y sont des variables
globales  visibles dans tout
l'algorithme


A est une variable locale
visible uniquement à
l'intérieur de la
procédure
```

Les sous programmes : Les passages

- **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.

```
AlGORITHME Passage_par_valeur ;
Variables N : entier
Procédure P1(A : entier) //*Déclaration de la procédure P1
Début
   A ← A * 2 ;
   écrire(A) ;
FinProc

Début                              //*Algorithme principal
   N ← 5 ;
   P1(N) ;
   écrire(N) ;
Fin
```

Les sous programmes : Les passages

- **Passage par référence**
  - la procédure ou fonction 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 sera donc modifiée.
  - Les paramètres passés par adresse sont précédés du mot clé Var.

```
ALGORITHME Passage_par_référence ;
Variables N : entier
//*Déclaration de la procédure P1
Procédure P1 (Var A : entier)
Début
    A ⟵ A * 2
    écrire(A)
FinProc
//*Algorithme Principal
Début
    N ⟵ 5
    P1(N)
    écrire(N)
Fin
```

Les sous programmes : Les fonctions

- **Exemple de fonction**

```
Algorithme Appel_fonction_Max
    Variables A, B, M : réel
    // * Déclaration de la fonction Max
    Fonction Max(X: réel, Y: réel) : réel
    DEBUT
        Si X > Y Alors
            max(x,y) ← X
        Sinon
            max(x,y)  ←  Y
        FinSi
    FIN

   //*Algorithme principal
    DEBUT
     Ecrire ("Donnez la valeur de A :")
     lire(A)
    Ecrire ("Donnez la valeur de B :")
     lire (B)

     //*Appel de la fonction Max
       M← Max(A,B)

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

   FIN
```

Logiciel Recommandé pour les TD : AlgoBox

https://www.xm1math.net/algobox/download.html