I. Introduction :

Les structures de contrôle sont le "cerveau" d'un programme. Sans elles, un ordinateur lirait simplement une liste d'instructions de haut en bas, sans jamais s'adapter. Elles permettent de rompre cette lecture linéaire pour introduire de l'intelligence : **décider**, **choisir** ou **répéter**.

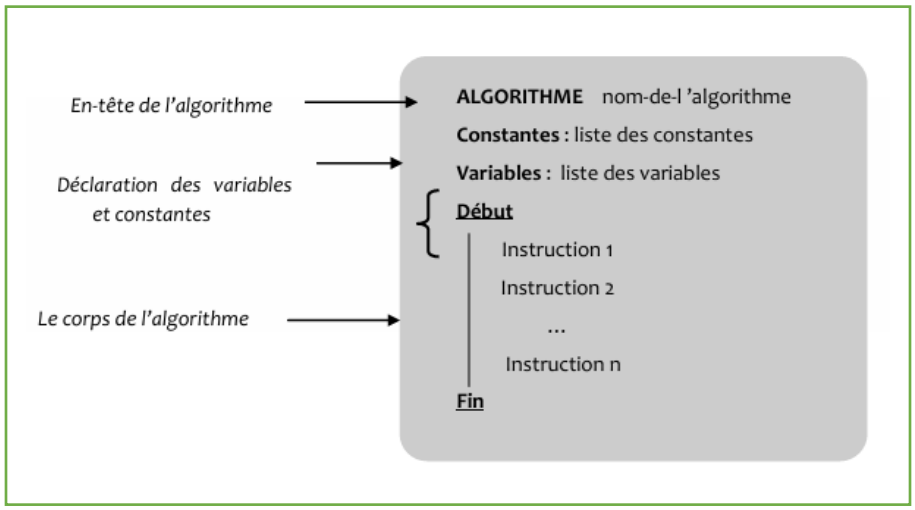

Figure 01 : La structure générale d'un algorithme

L'en-tête : permet tout simplement d'identifier l'algorithme.

Déclaration : liste de toutes les constantes et variables utilisées dans l'algorithme.

Le corps : cette partie contient les ordres (instructions) ou les tâches de l'algorithme.

Voici les trois grandes familles qui composent tout algorithme :

1. La Structure Séquentielle (L'ordre par défaut)

La **structure séquentielle** est la forme la plus simple et la plus fondamentale de l'algorithmique. C'est le mode d'exécution "naturel" d'un ordinateur.

1.1. Définition

Dans une structure séquentielle, les instructions sont exécutées l'une après l'autre, dans l'ordre exact où elles sont écrites, de haut en bas. Chaque instruction doit être terminée avant que la suivante ne commence. Il n'y a ni saut, ni retour en arrière, ni choix.

1.2. Fonctionnement technique

Elle repose sur trois étapes classiques de l'informatique :

- **Entrée des données :** On récupère les informations nécessaires (via le clavier ou une variable).
- **Traitement :** On effectue des calculs ou des transformations sur ces données.

- **Sortie des données :** On affiche ou on enregistre le résultat.

1.3. Exemple détaillé : Calcul de la surface d'un rectangle

Imaginons que nous voulons créer un petit programme qui demande la largeur et la longueur d'un rectangle pour en calculer la surface.

Version Algorithme (Pseudo-code) :

```
Algorithme CalculSurface
Variables
      longueur, largeur, surface : Réel
Début
      Afficher "Entrez la longueur :"   // Étape 1 : Entrée
      Saisir longueur
     Afficher "Entrez la largeur :"    // Étape 1 : Entrée
     Saisir largeur
     surface <- longueur * largeur     // Étape 2 : Traitement (Calcul)
     Afficher "La surface est : ", surface // Étape 3 : Sortie
 Fin
```

Version Programmation (Exemple en Python):

En Python, la séquence est respectée ligne par ligne :

```
# Étape 1 : Acquisition des données
  longueur = float(input("Entrez la longueur : "))
   largeur = float(input("Entrez la largeur : "))
# Étape 2 : Traitement mathématique
   surface = longueur * largeur
# Étape 3 : Affichage du résultat
   print("La surface totale est de :", surface)
```

1.4. Pourquoi est-ce important ?

Même dans les programmes les plus complexes (intelligence artificielle, jeux vidéo), la base reste séquentielle. Si vous inversez deux lignes (par exemple, essayer de calculer la surface *avant* d'avoir demandé la longueur), le programme plantera ou donnera un résultat faux (erreur de logique).

2. Les Structures de Décision (Conditionnelles)

Les **structures de décision** (ou conditionnelles) permettent à un programme de ne plus suivre un chemin unique, mais de choisir entre plusieurs voies en fonction de critères précis. C'est ici que

l'algorithme commence à "réfléchir" selon les circonstances (tests logiques).

2.1. Le concept : Le Test Logique

Toute décision repose sur une **condition** qui est soit **Vraie**, soit **Fausse** (type booléen). On utilise des opérateurs de comparaison :

- = = (égal), != (différent)
- < , > (inférieur, supérieur)
- <= , >= (inférieur ou égal, supérieur ou égal)

2.2. La Forme Simple : SI ... ALORS

On exécute un bloc de code **uniquement** si la condition est remplie. Si elle est fausse, on ne fait rien.

C'est-à-dire On vérifie une condition (Vrai ou Faux).

- **Exemple concret :** Un distributeur de billets.

  - **Condition :** Le solde est-il suffisant ?
  - **Si Vrai :** Donner l'argent.
  - **Sinon :** Afficher "Fonds insuffisants".

- **Exemple : Alerte température**

```
Algorithme :
  SI (température > 38) ALORS
    Afficher "Vous avez de la fièvre"
  FIN SI
```

- **Programmation (Python) :**

```
if temperature > 38:
    print("Vous avez de la fièvre")
```

2.3. La Forme Alternative : SI ... ALORS ... SINON

On prévoit deux chemins : un pour le cas où c'est vrai, un autre pour le cas où c'est faux.

Exemple : Accès à un site web

```
Algorithme :
  SI (age >= 18) ALORS
     Afficher "Accès autorisé"
  SINON
     Afficher "Accès refusé"
  FIN SI
```

- **Programmation (C) :**

```
if (age >= 18) {
   printf("Accès autorisé");
} else {
   printf("Accès refusé");
}
```

2.4. La Forme Imbriquée : SI ... SINON SI ... SINON

Utilisée lorsqu'il y a plus de deux possibilités.

Exemple : Mention à un examen

```
Algorithme :
SI (note >= 16) ALORS
   Afficher "Mention Très Bien"
SINON SI (note >= 12) ALORS
   Afficher "Mention Assez Bien"
SINON
   Afficher "Passable ou Échec"
FIN SI
```

Programme python :

```
python
# Saisie de la note (convertie en nombre décimal)
note = float(input("Entrez votre note : "))

# Structure de décision imbriquée
if note >= 16:
   print("Mention Très Bien")
elseif note >= 12:
   print("Mention Assez Bien")
else:
   print("Passable ou Échec")
```

2.5. La Structure de Choix Multiple : SELON/ Cas (ou Switch) (Le choix multiple) :

C'est une alternative plus lisible au "Si" imbriqué quand on teste plusieurs valeurs fixes d'une même variable. Utile quand il y a beaucoup de possibilités pour une même variable.

- **Exemple : Un menu de téléphone.**

```
Cas 1 : Consulter le solde.
Cas 2 : Recharger le compte.
Cas 3 : Parler à un conseiller.
```

- **Exemple : Choix d'un menu**

```
SELON (jour)
   1 : Afficher "Lundi"
   2 : Afficher "Mardi"
   ...
   SINON : Afficher "Jour inconnu"
FIN SELON
```

```
Algorithme :
   SELON (touche_pressée)
      1 : Afficher "Lancer la partie"
      2 : Afficher "Options"
      3 : Afficher "Quitter"
      SINON : Afficher "Touche invalide"
   FIN SELON
```

- **Programmation (JavaScript) :**

```
switch (touche) {
   case 1: console.log("Lancer"); break;
   case 2: console.log("Options"); break;
   default: console.log("Invalide");
}
```

Pourquoi les utiliser ?

Sans elles, un logiciel serait incapable de gérer les erreurs (ex: "Mot de passe incorrect") ou de s'adapter au profil de l'utilisateur. Elles apportent la **flexibilité**.

3. Les Structures de Répétition (Boucles)

Elles permettent d'automatiser des tâches répétitives sans réécrire le code.

A. La boucle "Pour" (Itération définie)

On l'utilise quand on sait **exactement** combien de fois on doit répéter l'action.

- **Exemple :** Envoyer un email de vœux à 50 employés.
    - On initialise un compteur à 1.
    - On envoie l'email.
    - On s'arrête une fois arrivé à 50.

B. La boucle "Tant que" (Itération indéfinie)

On l'utilise quand la répétition dépend d'un événement futur dont on ne connaît pas le moment exact.

- **Exemple :** Saisir un mot de passe.
    - **Tant que** le mot de passe est incorrect :
        - Afficher "Erreur, recommencez".

- (Dès qu'il est juste, la boucle s'arrête).

Pourquoi est-ce crucial ?

Imaginez un logiciel de navigation GPS. Sans structure de contrôle, il ne pourrait pas recalculer d'itinéraire (**Si** vous tournez à gauche au lieu de droite) ou mettre à jour votre position en continu (**Tant que** vous n'êtes pas arrivé).

Souhaitez-vous un **exercice pratique** pour tester votre compréhension de ces concepts ?