LA GRANDE ÉCOLE
DES TÉLÉCOMMUNICATIONS
ET DES TECHNOLOGIES
DE L'INFORMATION

**Filière Smart-ICT**

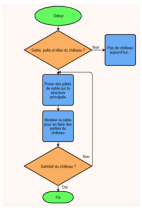
# Algorithmique et Programmation C

**Mr N.EL FADDOULI**
*elfaddouli@emi.ac.ma*
*nfaddouli@gmail.com*

**Année Universitaire:2024/2025**

1

# Algorithmique



4

## Définitions: L'informatique

- L'informatique est une science de traitement automatique d'informations par ordinateur.
- Ce traitement est fait à travers des logiciels spécifiques installés et exécutés sur ordinateur.
- Nous constatons donc que dans l'informatique on a besoin d'ordinateur qui est une machine électronique et aussi de logiciels développés pour résoudre des problèmes spécifiques.

Informatique = Science + Technologie

OU

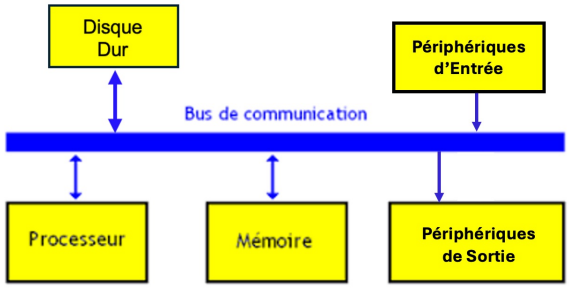
Informatique = Logiciels + Ordinateur

- Les informations (données) traitées sont de différents types: texte, audio, son, image, vidéo



## Définitions: L'ordinateur

- L'ordinateur est une machine constituée de plusieurs composant électronique dont chacun jour un rôle spécifique dans le traitement de l'information.
- Tous ces composants sont reliés entre eux via une carte mère qui permet l'échange de données et de commandes à travers des bus de communication.
- Les principaux composants et échanges entre eux sont indiqués dans la figure suivante:



## Définitions: L'ordinateur

- Le disque dur (HD): C'est une mémoire **permanente** dans laquelle sont stockés le système d'exploitation, les programmes (logiciels) installés et les données utilisateur (documents texte, images, vidéos, …).
- La RAM, appelée aussi mémoire centrale, est une mémoire **volatile** contenant les programmes en cours d'exécution, les entrées (valeurs saisie par l'utilisateur) et les valeurs calculées.
- Le processeur (CPU) qui exécute les instructions d'un programmes une à une.
- Les périphériques d'entrée permettent à l'utilisateur d'entrer les données à traiter (entrées), par exemple: clavier, micro, caméra, …etc.
- Les périphériques de sortie permettent de communiquer des données (sorties) à l'utilisateur, par exemple: écran, imprimante, haut-parleur, …etc.



## Définitions: L'ordinateur

Les principaux échanges entre les composants d'un ordinateur sont:

- HD → RAM: les programmes lancés par l'utilisateur ou les fichiers de données à traiter sont chargés dans la RAM depuis le disque dur (HD).
- RAM→HD: le contenu de la RAM peut être sauvegardé dans des fichiers sur disque
- RAM → CPU: le processeur (CPU) récupère les instructions à exécuter depuis la RAM. Lorsuq'il s'agit de faire des calculs, le CPU récupère, depuis la RAM, les valeurs à utiliser.
- CPU → RAM: Les résultats des calculs effectués par le CPU sont stockés dans la RAM
- Périphériques d'Entrée → RAM: les données entrées (ou saisies) par l'utilisateur sont stockées dans la RAM
- RAM → Périphériques de Sortie: Les résultats de calcul sont envoyée

## Définitions: Programme-Logiciel

- Un programme est une suite d'instructions pour traiter des entrées et produire (calculer) des sorties.
- Un programme accomplit une tâche spécifique

  Exemple: Programme de résolution d'une équation de 2°
- Un logiciel est constitué généralement de plusieurs programmes intégrés et dont chacun fait un traitement spécifique.
- Un logiciel est souvent conçu pour réaliser une gamme de tâches ou pour offrir une solution complète à un problème.

  Exemple: Microsoft Word est un logiciel qui inclut des programmes pour l'édition de texte, la vérification orthographique, la gestion des fichiers, etc





## Définitions: Comment un programme est stocké dans la RAM ?

- La RAM est constituée de zones mémoire dont chacune contient une donnée (valeur) ou une instruction.
- La taille de chaque zone mémoire dépend de la taille de son contenu (donnée ou instruction)
- Les utilités de mesure de la RAM sont:

1To = 1024 Go

1Go= 1024 Mo

1Mo= 1024 Ko

1Ko= 1024 Octets

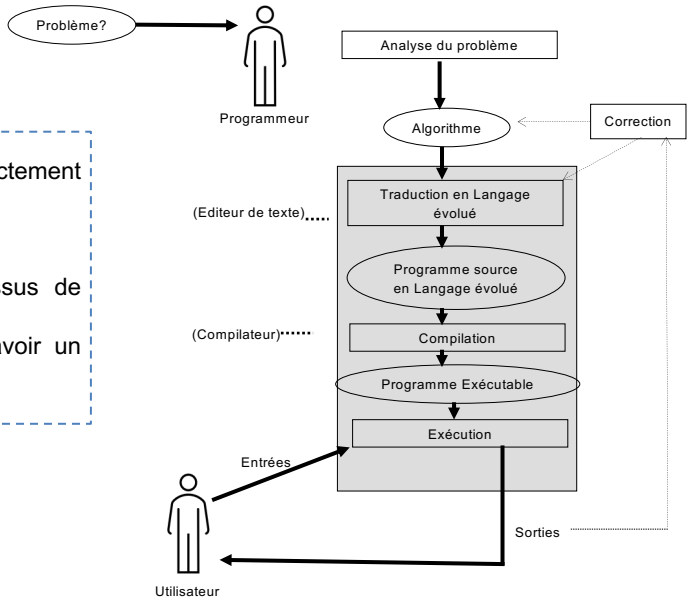
1Octet= 8 bits

1 bit = 0 ou 1

- On peut conclure que tout ce qui est dans la RAM est représenté en **langage binaire (ou langage machine)** sous forme de suite de 0 et 1
- Le processeur qui exécute les instructions des programmes ne comprend donc que le langage binaire.

## Etapes de développement d'un programme

- On ne peut pas écrire directement un programme en binaire.
- On doit suivre un processus de plusieurs étapes jusqu'à avoir un programme en binaire.








## Etapes de développement d'un programme

- Tout commence par la spécification et la description du problème à résoudre par ordinateur via un programme qu'on doit développer.

  Exemple:

  On veut un programme qui fait la résolution d'une équation de deuxième degré $AX^2+BX+C=0$ en faisant l'hypothèse que A est différent de 0.

- Le programmeur (ou analyste programmeur) **analyse** le problème en essayant de répondre à ces trois questions:

1. Quelles sont les données à traiter ? (Entrées)
2. Quels sont les résultats à calculer ou déterminer ? (Sorties)
3. Quelles sont les étapes à suivre pour obtenir les sorties à partir des entrées ? (Traitement)

- Les réponses à ces trois questions permettent d'avoir un **algorithme** écrit en langage naturel.

## Etapes de développement d'un programme

- Exemple d'analyse du problème d'équation de deuxième degré

**Entrées**: A, B et C que l'utilisateur doit saisir

**Sorties**: X (solution unique), $X_1$ et $X_2$ ou un message d'erreur (pas de solution dans IR)

**Traitement**:

1. L'utilisateur entre au clavier les valeurs de A, B et S
2. On (**ordinateur**) calcule le déterminant Δ qui vaut $B^2 - 4AC$
3. Si Δ est égal à 0:
    4. On calcule la solution unique X qui vaut –B/2A
    5. On affiche la valeur de X sur écran
6. Si Δ est supérieur à 0:
    7. On calcule deux solutions $X_1$ et $X_2$ avec les deux formules respectives $(-B - \sqrt{\Delta})/2A$ et $(-B + \sqrt{\Delta})/2A$
    8. On affiche les valeurs de $X_1$ et $X_2$ sur écran
9. Si Δ est inférieur à 0:
    10. On affiche sur écran le message « Pas de solution dans IR »



## Etapes de développement d'un programme

- L'algorithme écrit en langage naturel est ensuite traduit en un langage de programmation **évolué** comme le langage C.
- Chaque langage de programmation évolué possède une syntaxe formelle particulière qu'on doit respecter.
- Après traduction de l'algorithme, on obtient sur ordinateur un programme **source** en un langage de programmation.
- On utilise un compilateur pour vérifier la syntaxe du programme source et le traduire ensuite en langage binaire.
- Le compilateur est un programme qui permet de détecter les erreurs de syntaxe du programme source et le traduire, lorsqu'il est correct, en langage binaire pour avoir un programme exécutable

## Etapes de développement d'un programme

- Le programme exécutable obtenu doit être testé sur plusieurs jeux de données pour vérifier s'il fournit les bons résultats (Sorties).
- Si les sorties sont incorrectes, on doit corriger l'algorithme, refaire la traduction et recompiler le nouveau programme source obtenu. Ce cycle peut être répété plusieurs fois jusqu'à être sûr que le programme donne les bons résultats dans tous les cas possibles.



## L'ALGORITHMIQUE: Concepts fondamentaux

☞ **Action (Ordre ou instruction):**

- C'est une opération de base que l'ordinateur sait faire.
- Elle produit un **effet** spécifique en un **temps fini**.

Par exemple, l'effet de l'instruction « Afficher la valeur de X sur écran » sera la modification du contenu de l'écran sur lequel un nouveau message (valeur de X) apparaitra. On a donc un effet visuel.

☞ **Algorithme:**

- C'est une suite **finie** d'instructions dans un ordre déterminé.
- Il est appliqué à un nombre fini de données (entrées) pour produire des résultats (sorties)
- Il est indépendant du type d'ordinateur et des langages de programmation évolués.

## L'ALGORITHMIQUE: Concepts fondamentaux

☞ *Algorithme de la vie quotidienne:*

- Dans la vie quotidienne, on applique plusieurs algorithmes pour effectuer des tâches particulières comme une recette de cuisine ou un guide d'appareil électrique.
- Ces algorithmes ont ces caractéristiques:
  - Ils sont exécutés par l'homme.
  - Ils peuvent être non détaillés ou exprimés à demi-mot.
  - Ils peuvent contenir des implicites.

☞ *Algorithme pour ordinateur:*

- Les algorithmes destinés à l'ordinateur doivent être **précis** et **compréhensible**
- On doit bien préciser les **Entrées**, les **Sorties** et les **Instructions** (*traitement*)



## L'ALGORITHMIQUE: Concepts fondamentaux

☞ **Exemple d'algorithme**:

Calcul des montants hors taxe et avec taxe comprise d'une facture d'électricité

**Entrées**: *consommation*, *prix_unitaire*, *taux_taxe*.

**Sorties** (Le résultat final à afficher ): le prix hors taxe *PHT* et le prix avec taxe *PT*.

**Traitement:**

1. L'utilisateur entre au clavier les valeurs de *consommation*, *prix_unitaire* et *taux_taxe*
2. On calcule la valeur de *PHT* avec la formule (*consommation * prix_unitaire*)
3. On calcule la valeur de *PT* avec la formule *PHT + PHT*taux_taxe*
4. On affiche un message contenant les valeurs de *PHT* et *PT*

*Cas de figure:* Consommation=100 Kw, Prix unitaire=0.50 DH/Kw, taux de la taxe= 20%

L'ordinateur attendra la saisie des entrées (100, 0.50, 0.2) par l'utilisateur, fait les calculs de PHT = (100*0.5)=50 et PT = 50*(1+0.2)=60 et affiche les deux valeurs 50 et 60

## L'ALGORITHMIQUE: Pseudo-code (1/2)

☞ Un algorithme peut être écrit en **langage naturel** afin de préciser les instructions que l'ordinateur doit exécuter dans l'ordre pour avoir les résultats souhaités.

☞ Ces instructions contiennent des verbes d'actions comme calculer, afficher …etc.

☞ Ces instructions permettent de guider l'ordinateur afin de résoudre un problème donné.

☞ Exemple: Résolution d'une équation de 2°

1. L'utilisateur saisit les valeurs des coefficients A, B et C de l'équation
2. On calcule le déterminant Δ dont la formule est $B^2 - 4AC$
3. Si on a Δ =0 alors on calcule la solution unique $X_0$ dont la formule est -B/2A et on l'affiche.

…..

…..



## L'ALGORITHMIQUE: Pseudo-code (2/2)

☞ On peut également **simplifier** les phrases exprimant des instructions afin de ne garder que l'essentiel.

Exemple: L'instruction « ***Calculer*** Δ ***dont la formule est*** $B^2 - 4AC$ » peut être remplacée par « $\Delta \leftarrow B^2 - 4AC$ » qui veut dire que la valeur de Δ est obtenue par le calcul de $B^2 - 4AC$

☞ On utilisera ainsi un **pseudo-code** pour écrire un algorithme.

☞ Un **pseudo code** est un langage:

- **informel** : il n'a pas une syntaxe particulière ou des règles strict à respecter
- **proche du langage naturel**
- **indépendant de tout langage de programmation**: l'algorithme en pseudo-code peut être traduit ensuite en un programme en n'importe quel langage de programmation évolué.

## L'ALGORITHMIQUE: Fonctions de base d'un ordinateur (1/2)

☞ Les instructions d'un algorithme sont destinées pour l'ordinateur. Par conséquent chacune d'elles doit faire partie de ses fonctions de base.

☞ Les fonctions de base d'un ordinateur sont les ordres qu'un ordinateur est capable d'exécuter.

☞ Ces fonctions de base sont:

1. **L'affichage**: L'ordinateur est capable d'afficher des messages et des valeurs sur écran.
2. **La lecture**: L'ordinateur peut prendre les valeurs saisies par l'utilisateur et les stocker dans la mémoire centrale (RAM).
3. **Les calculs**: L'ordinateur peut faire des calculs:
   - Arithmétiques sur des nombres (+, -, *, /)
   - Logiques en utilisant les opérateurs de comparaison(=, ≤, ≥, <, >, ≠) et les opérateurs logiques (ET, OU, NON)



## L'ALGORITHMIQUE: Fonctions de base d'un ordinateur (1/2)

4. **La mémorisation**: L'ordinateur mémorise dans la RAM les entrées et les valeurs calculées.
5. **Le séquencement**: L'ordinateur exécute les instructions dans l'ordre depuis la première jusqu'à la dernière.
6. **La sélection**: On peut demander à l'ordinateur de choisir un seul traitement (suite d'instructions) à exécuter parmi plusieurs traitements possibles (un ou plus). Le choix sera effectué selon la vérification d'une condition bien déterminée.

   Exemple: Dans le problème de résolution d'une équation de second degré, on a trois traitements possibles: *calculer et afficher une seule solution*, *calculer et afficher deux solutions* ou *afficher un message d'erreur(pas de solution dans IR)*.

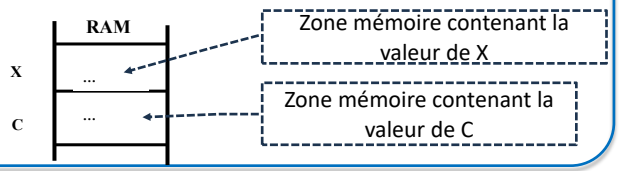
   Chaque traitement est possible lorsqu'une condition bien déterminée est satisfaite *(Δ=0, Δ>0, Δ<0)*
7. **La répétition**: L'ordinateur est capable de répéter un traitement plusieurs fois

➥ **On peut demander à l'ordinateur d'exécuter toute instruction qui fait partie de ces 7 fonction de base**

## L'ALGORITHMIQUE: Les variables

- Une **variable** est un symbole qui désigne une donnée pouvant être une entrée, une sortie ou un résultat intermédiaire.
- Une variable est un conteneur dont le contenu (*valeur*) peut être de différents types (entier, réel, texte, …)
- Une variable est caractérisée par:
  - Nom qui identifie la variable dans l'algorithme.
  - Valeur: c'est le contenu de la variable qui peut être modifié à tout moment.
  - Type: c'est la nature de la valeur de la variable (entier, caractère, …).
  - Zone mémoire qui contient la valeur de la variable dans la RAM
- Toute variable doit être **déclarée** dans l'algorithme selon la syntaxe qu'on adoptera dans notre pseudo-code: **Type** *variable1*, *variable2*, …

*Exemple*: Entier X
Caractère C






## L'ALGORITHMIQUE: Structure d'un algorithme

- Un algorithme est composé de deux parties:
  1. La déclaration des variables utilisées pour manipuler les entrées et calculer les sorties.
  2. Les instructions permettant de manipuler des variables et indiquer les étapes à suivre pour avoir les résultats finaux.
- Ces deux parties peuvent être structurées comme suit:

```
Déclaration de variables
Début
Instructions
Fin
```

## L'ALGORITHMIQUE: L'affectation

- L'instruction d'**affectation** consiste à **évaluer** *(calculer)* une **expression** *(formule)* et stocker le résultat dans une variable.
- L'ancienne valeur de la variable sera écrasée après l'affectation
- On adoptera la syntaxe suivante pour indiquer une affectation:

*Variable* ← *Expression*

Calculer la valeur de **Expression** et stocker le résultat dans **Variable**

- On lit cette instruction « *Variable* **reçoit** *Expression* »
- L'expression peut être une ***constante***, une ***variable*** ou une ***formule*** de calcul
- *Exemple:*

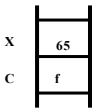
```
Entier X
Caractère C
Début
   X ← 10
   X ← (X*2) + 45
   C ← 'f'
Fin
```

X reçoit 10 ⇨ X=10

X reçoit (X*2) + 45 ⇨ X= 2*10+45 ⇨ X=65

L'ancienne valeur 10 de X sera remplacée par 65

C reçoit la lettre f ⇨ C='f'






## L'ALGORITHMIQUE: L'affectation

- L'expression dans une instruction d'affectation peut contenir les opérateurs arithmétiques suivants:

| Opérateur | Description | Exemple avec A= 17 B=5 |
|---|---|---|
| + | Calcul de la somme de deux nombres | C ← A+3 ⇨ C=20 |
| - | Calcul de la différence de deux nombres | C ← A-10 ⇨ C=7 |
| * | Calcul du produit de deux nombres | C ← B*3 ⇨ C=15 |
| / | Calcul du résultat de la division réelle d'un nombre par un autre | C ← A/B ⇨ C=3.4 |
| Div | Calcul du quotient de la division euclidienne d'un entier par un autre | C ← A Div B ⇨ C= 3 |
| Mod | Calcul du reste de la division euclidienne d'un entier par un autre | C ← A Mod B ⇨ C=2 |

## L'ALGORITHMIQUE: Exercices

Exercice 1:

Identifier les erreurs de syntaxes dans ces instructions:

```
Entier A , B
A ← 55
B ← A + F
B+A ← 32
B → 40
```

Exercice 2:

Donner les valeurs de A, B et C après l'exécution des 4 instructions d'affectation:

```
Entier A , B, B
A ← 55
B ← A*2
C ← B Div 3
A ← (A + B) Mod C
```



27

## L'ALGORITHMIQUE: La lecture des entrées

☞ **La lecture**: C'est une instruction qui consiste à demander à l'ordinateur d'**attendre** la **saisie** d'une ou plusieurs **valeurs** par l'**utilisateur** et les **stocker** ensuite dans des **variables** spécifiques après validation de la saisie avec la touche Entrée.

La syntaxe en pseudo-code de cette instruction est la suivante:

**Lire (*Variable1, Variable2, …*)**

*Exemple:*

```
Entier a, b, c
Début
    Lire (a)
    Lire (a, b, c)
```

L'utilisateur doit saisir un entier qui sera ensuite stocké dans la variable **a**

L'utilisateur doit saisir trois entiers qui seront ensuite stockés respectivement dans les variables **a**, **b** et **c.**

☞ Il est recommandé d'afficher un message à l'utilisateur avant l'instruction Lire pour lui indiquer quoi saisir.



28

## L'ALGORITHMIQUE: L'affichage des résultats

☞ **L'affichage**: C'est une instruction qui consiste à demander à l'ordinateur d'**afficher** un simple **message**, la valeur d'une **variable**, le résultat d'une **expression** ou une combinaison des trois.

La syntaxe en pseudo-code de cette instruction est la suivante:

**Ecrire (*"Message", variable, expression, …*)**

*Exemple:*

```
Entier  a, b, c
Début
        Ecrire ("Donner deux entiers :")
        Lire (a, b)
        c ← a + b
        Ecrire ("Le résultat est:", c)
        Ecrire (a + b)
        Ecrire ("La somme de ",a, " et ", b ,"=",  a + b)
Fin
```



## L'ALGORITHMIQUE: Exercices

1- Écrire un algorithme qui calcule et affiche la somme et le produit de deux entiers saisis au clavier.

3- Écrire un algorithme qui lit un entier N composé de 4 chiffres, calcule et affiche l'entier M composé des chiffres de N à l'envers

**Exemple:** N = 2579 ⇨ M = 9752

**Rappel:** Pour avoir le reste de la division euclidienne, on utilise l'opérateur **Mod**: Soient X et Y deux entiers:

X Mod Y ⇔ Le reste de la division de X par Y

## L'ALGORITHMIQUE: La sélection binaire simple (1/3)

- La **sélection binaire** permet de demander à l'ordinateur de **choisir un seul bloc** d'instructions à exécuter parmi **deux blocs possibles**.
- Le choix est effectué selon une **condition** bien déterminée
- La syntaxe en pseudo-code de cette instruction est la suivante:

```
SI   Condition   ALORS
Bloc1 ------  instruction1
              ...
SINON
Bloc2 ------  instruction1'
              ...
FinSi
```

- Le *Bloc1* sera exécuté si la *Condition* est vérifiée (vraie)
- Le *Bloc2* sera exécutée si la *Condition* est fausse.

*Important: L'indentation est utilisée pour une meilleure lisibilité de l'instruction*



## L'ALGORITHMIQUE: La sélection binaire simple (2/3)

- *Exemple:* Afficher le maximum de A et B saisis au clavier

```
Entier A, B
Début
Ecrire ("Entrez deux entiers: ")
Lire (A, B)
Si A>B Alors
        Ecrire ("Le maximum est: ",A)
Sinon
        Ecrire ("Le maximum est: ",B)
FinSi
Fin
```

```
Si A>B Alors
        C ← A
Sinon
        C ← B
FinSi
Ecrire ("Le maximum est: ",C )
```

Exercice: Calcul de la taxe sur le chiffre d'affaires CA saisi au clavier sachant que le taux est de 10% si CA<5000Dh et 20% si CA≥5000Dh

## L'ALGORITHMIQUE: La sélection binaire simple (1/3)

- La **sélection binaire réduite** permet de demander à l'ordinateur de n'exécuter **un bloc** d'instructions que lorsqu'une condition est satisfaite. Dans le cas contraire, on doit poursuivre l'exécution de l'algorithme.
- La syntaxe en pseudo-code de cette instruction est la suivante:

```
SI  Condition  ALORS
        instruction1
        …
FinSi
```

*Bloc d'instructions*

- Exemple: Calcul du prix total ***PT*** de ***N*** articles de prix unitaire est ***PU*** sachant qu'il y a avec remise de ***10%*** si le montant dépasse ***1500*** Dh.

```
Ecrire ("Donnez le nombre d'articles et le prix unitaire:")
Lire ( N, PU)
PT ← N * PU
Si PT >1500 Alors
        PT ← PT - PT* 0.1
Finsi
Ecrire ("Le montant est: ",PT)
```



## L'ALGORITHMIQUE: Condition d'une sélection

- La condition dans une instruction de sélection est une expression logique ou booléenne dont la valeur est Vrai ou Faux
- Elle est exprimée en utilisant les opérateurs de comparaison (>, ≥, <, ≤, =, ≠) et les opérateurs logiques (ET, OU, NON)
- Exemple: On a trois variables A=4 , B=7 et C=19

| Condition | Valeur |
|---|---|
| A+2>5 **ET** B<C | Vrai |
| A<C **OU** B>C | Vrai |
| (A<0 **OU** C>B) **ET** (A ≠ B-3) | Faux |
| **NON** (A<C **OU** B>C) | Faux |

- On peut stocker la valeur d'une expression booléenne dans une variable de type Booléen

*Exemple:* Soient A, B et C les coefficients d'une équation de second degré.

```
Delta ← B*B – 4*A*C
V ← ( Delta > 0 )
```

```
Si V=Vrai ALORS          Si V ALORS
   …..                      ….
```

## L'ALGORITHMIQUE: La sélection imbriquée

- La **sélection imbriquée** permet de demander à l'ordinateur de choisir **un bloc** d'instructions à exécuter parmi **plusieurs** blocs possibles dont chacun est exécuté lorsqu'une condition spécifique est satisfaite.
- La syntaxe en pseudo-code de cette instruction est la suivante:

```
SI   Condition   ALORS
Bloc1 ---       instruction1
                ...
SINON
                instruction1'
Bloc2 ---       ...
FinSi
```

- Le *Bloc1* et/ou le *Bloc2* contient une autre instruction de sélection (si…sinon).

- Exemple:

  Déterminer le signe d'un entier A

```
Lire (A)
Si A > 0 Alors
        Ecrire (A, " est positif")
Sinon
        Si A < 0 Alors
                Ecrire (A, " est négatif")
        Sinon
                Ecrire (A, " est nul")
        FinSi
FinSi
```



## L'ALGORITHMIQUE: La sélection multiple

- La **sélection multiple** permet de demander à l'ordinateur de choisir **un bloc** d'instructions à exécuter permis **plusieurs** blocs possibles selon la valeur d'une expression donnée.
- La syntaxe en pseudo-code de cette instruction est la suivante:

```
SELON Expression
    Valeur1 : Bloc1
    Valeur2 : Bloc2
    ….
    Valeurn : Blocn
    SINON   : Bloc0
FinSelon
```

- Le $Bloc_i$ est exécuté si la valeur de ***Expression*** est égale à $Valeur_i$
- Le $Bloc_0$ est exécuté lorsque la valeur de **Expression** est différente de toutes les valeurs $Valeur_i$

- **Exemple**:

  Effectuer une opération (+, -, * ou /) sur deux deux entiers A et B selon la valeur d'un caractère C

```
Lire (A, B)
Lire (C)
SELON C
    '+': Ecrire ("La Somme=", A+B)
    '-' : Ecrire ("La différence=", A-B)
    '*' : Ecrire ("Le produit=", A*B)
    '/' :  Si B≠0 Alors
                Ecrire ("Le quotient=", A/B)
           Sinon
                Ecrire("Division impossible")
           FinSi
    SINON: Ecrire ("Opération invalide")
FinSelon
```

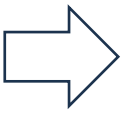
## L'ALGORITHMIQUE: La boucle POUR (1/2)

- Les **instructions d'itération** ou de répétition permettent d'exécuter un bloc d'instructions plusieurs fois.
- Ces instructions sont appelées des **boucles**.
- Le bloc d'instructions répétées constitue le **corps de la boucle**.
- La boucle **POUR** est utilisée lorsque le nombre de répétitions d'un bloc est connu à l'avance.
- **Exemple**: Calculer et afficher la somme S des entiers entre 1 et N (*saisi au clavier*)

S = 1 + 2 + 3 + ….+N

```
Lire (N)
S ← 0
S ← S + 1
S ← S + 2
S ← S + 3
…..
S ← S + N

Ecrire ("La somme= ", S)
```



```
Lire (N)
S ← 0
Pour i ← 1 à N  Pas=1
        S ← S + i
FinPour
Ecrire ("La somme= ", S)
```

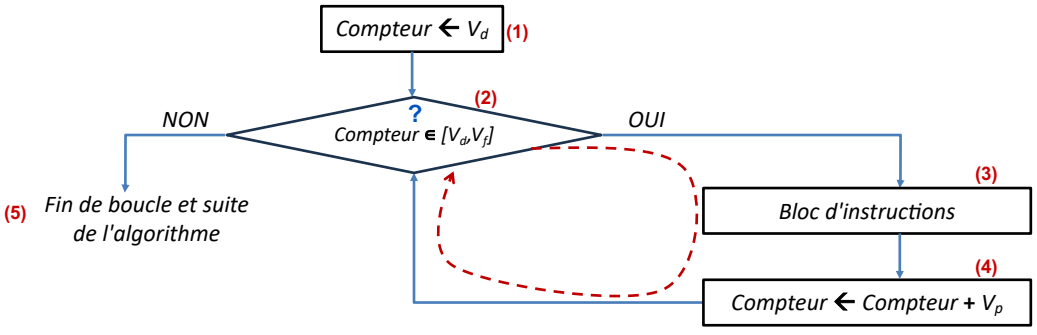


## L'ALGORITHMIQUE: La boucle POUR (2/2)

- La syntaxe en pseudo-code de la boucle POUR est la suivante:

```
POUR Compteur ← Vd à Vf  Pas=Vp
        Bloc d'Instructions
FinPour
```

- Cette boucle est exécutée selon l'organigramme suivant:




- Exercice: Afficher les nombres impairs se trouvant entre 1 et N

## L'ALGORITHMIQUE: La boucle TANT QUE (1/3)

- La boucle **TANT QUE** permet de répéter un bloc d'instructions plusieurs fois tant qu'une **condition** donnée est vérifiée.
- La syntaxe de cette boucle en pseudo-code est la suivante:

```
TANT QUE Condition        ..... Condition d'exécution
    Bloc d'Instructions   } Bloc d'instructions à répéter
FinTQ
```

- Exemple: Calculer et afficher la somme S des entiers entre 1 et N (*saisi au clavier*)

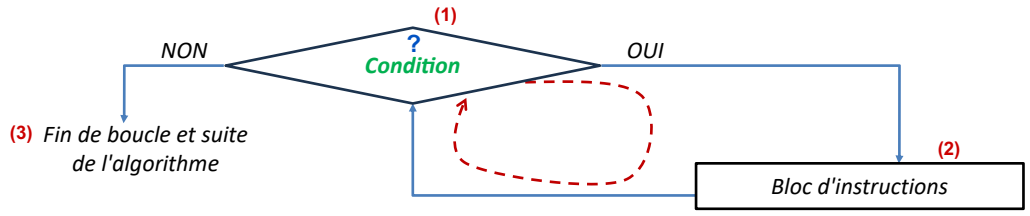
S = 1 + 2 + 3 + ….+N

```
Lire (N)
S ← 0
i ← 1
TANT QUE i ≤ N
   S ← S + i
   i ← i + 1
FinTQ
Ecrire ("La somme= ", S)
```



## L'ALGORITHMIQUE: La boucle TANT QUE (2/3)

- La boucle **TANT QUE** est exécutée selon l'organigramme suivant:




- ⚠ Il faut noter que l'ordinateur commencera par vérifier si la condition est vraie ou fausse. Il exécutera ensuite le bloc d'instructions si la condition est vraie.
- ⚠ Si la condition est toujours vérifiée, on aura une boucle **infinie**.
- ⚠ Le bloc d'instructions de la boucle **TANT QUE** ne sera jamais exécuté si la condition est fausse dès le départ.
- ⚠ Le bloc d'instructions dans "**TANT QUE**" sera donc exécuté **0** ou **plusieurs** fois.

## L'ALGORITHMIQUE: La boucle TANT QUE (1/3)

☞ ***Exemple:*** Si l'utilisateur saisit ***N ≤ 0*** dans l'algorithme précédent, les deux instructions du corps de la boucle ne seront jamais exécutées.

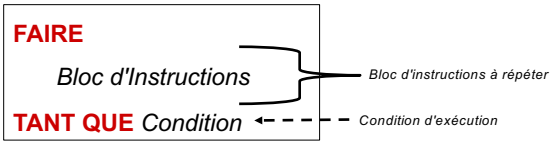
```
Lire (N)
S ← 0
i ← 1
TANT QUE  i ≤ N
   S ← S + i
   i ← i + 1
FinTQ
Ecrire  ("La somme= ", S)
```



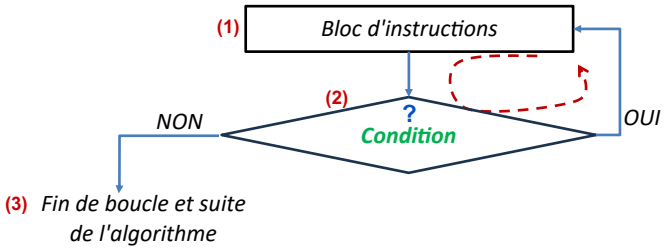
## L'ALGORITHMIQUE: La boucle FAIRE…TANT QUE (1/2)

☞ L'instruction d'itération **FAIRE…TANT QUE** permet d'exécuter un bloc d'instructions plusieurs fois tant qu'une condition donnée est vérifiée **en commençant par exécuter le bloc** d'instructions **avant de vérifier si la condition est toujours vraie**.

☞ La syntaxe en pseudo-code de cette boucle est la suivante:




☞ La boucle **FAIRE…TANT QUE** est exécutée selon l'organigramme suivant:



## L'ALGORITHMIQUE: La boucle FAIRe…TANT QUE (2/2)

- ⚠ Il faut rappeler que l'ordinateur commencera par vérifier si la condition est vraie ou fausse. Il exécutera ensuite le bloc d'instructions si la condition est vraie.
- ⚠ Le corps de la boucle **FAIRE…TANT QUE** es donc t exécuté **au moins** une fois.
- **Exemple**: Afficher les entiers entre 0 et N à l'envers (Hypothèse: N≥0)

```
FAIRE
    Ecrire ( N)
    N← N-1
TANT QUE  N >= 0
```

*Remarque:*
*Après la sortie de la boucle, N vaut -1*

- **Exemple:** Exiger la saisie d'une entier N strictement positif avant de poursuivre l'algorithme

```
FAIRE
    Ecrire ( "Donnez un entier >0")
    Lire (N)
TANT QUE  N ≤ 0
```



## L'ALGORITHMIQUE: Les boucle imbriquées

- ⚠ Le corps d'une boucle peut contenir à son tour une autre boucle.
- **Exemple:** Calculer le nombre de diviseurs de chaque entier dans l'intervalle [A, B] avec A et B saisis au clavier.

**On doit prendre les entiers de [A, B] un par un** ⇨ **Boucle pour parcourir l'intervalle [A,B] élément par élément**

**Pour un entier X de [A,B] on doit calculer le nombre de ses diviseurs qui sont dans [1,X]** ⇨ **Boucle pour parcourir l'intervalle [1,X], trouver les diviseurs de X et calculer leur nombre.**

```
Pour X ← A à B  Pas=1
    Nd ← 0
    Pour i ← 1 à X Pas=1
        Si X mod i =0 Alors
            Nd ← Nd +1
        FinSI
    FinPour
    Ecrire ("Nombre de diviseurs de ", X " est", Nd)
FinPour
```

Ce code sera exécuté pour chaque valeur de X allant de A jusqu'à B.
Il sera exécuté pour X=A, X=A+1, …., X=B