Cours d'Informatique 1

1ère année SM

2024/2025, Semestre 1

SOUIOU. W

Département de Sciences de la matiere,

Université Badji Mokhtar Annaba

souiou@yahoo.fr

Objectif et plan du cours

- **Objectif:**
  - Apprendre les concepts de base de l'algorithmique et de la programmation
  - Etre capable de mettre en œuvre ces concepts pour analyser des problèmes simples et écrire les programmes correspondants

Etapes de réalisation d'un programme

Enoncé du problème

↓ *Spécification*

Cahier des charges

↓ *Analyse*

Algorithme

↓ *Traduction en langage*

Programme source

↓ *Compilation*

Programme exécutable

↓ *Tests et modifications*

Version finale et résultats

La réalisation de programmes passe par l'écriture d'algorithmes

$\Longrightarrow$ D'où l'intérêt de l'**Algorithmique**

Algorithmique

- Le terme **algorithme** vient du nom du mathématicien arabe **Al-Khawarizmi** (820 après J.C.)
- Un algorithme est une description complète et détaillée des actions à effectuer et de leur séquencement pour arriver à un résultat donné
  - Intérêt: séparation analyse/codage (pas de préoccupation de syntaxe)
  - Qualités: **exact** (fournit le résultat souhaité), **efficace** (temps d'exécution, mémoire occupée), **clair** (compréhensible), **général** (traite le plus grand nombre de cas possibles), …
- **L'algorithmique** désigne aussi la discipline qui étudie les algorithmes et leurs applications en Informatique
- Une bonne connaissance de l'algorithmique permet d'écrire des algorithmes exacts et efficaces

Représentation d'un algorithme

Historiquement, deux façons pour représenter un algorithme:

- **L'Organigramme:** représentation graphique avec des symboles (carrés, losanges, etc.)
  - offre une vue d'ensemble de l'algorithme
  - représentation quasiment abandonnée aujourd'hui

- **Le pseudo-code:** représentation textuelle avec une série de conventions ressemblant à un langage de programmation (sans les problèmes de syntaxe)
  - plus pratique pour écrire un algorithme
  - représentation largement utilisée

Algorithmique

Notions et instructions de base

Notion de variable

- Dans les langages de programmation une **variable** sert à stocker la valeur d'une donnée

- Une variable désigne en fait un emplacement mémoire dont le contenu peut changer au cours d'un programme (d'où le nom variable)

- Règle : Les variables doivent être **déclarées** avant d'être utilisées, elle doivent être caractérisées par :
  - un nom (**Identificateur**)
  - un **type** (entier, réel, caractère, chaîne de caractères, …)

Choix des identificateurs (1)

Le choix des noms de variables est soumis à quelques règles qui varient selon le langage, mais en général:

- Un nom doit commencer par une lettre alphabétique
  exemple valide: A1 **exemple invalide: 1A**
- doit être constitué uniquement de lettres, de chiffres et du soulignement _ (Eviter les caractères de ponctuation et les espaces)
  valides: SMIP2007, SMP_2007 **invalides: SMP 2005,SMI-2007,SMP;2007**
- doit être différent des mots réservés du langage (par exemple en Java: **int, float, else, switch, case, default, for, main, return**, …)
- La longueur du nom doit être inférieure à la taille maximale spécifiée par le langage utilisé

Choix des identificateurs (2)

Conseil: pour la lisibilité du code choisir des noms significatifs qui décrivent les données manipulées

exemples: TotalVentes2004, Prix_TTC, Prix_HT

Remarque: en pseudo-code algorithmique, on va respecter les règles citées, même si on est libre dans la syntaxe

Types des variables

Le type d'une variable détermine l'ensemble des valeurs qu'elle peut prendre, les types offerts par la plus part des langages sont:

- Type numérique (entier ou réel)
  - **Byte** (codé sur 1octet): de 0 à 255
  - **Entier court** (codé sur 2 octets) : -32 768 à 32 767
  - **Entier long** (codé sur 4 ou 8 octets)
  - **Réel simple précision** (codé sur 4 octets)
  - **Réel double précision** (codé sur 8 octets)
- Type logique ou booléen: deux valeurs VRAI ou FAUX
- Type caractère: lettres majuscules, minuscules, chiffres, symboles, …

  exemples: 'A', 'a', '1', '?', …
- Type chaîne de caractère: toute suite de caractères,

  exemples: " Nom, Prénom", "code postale: 1000", …

Déclaration des variables

- Rappel: toute variable utilisée dans un programme doit avoir fait l'objet d'une déclaration préalable
- En pseudo-code, on va adopter la forme suivante pour la déclaration de variables

Variables liste d'identificateurs : type

- Exemple:

Variables i, j,k : entier

x, y : réel

OK: booléen

ch1, ch2 : chaîne de caractères

- Remarque: pour le type numérique on va se limiter aux entiers et réels sans considérer les sous types

L'instruction d'affectation

- **l'affectation** consiste à attribuer une valeur à une variable
  (ça consiste en fait à remplir où à modifier le contenu d'une zone mémoire)

- En pseudo-code, l'affectation se note avec le signe ←

  Var← e: attribue la valeur de e à la variable Var

  - e peut être une valeur, une autre variable ou une expression
  - Var et e doivent être de même type ou de types compatibles
  - l'affectation ne modifie que ce qui est à gauche de la flèche

- **Ex valides:** **i ←1** **j ←i** **k ←i+j**
  x ←10.3 **OK ←FAUX** **ch1 ←"SMI"**
  ch2 ←ch1 **x ←4** **x ←j**

  (voir la déclaration des variables dans le transparent précédent)

- **non valides:** **i ←10.3** **OK ←"SMI"** **j ←x**

Quelques remarques

- Beaucoup de langages de programmation (C/C++, Java, …) utilisent le signe égal = pour l'affectation ←. Attention aux confusions:
  - l'affectation n'est pas commutative : A=B est différente de B=A
  - l'affectation est différente d'une équation mathématique :
    - A=A+1 a un sens en langages de programmation
    - A+1=2 n'est pas possible en langages de programmation et n'est pas équivalente à A=1
- Certains langages donnent des valeurs par défaut aux variables déclarées. Pour éviter tout problème il est préférable **d'initialiser les variables** déclarées

Exercices simples sur l'affectation (1)

Donnez les valeurs des variables A, B et C après exécution des instructions suivantes ?

Variables A, B, C: **Entier**
Début
A ← 3
B ← 7
A ← B
B ← A+5
C ← A + B
C ← B – A
Fin

Exercices simples sur l'affectation (2)

Quelles seront les valeurs des variables A et B après exécution des instructions suivantes ?

Var A, B :**Entier ;**

Début

A **:=** 1 ;

B **:=** A + 3 ;

A **:=** 3 ;

Fin.

Exercices simples sur l'affectation (2)

Donnez les valeurs des variables A et B après exécution des instructions suivantes ?

Variables A, B : **Entier**
Début
A ← 1
B ← 2
A ← B
B ← A
Fin

Les deux dernières instructions permettent-elles d'échanger les valeurs de A et B ?

Exercices simples sur l'affectation (3)

Ecrire un algorithme permettant d'échanger les valeurs de deux variables A et B

Expressions et opérateurs

- Une **expression** peut être une valeur, une variable ou une opération constituée de variables reliées par des **opérateurs**

  exemples: 1, b, a*2, a+ 3*b-c, …

- L'évaluation de l'expression fournit une valeur unique qui est le résultat de l'opération

- Les **opérateurs** dépendent du type de l'opération, ils peuvent être :

  - **des opérateurs arithmétiques: +, -, *, /, % (modulo), ^ (puissance)**
  - **des opérateurs logiques: NON, OU, ET**
  - **des opérateurs relationnels: =, ≠ , <, >, <=, >=**
  - **des opérateurs sur les chaînes: & (concaténation)**

- Une expression est évaluée de gauche à droite mais en tenant compte de **priorités**

Priorité des opérateurs

- Pour les opérateurs arithmétiques donnés ci-dessus, l'ordre de priorité est le suivant (du plus prioritaire au moins prioritaire) :
  - ^ : (élévation à la puissance)
  - * , / (multiplication, division)
  - % (modulo)
  - + , - (addition, soustraction)

exemple: 2 + 3 * 7 vaut 23

- En cas de besoin (ou de doute), on utilise les parenthèses pour indiquer les opérations à effectuer en priorité

exemple: (2 + 3) * 7 vaut 35

Excercice

Que produit l'algorithme suivant ?

Var A, B, C **: Caractères ;**

Début

A := “423“ ;

B := “12” ;

C := A + B ;

Fin.

Excercice

Que produit l'algorithme suivant ?

Var A, B, C : **Caractères ;**

Début

A := "423" ;

B := "12" ;

C := A & B ;

Fin

Les instructions d'entrées-sorties: lecture et écriture (1)

- Les instructions de lecture et d'écriture permettent à la machine de communiquer avec l'utilisateur
- La **lecture** permet d'entrer des donnés à partir du clavier
  - En pseudo-code, on note: **lire (var)**

    la machine met la valeur entrée au clavier dans la zone mémoire nommée var
  - Remarque: Le programme s'arrête lorsqu'il rencontre une instruction Lire et ne se poursuit qu'après la frappe d'une valeur au clavier et de la touche Entrée

Les instructions d'entrées-sorties: lecture et écriture (2)

- **L'écriture** permet d'afficher des résultats à l'écran (ou de les écrire dans un fichier)

  - En pseudo-code, on note: **écrire (var)**

    la machine affiche le contenu de la zone mémoire var

  - Conseil: Avant de lire une variable, il est fortement conseillé d'écrire des messages à l'écran, afin de prévenir l'utilisateur de ce qu'il doit frapper

Exemple (lecture et écriture)

Ecrire un algorithme qui demande un nombre entier à l'utilisateur, puis qui calcule et affiche le double de ce nombre

Exemple (lecture et écriture)

Ecrire un algorithme qui demande un nombre entier à l'utilisateur, puis qui calcule et affiche le double de ce nombre

Algorithme Calcul_double

variables A, B : **entier**

Début

écrire("entrer le nombre ")

lire(A)

B ← 2*A

écrire("le double de ", A, "est :", B)

Fin

Exercice (lecture et écriture)

Ecrire un algorithme qui vous demande de saisir votre nom puis votre prénom et qui affiche ensuite votre nom complet

Exercice (lecture et écriture)

Ecrire un algorithme qui vous demande de saisir votre nom puis votre prénom et qui affiche ensuite votre nom complet

Algorithme AffichageNomComplet

variables Nom, Prenom, Nom_Complet : **chaîne de caractères**

Début

écrire("entrez votre nom")

lire(Nom)

écrire("entrez votre prénom")

lire(Prenom)

Nom_Complet ← Nom & ’’ ‘’& Prenom

écrire("Votre nom complet est : ", Nom_Complet)

Fin

Exercice (lecture et écriture)

- Ecrire un algorithme qui lit le prix HT d'un article, le nombre d'articles et le taux de TVA, et qui fournit le prix total TTC correspondant. Faire en sorte que des libellés apparaissent clairement.

Exercice (lecture et écriture)

Algorithme Calcul_Prix;
Variables pht, ttva, pttc **en Réel**
nb **en Entier**
Début
Ecrire “Entrez le prix hors taxes :”
Lire (pht)
Ecrire “Entrez le nombre d’articles :”
Lire (nb)
Ecrire “Entrez le taux de TVA :”
Lire (ttva)
pttc ← nb * pht * (1 + ttva/100)
Ecrire “Le prix toutes taxes est : ”, pttc
Fin

Tests: instructions conditionnelles (1)

- Les instructions conditionnelles servent à n'exécuter une instruction ou une séquence d'instructions que si une condition est vérifiée
- On utilisera la forme suivante: **Si** condition **alors**

  instruction ou suite d'instructions1

  Sinon

  instruction ou suite d'instructions2

  Finsi

  - la condition ne peut être que vraie ou fausse
  - si la condition est vraie, se sont les instructions1 qui seront exécutées
  - si la condition est fausse, se sont les instructions2 qui seront exécutées
  - la condition peut être une condition simple ou une condition composée de plusieurs conditions

Tests: instructions conditionnelles (2)

- La partie Sinon n'est pas obligatoire, quand elle n'existe pas et que la condition est fausse, aucun traitement n'est réalisé

  - On utilisera dans ce cas la forme simplifiée suivante:

Si condition **alors**

instruction ou suite d'instructions1

Finsi

Exercice (tests)

Ecrire un algorithme qui demande un nombre entier à l'utilisateur, puis qui teste et affiche s'il est divisible par 3

Exercice (tests)

Ecrire un algorithme qui demande un nombre entier à l'utilisateur, puis qui teste et affiche s'il est divisible par 3

```
Algorithme Divsible_par3
Variable n : entier
Début
        Ecrire " Entrez un entier : "
        Lire (n)
        Si (n%3=0) alors
                Ecrire (n," est divisible par 3")
        Sinon
                Ecrire (n," n'est pas divisible par 3")
        Finsi
Fin
```

Exercice (tests)

- Ecrire un algorithme qui demande un nombre à l'utilisateur, et l'informe ensuite si ce nombre est positif ou négatif (on laisse de côté le cas où le nombre vaut zéro).

Exercice (tests)

- Ecrire un algorithme qui demande un nombre à l'utilisateur, et l'informe ensuite si ce nombre est positif ou négatif (on laisse de côté le cas où le nombre vaut zéro).

Variable n **en Entier**
Début
Ecrire "Entrez un nombre : "
Lire (n)
Si n > 0 **Alors**
    Ecrire "Ce nombre est positif"
Sinon
    Ecrire "Ce nombre est négatif"
Finsi
Fin

Conditions composées

- Une condition composée est une condition formée de plusieurs conditions simples reliées par des opérateurs logiques:

  ET, OU, OU exclusif (XOR) et NON

- Exemples :
  - x compris entre 2 et 6 : (x > 2) ET (x < 6)
  - n divisible par 3 ou par 2 : (n%3=0) OU (n%2=0)
  - deux valeurs et deux seulement sont identiques parmi a, b et c : (a=b) XOR (a=c) XOR (b=c)

- L'évaluation d'une condition composée se fait selon des règles présentées généralement dans ce qu'on appelle tables de vérité

Tables de vérité

C1	C2	**C1 ET C2**
VRAI	VRAI	**VRAI**
VRAI	FAUX	**FAUX**
FAUX	VRAI	**FAUX**
FAUX	FAUX	**FAUX**

C1	C2	**C1 OU C2**
VRAI	VRAI	**VRAI**
VRAI	FAUX	**VRAI**
FAUX	VRAI	**VRAI**
FAUX	FAUX	**FAUX**

C1	C2	**C1 XOR C2**
VRAI	VRAI	**FAUX**
VRAI	FAUX	**VRAI**
FAUX	VRAI	**VRAI**
FAUX	FAUX	**FAUX**

C1	**NON C1**
VRAI	**FAUX**
FAUX	**VRAI**

Exercice (tests)

Ecrire un algorithme qui demande deux nombres à l'utilisateur et l'informe ensuite si leur produit est négatif ou positif (on laisse de côté le cas où le produit est nul).

Attention toutefois : on ne doit pas calculer le produit des deux nombres.

Exercice (tests)

Ecrire un algorithme qui demande deux nombres à l'utilisateur et l'informe ensuite si leur produit est négatif ou positif (on laisse de côté le cas où le produit est nul).

Attention toutefois : on ne doit pas calculer le produit des deux nombres.

Variables m, n **en Entier**

Début

Ecrire "Entrez deux nombres : "

Lire(m, n)

Si (m > 0 ET n > 0) OU (m < 0 ET n < 0) **Alors**

    Ecrire "Leur produit est positif"

Sinon

    Ecrire "Leur produit est négatif"

Finsi

Fin

Tests imbriqués

- Les tests peuvent avoir un degré quelconque d'imbrications

```
Si condition1 alors
        Si condition2 alors
                instructionsA
        Sinon
                instructionsB
         Finsi
Sinon
        Si condition3 alors
                instructionsC
        Finsi
Finsi
```

Tests imbriqués: exemple 1

Ecrire un algorithme qui demande un nombre à l'utilisateur, et l'informe ensuite si ce nombre est positif ou négatif (on inclut cette fois le traitement du cas où le nombre vaut zéro).

Tests imbriqués: exemple 1 (version 1)

```
Variable n : entier
Début
        Ecrire ("entrez un nombre : ")
        Lire (n)
        Si (n < 0) alors
                Ecrire ("Ce nombre est négatif")
        Sinon
                Si (n = 0) alors
                        Ecrire ("Ce nombre est nul")
                Sinon
                        Ecrire ("Ce nombre est positif")
                Finsi
        Finsi

Fin
```

Tests imbriqués: exemple 1 (version 2)

```
Variable n : entier
 Début
      Ecrire ("entrez un nombre : ")
      Lire (n)
      Si (n < 0) alors    Ecrire ("Ce nombre est négatif")
      Finsi
      Si (n = 0) alors    Ecrire ("Ce nombre est nul")
      Finsi
      Si (n > 0) alors    Ecrire ("Ce nombre est positif")
      Finsi
 Fin
```

Remarque : dans la version 2 on fait trois tests systématiquement alors que dans la version 1, si le nombre est négatif on ne fait qu'un seul test

Conseil : utiliser les tests imbriqués pour limiter le nombre de tests et placer d'abord les conditions les plus probables

Tests imbriqués: exercice

- Ecrire un algorithme qui demande deux nombres à l'utilisateur et l'informe ensuite si le produit est négatif ou positif (on inclut cette fois le traitement du cas où le produit peut être nul). Attention toutefois, on ne doit pas calculer le produit !

Tests imbriqués: exercice

```
Variables m, n en Entier
Début
Ecrire “Entrez deux nombres : ”
Lire m, n
Si m = 0 OU n = 0 Alors
Ecrire “Le produit est nul”
Sinon
        Si (m < 0 ET n < 0) OU (m > 0 ET n > 0) Alors
            Ecrire “Le produit est positif”
        Sinon
             Ecrire “Le produit est négatif”
        Finsi
        Fin
```

Tests imbriqués: exercice

Un magasin de reprographie facture 0,10 DA les dix premières photocopies, 0,09 DA les vingt suivantes et 0,08 DA au-delà.

Ecrivez un algorithme qui demande à l'utilisateur le nombre de photocopies effectuées et qui affiche la facture correspondante.

Tests imbriqués: corrigé de l'exercice

```
Variable n en Entier
         p en Réel
Début
Ecrire "Nombre de photocopies : "
Lire n
Si n <= 10 Alors
p ← n * 0,1
SinonSi n <= 30
p ← 10 * 0,1 + (n – 10) * 0,09
Sinon
p ← 10 * 0,1 + 20 * 0,09 + (n – 30) * 0,08
FinSi
Ecrire "Le prix total est : ", p
Fin
```

Instructions itératives: les boucles

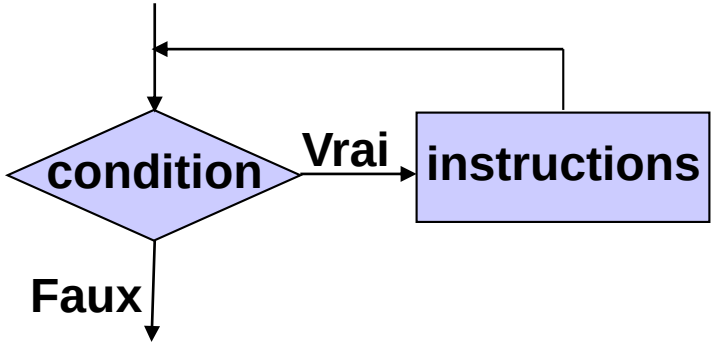
- Les boucles servent à répéter l'exécution d'un groupe d'instructions un certain nombre de fois
- On distingue trois sortes de boucles en langages de programmation :
  - Les **boucles tant que** : on y répète des instructions tant qu'une certaine condition est réalisée
  - Les **boucles jusqu'à** : on y répète des instructions jusqu'à ce qu'une certaine condition soit réalisée
  - Les **boucles pour** ou avec compteur : on y répète des instructions en faisant évoluer un compteur (variable particulière) entre une valeur initiale et une valeur finale

Les boucles Tant que

TantQue (condition)

instructions

FinTantQue



- la condition (dite condition de contrôle de la boucle) est évaluée avant chaque itération
- si la condition est vraie, on exécute instructions (corps de la boucle), puis, on retourne tester la condition. Si elle est encore vraie, on répète l'exécution, …
- si la condition est fausse, on sort de la boucle et on exécute l'instruction qui est après FinTantQue

Les boucles Tant que : remarques

- Le nombre d'itérations dans une boucle TantQue n'est pas connu au moment d'entrée dans la boucle. Il dépend de l'évolution de la valeur de condition

- Une des instructions du corps de la boucle doit absolument changer la valeur de condition de vrai à faux (après un certain nombre d'itérations), sinon le programme tourne indéfiniment

$\Rightarrow$ **Attention aux boucles infinies**

- Exemple de boucle infinie :

```
i ← 2
TantQue   (i > 0)
i ←  i+1        (attention aux erreurs de frappe : + au lieu de -)
FinTantQue
```

Boucle Tant que : exemple1

- Ecrire un algorithme qui demande à l'utilisateur un nombre compris entre 1 et 3 jusqu'à ce que la réponse convienne.

Boucle Tant que : exemple1

```
Variable N en Entier
Debut
N ← 0
Ecrire ("Entrez un nombre entre 1 et 3")
TantQue N < 1 ou N > 3
    Lire (N)
    Si N < 1 ou N > 3 Alors
        Ecrire "Saisie erronée. Recommencez"
    FinSi
FinTantQue
Fin
```

Boucle Tant que : exemple2

- Ecrire un algorithme qui demande un nombre compris entre 10 et 20, jusqu'à ce que la réponse convienne. En cas de réponse supérieure à 20, on fera apparaître un message : « Plus petit ! », et inversement, « Plus grand ! » si le nombre est inférieur à 10.

Boucle Tant que : exemple2

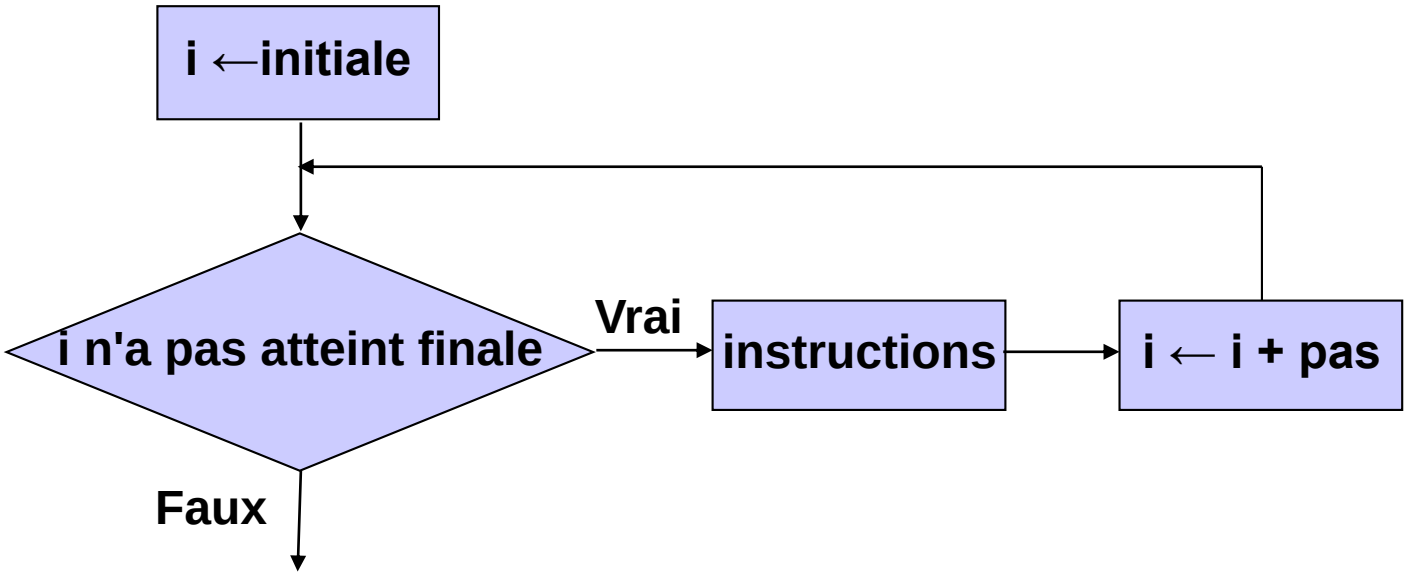
```
Variable N en Entier
Debut
N ← 0
Ecrire “Entrez un nombre entre 10 et 20”
TantQue N < 10 ou N > 20
    Lire N
    Si N < 10 Alors
        Ecrire “Plus grand !”
     SinonSi N > 20 Alors
          Ecrire “Plus petit !”
    FinSi
FinTantQue
Fin
```

Les boucles Pour

Pour compteur **allant de** initiale **à** finale par **pas** valeur du pas

instructions

FinPour



Les boucles Pour

- Remarque : le nombre d'itérations dans une boucle Pour est connu avant le début de la boucle

- **Compteur** est une variable de type entier. Elle doit être déclarée

- **Pas** est un entier qui peut être positif ou négatif. **Pas** peut ne pas être mentionné, car par défaut sa valeur est égal à 1.

- **Initiale et finale** peuvent être des valeurs, des variables définies avant le début de la boucle ou des expressions de même type que compteur

Déroulement des boucles Pour

1) La valeur initiale est affectée à la variable compteur

2) On compare la valeur du compteur et la valeur de finale :

   a) Si la valeur du compteur est > à la valeur finale dans le cas d'un pas positif (ou si compteur est < à finale pour un pas négatif), on sort de la boucle et on continue avec l'instruction qui suit FinPour

   b) Si compteur est <= à finale dans le cas d'un pas positif (ou si compteur est >= à finale pour un pas négatif), instructions seront exécutées

      i. Ensuite, la valeur de compteur est incrémentée de la valeur du pas si pas est positif (ou décrémenté si pas est négatif)

      ii. On recommence l'étape 2 : La comparaison entre compteur et finale est de nouveau effectuée, et ainsi de suite …

Boucle Pour : exemple1

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

Boucle Pour : exemple1

- **Variables** N, i **en Entier**
  Debut
  Ecrire (“Entrez un nombre : ”)
  Lire (N)
  Ecrire (“Les 10 nombres suivants sont : ”)
  Pour i ← N + 1 **à** N + 10
  Ecrire (i)
  i **Suivant**
  Fin.

Boucle Pour : exemple2

Ecrire un algorithme qui demande un nombre de départ, et qui ensuite écrit la table de multiplication de ce nombre, présentée comme suit (cas où l'utilisateur entre le nombre 7) :

Table de 7 :

7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
…
7 x 10 = 70

Boucle Pour : exemple2

Variables N, i **en Entier**
Debut
Ecrire (“Entrez un nombre : ”)
Lire (N)
Ecrire (“La table de multiplication de ce nombre est : ”)
Pour i ← 1 **à** 10
Ecrire (N,“ x ”,i, “ = ”, n*i)
i **Suivant**
Fin.

Boucle Pour : exemple3

- **Ecrire un algorithme qui demande un nombre de départ, et qui calcule la somme des entiers jusqu'à ce nombre. Par exemple, si l'on entre 5, le programme doit calculer :**

  1 + 2 + 3 + 4 + 5 = 15

Boucle Pour : exemple3

```
Variables N, i, Som en Entier
    Debut
        Ecrire (“Entrez un nombre : ”)
        Lire (N)
        Som ← 0
        Pour i ← 1 à N
                    Som ← Som + i
        i Suivant
        Ecrire (“La somme est : ”, Som)
    Fin.
```

Boucle Pour : exemple4

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

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

Boucle Pour : exemple4

```
Variables N, i, F en Entier
    Debut
        Ecrire (“Entrez un nombre : ”)
        Lire (N)
        F ← 1
        Pour i ← 2 à N
                    F ← F * i
        i Suivant
        Ecrire (“La factorielle est : ”, F)
    Fin.
```

Boucle Pour : remarque

- Il faut éviter de modifier la valeur du compteur (et de finale) à l'intérieur de la boucle. En effet, une telle action :
  - perturbe le nombre d'itérations prévu par la boucle Pour
  - rend difficile la lecture de l'algorithme
  - présente le risque d'aboutir à une boucle infinie

Exemple : **Pour** i **allant de** 1 **à** 5
i ← i -1
écrire(" i = ", i)
Finpour

Lien entre Pour et TantQue

La boucle Pour est un cas particulier de Tant Que (cas où le nombre d'itérations est connu et fixé) . Tout ce qu'on peut écrire avec Pour peut être remplacé avec TantQue (la réciproque est fausse)

Pour compteur **allant de** initiale **à** finale par **pas** valeur du pas

instructions

FinPour

peut être remplacé par :

(cas d'un pas positif)

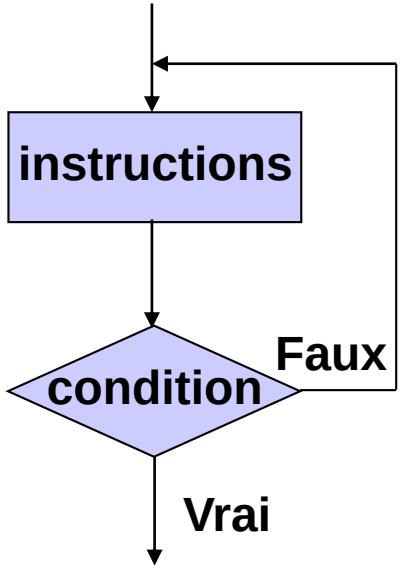
compteur ← initiale
TantQue compteur <= finale
instructions
compteur ← compteur+pas
FinTantQue

Les boucles Répéter … jusqu'à …

Répéter

instructions

Jusqu'à condition



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

Boucle Répéter jusqu'à : exemple

Un algorithme qui détermine le premier nombre entier N tel que la somme de 1 à N dépasse strictement 100 **(version avec répéter jusqu'à)**

```
Variables som, i : entier
Debut
        som ← 0
        i ← 0
        Répéter
                i ← i+1
                som ← som+i
        Jusqu'à ( som > 100)
        Ecrire (" La valeur cherchée est N=  ", i)
Fin
```

Choix d'un type de boucle

- Si on peut déterminer le nombre d'itérations avant l'exécution de la boucle, il est plus naturel d'utiliser *la boucle Pour*
- S'il n'est pas possible de connaître le nombre d'itérations avant l'exécution de la boucle, on fera appel à l'une des *boucles TantQue ou répéter jusqu'à*
- Pour le choix entre *TantQue* et *jusqu'à :*
  - Si on doit tester la condition de contrôle avant de commencer les instructions de la boucle, on utilisera *TantQue*
  - Si la valeur de la condition de contrôle dépend d'une première exécution des instructions de la boucle, on utilisera *répéter jusqu'à*