مدرسة علوم المعلومات
ⵜⵉⵎⵔⵔⴰⵜ ⵏ ⵜⵓⵙⵙⵏⴰⵡⵉⵏ ⵉⵏⴼⵓⵔⵎⴰⵜⵉⵡⵏ
ECOLE DES SCIENCES DE L'INFORMATION
www.esi.ac.ma

Algorithmique

4. Les structures itératives

Sommaire

Rappel

Rappel

- Algorithme informatique
  - Suite d'instructions ordonnée qui décrit de façon exhaustive les différentes étapes à suivre par un processeur pour résoudre un problème donné en un temps fini

Rappel

- Pseudo-code algorithmique

ALGORITHME *nom_de_l'algorithme*

<partie des déclarations>

DEBUT

<partie des instructions>

//commentaire

FIN

Rappel

- Un programme n'est pas purement séquentiel ➔ nécessité d'avoir des **structures de contrôle**

1. Les structures alternatives (tests)
2. Les structures itératives (boucles)

Rappel

- Un programme n'est pas purement séquentiel ➔ nécessité d'avoir des **structures de contrôle**

1. **Les structures alternatives (tests)**
2. Les structures itératives (boucles)

Rappel

ALGORITHME *nom_de_l'algorithme*
<partie des déclarations>
DEBUT
séquence1
SI *condition1* **ALORS**
séquence2
FINSI
séquence3
FIN

Rappel

ALGORITHME *nom_de_l'algorithme*
<partie des déclarations>
DEBUT
séquence1
SI *condition1* **ALORS**
séquence2
SINON
séquence3
FINSI
séquence4
FIN

Rappel

- Une **condition** est un booléen qui peut être
  - Valeur booléenne
  - Expression booléenne
  - Comparaison
    - Entre deux valeurs ➔ condition simple
    - Entre plusieurs valeurs ➔ condition composée

Rappel

- Une **condition** est un booléen qui peut être
  - Valeur booléenne
  - Expression booléenne
  - Comparaison
- Pour qu'un test soit utile, il faut que la condition ne prenne pas toujours la même valeur (i.e. ne soit pas toujours fausse ou toujours vraie)

Rappel

- Un test imbriqué est exprimé comme suit

SI *condition 1* **ALORS**

séquence 1

SINON

SI *condition2* **ALORS**

séquence2

SINON

séquence3

FINSI

FINSI

Rappel

- Pour alléger l'écriture et améliorer la lisibilité, on peut fusionner SINON et SI en **SINONSI** ➔ un seul bloc de test

SI *condition I* **ALORS**
    séquence I
SINONSI *condition2* **ALORS**
    séquence2
SINON
    séquence3
FINSI

Rappel

- Les tests imbriqués présentent plusieurs avantages
  - Simplification de l'écriture des tests
  - Amélioration de la lisibilité de l'algorithme / programme
  - Réduction du temps d'exécution

Rappel

- Test « SUIVANT … CAS »
  - Permet de sélectionner le bloc à exécuter en fonction de la valeur d'une variable
  - Spécialisation de l'instruction SI … SINONSI
  - Utile quand une variable a plusieurs valeurs à tester

Rappel

- Test « SUIVANT … CAS »

SUIVANT *variable* **FAIRE**

CAS *valeur_1* : *sequence_1*

CAS *valeur_2* : *sequence_2*

…

CAS *valeur_n* : *sequence_n*

AUTRES CAS : sequence_autre

FINSUIVANT

Rappel

- Test « SELONQUE … CAS … »
  - Permet de sélectionner le bloc à exécuter en fonction de conditions
  - Spécialisation de l'instruction SI … SINONSI …
  - Utile quand il y a plusieurs conditions à tester

Rappel

- Test « SELONQUE … CAS … »

SELONQUE

condition_1 : *sequence_1*

condition_2 : *sequence_2*

…

condition_n : *sequence_n*

SINON: sequence_sinon

FINSELONQUE

Rappel

- Un programme n'est pas purement séquentiel ➔ nécessité d'avoir des **structures de contrôle**

1. Les structures alternatives (tests)
2. **Les structures itératives (boucles)**

Problématique

- Il est parfois nécessaire de répéter des instructions un certain nombre de fois
  - Exemples :
    - Calculer le prix TTC d'un produit saisi par l'utilisateur, puis permettre à ce dernier de saisir un autre, puis un autre …
    - Tester la valeur saisie au clavier par l'utilisateur et lui redemander de la saisir si elle est erronée
    - …

Problématique

- Il est parfois nécessaire de répéter des instructions un certain nombre de fois
- La répétition est réalisée en utilisant une **boucle**

Répétition

- Une **boucle** est une structure de contrôle de type itératif (ou répétitif)
- Elle permet de répéter, plusieurs fois, une instruction ou un ensemble d'instructions
- La répétition est soumise à une condition

Répétition

- On utilise trois types de boucles
  - TANTQUE … FAIRE
  - POUR
  - RÉPÉTER … JUSQU'À

La boucle « TANTQUE…FAIRE»

Boucle **TANTQUE … FAIRE**

- Permet de répéter une instruction tant qu'une condition est vraie

TANTQUE *condition* **FAIRE**

instructions

FINTANTQUE

Boucle **TANTQUE … FAIRE**

- Permet de répéter une instruction tant qu'une condition est vraie

TANTQUE *condition* **FAIRE**

instructions

FINTANTQUE

- La boucle s'arrête quand la condition est fausse

➔ Les instructions doivent modifier la valeur de la condition à un moment sinon : boucle infinie !

Boucle **TANTQUE … FAIRE**

- Permet de répéter une instruction tant qu'une condition est vraie

TANTQUE *condition* **FAIRE**

instructions

FINTANTQUE

- La boucle s'arrête quand la condition est fausse
- Le nombre d'itérations n'est pas connu à l'avance

Boucle **TANTQUE … FAIRE**

- Exemple

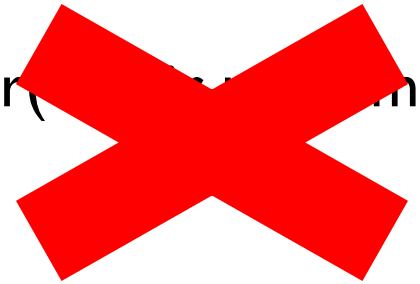
```
ALGORITHME exemple_boucle
VAR n : entier
DEBUT
   n ← 0
  TANTQUE n < 10 FAIRE
     n ← n + 1
  FINTANTQUE
FIN
```

Boucle **TANTQUE … FAIRE**

```
ALGORITHME exemple_boucle
VAR x : réel
DEBUT
   TANTQUE x < 0 FAIRE
      Afficher("Saisir un nombre positif")
      Lire(x)
    FINTANTQUE
   Afficher("Saisie terminée. Merci. ")
FIN
```

Boucle **TANTQUE … FAIRE**

```
ALGORITHME exemple_boucle
VAR x : réel
DEBUT
   TANTQUE x < 0 FAIRE
      Afficher( nbre positif")
      Lire(x)
   FINTANTQUE
   Afficher("Saisie terminée. Merci. ")
FIN
```



La boucle ne sera jamais exécutée car x n'est pas initialisé

Boucle **TANTQUE … FAIRE**

```
ALGORITHME exemple_boucle
VAR x : réel
DEBUT
  x ← −1
  TANTQUE x < 0 FAIRE
     Afficher("Saisir un nombre positif")
     Lire(x)
   FINTANTQUE
   Afficher("Saisie terminée. Merci. ")
FIN
```

Boucle **TANTQUE … FAIRE**

```
ALGORITHME exemple_boucle
VAR x : réel
DEBUT
   Afficher("Saisir un nombre positif")
   Lire(x)
   TANTQUE x < 0 FAIRE
      Afficher("Saisir un nombre positif")
      Lire(x)
    FINTANTQUE
    Afficher("Saisie terminée. Merci. ")
FIN
```

Boucle **TANTQUE … FAIRE**

- Il faut veiller à
  - Avoir une condition d'entrée dans la boucle qui soit réalisable (i.e. qui ne soit pas toujours fausse)
  - Initialiser les variables utilisées dans la condition d'entrée
  - Avoir une condition d'arrêt dans la boucle pour ne pas avoir de boucle infinie

Exercices

- Écrire un algorithme qui demande à l'utilisateur de saisir un nombre compris entre 0 et 10 et répète jusqu'à ce que la saisie soit correcte

Exercices

```
ALGORITHME boucle_saisie_nb
VAR n : réel
DEBUT
   Afficher("Saisir un nombre compris entre 0 et 10")
   Lire(n)
   TANTQUE n < 0 OU n > 10 FAIRE
        Afficher("Saisir un nombre compris entre 0 et 10")
        Lire(n)
    FINTANTQUE
    Afficher("Saisie terminée. Merci. ")
FIN
```

```
ALGORITHME boucle_saisie_nb
VAR n : réel
DEBUT
  n ← −1
  TANTQUE n < 0 OU n > 10 FAIRE
        Afficher("Saisir un nombre compris entre 0 et 10")
        Lire(n)
        SI n < 0 OU n > 10 ALORS
            Afficher("Saisie erronée. Veuillez recommencer. ")
        FINSI
    FINTANTQUE
   Afficher("Saisie terminée. Merci. ")
FIN
```

Exercices

- Écrire un algorithme qui demande à l'utilisateur de saisir un nombre compris entre 0 et 10 et répète jusqu'à ce que la saisie soit correcte
  - Si le nombre est inférieur à 0, afficher un message demandant un nombre plus grand
  - Si le nombre est supérieur à 10, afficher un message demandant un nombre plus petit

```
ALGORITHME boucle_saisie_nb
VAR n : réel
DEBUT
  Afficher("Saisir un nombre compris entre 0 et 10")
  Lire(n)
  TANTQUE n < 0 OU n > 10 FAIRE
      SI n < 0 ALORS
          Afficher("Donner un nombre plus grand. ")
          Lire(n)
      SINONSI n > 10 ALORS
          Afficher("Donner un nombre plus petit. ")
          Lire(n)
      FINSI
   FINTANTQUE
  Afficher("Saisie terminée. Merci. ")
FIN
```

Exercices

- Écrire un algorithme qui demande à l'utilisateur de saisir un nombre puis qui décrémente ce nombre de 1 jusqu'à atteindre 0 en affichant la valeur de chaque décrémentation

Exercices

```
ALGORITHME boucle_décrémentation
VAR n : réel
DEBUT
   Afficher("Saisir un nombre")
   Lire(n)
   TANTQUE n <> 0 FAIRE
        n ← n – 1
        Afficher(n)
    FINTANTQUE
FIN
```

Exercices

- Écrire un algorithme qui demande à l'utilisateur de saisir un nombre puis qui affiche les 10 nombres suivants

Exercices

```
ALGORITHME boucle_dix_nb_suivants
VAR n, max : réel
DEBUT
   Afficher("Saisir un nombre")
   Lire(n)
   max ← n + 10
   Afficher("Les dix nombres suivants sont :")
   TANTQUE n <= max FAIRE
        n ← n + 1
        Afficher(n)
    FINTANTQUE
FIN
```

Exercices

```
ALGORITHME boucle_dix_nb_suivants
VAR n, i : réel
DEBUT
  Afficher("Saisir un nombre")
  Lire(n)
  i ← 1
  Afficher("Les dix nombres suivants sont :")
  TANTQUE i <= 10 FAIRE
      n ← n + i
      Afficher(n)
      i ← i + 1
   FINTANTQUE
FIN
```

Exercices

- Écrire un algorithme qui demande à l'utilisateur de saisir un nombre entier n puis qui affiche la somme de tous les entiers jusqu'à n (1 + 2 + … + n)

Exe

```
ALGORITHME somme_1_n
VAR n, i, somme : entier
DEBUT
   Afficher("Saisir un nombre")
   Lire(n)
   i ← 1
   somme ← 0
   TANTQUE i <= n FAIRE
        somme ← somme + i
        i ← i + 1
    FINTANTQUE
    Afficher("La somme de 1 à n est : ", somme)
FIN
```

La boucle « POUR »

Boucle **POUR**

- Permet de répéter une instruction un nombre déterminé de fois
  - i.e. le nombre d'itérations est connu à l'avance
- Utilise un **compteur** qui est incrémenté après chaque exécution du bloc d'instructions de la boucle
  - Le programmeur n'a pas à gérer l'incrémentation du compteur

Boucle **POUR**

- Le compteur a une valeur minimale = condition d'entrée dans la boucle
- Le compteur a une valeur maximale = condition de sortie de la boucle
- L'incrémentation du compteur se fait selon un **pas**
  - Par défaut, le pas = 1

Boucle **POUR**

- Structure de la boucle POUR

POUR *compteur* ← *valeur_initiale* ***à*** *valeur_finale* **pas** *valeur_pas*

    instructions

compteur **SUIVANT**

Boucle **POUR**

- Exemple

```
ALGORITHME exemple_boucle_pour
VAR n, i : entier
DEBUT
   n ← 0
  POUR i ← 1 à 10 pas 1
     n ← n + i
  i SUIVANT
FIN
```

Exercices

- Écrire un algorithme qui demande à l'utilisateur de saisir un nombre puis qui affiche les 10 nombres suivants

Exercices

```
ALGORITHME boucle_dix_nb_suivants
VAR n, m : réel
VAR i : entier
DEBUT
  Afficher("Saisir un nombre")
  Lire(n)
  Afficher("Les dix nombres suivants sont :")
  POUR i ← 1 à 11 pas 1
      m ← n + i
      Afficher(m)
   i SUIVANT
FIN
```

Exercices

- Écrire un algorithme qui affiche la table de multiplication du chiffre 9

Exercices

```
ALGORITHME boucle_table_neuf
VAR i: entier
DEBUT
    POUR i ← 1 à 11
        Afficher("9 *" , i , " = " , 9*i)
    i SUIVANT
FIN
```

Exercices

- Écrire un algorithme qui demande à l'utilisateur de saisir un nombre puis affiche la table de multiplication de ce nombre

```
ALGORITHME boucle_table
VAR n: réel
VAR i: entier
DEBUT
    Afficher("Saisir un nombre")
    Lire(n)
    POUR i ← 1 à 11
        Afficher(n, " * " , i , " = " , n*i)
    i SUIVANT
FIN
```

Exercices

- Écrire un algorithme qui demande à l'utilisateur de saisir un nombre entier n puis qui affiche la somme de tous les entiers jusqu'à n (1 + 2 + … + n)

```
ALGORITHME boucle_somme
VAR n, i, somme: entier
DEBUT
    Afficher("Saisir un nombre")
    Lire(n)
    somme ← 0
    POUR i ← 1 à n + 1
        somme ← somme + i
    i SUIVANT
    Afficher(" La somme de 1 à " , n , " est " , somme)
FIN
```

Exercices

- Écrire un algorithme qui demande à l'utilisateur de saisir un nombre entier n puis qui calcule son produit factoriel n! (2 * … * n)

```
ALGORITHME boucle_produit_factoriel
VAR n, i, produit: entier
DEBUT
    Afficher("Saisir un nombre")
    Lire(n)
    somme ← 1
    POUR i ← 2 à n + 1
        somme ← somme * i
    i SUIVANT
    Afficher(" Le produit factoriel de " , n , " est " , produit)
FIN
```

Exercices

- Écrire un algorithme qui demande à l'utilisateur de saisir dix nombres positifs puis qui retourne le nombre le plus grand

```
ALGORITHME boucle_pgn
VAR n, i, pgn: réel
DEBUT
   pgn ← 0
   POUR i ← 1 à 11
        Afficher(" Saisir le nombre numéro ", i)
        Lire(n)
        SI n > pgn ALORS
             pgn ← n
        FINSI
    i SUIVANT
   Afficher(" Le nombre le plus grand est " , pgn)
FIN
```

Exercices

- Écrire un algorithme qui demande à l'utilisateur de saisir dix nombres positifs puis qui retourne le nombre le plus grand et sa position

```
ALGORITHME boucle_pgn
VAR n, i, pgn, pos: réel
DEBUT
   pgn, pos ← 0
   POUR i ← 1 à 11
       Afficher(" Saisir le nombre numéro ", i)
       Lire(n)
       SI n > pgn ALORS
            pgn ← n
            pos ← i
       FINSI
    i SUIVANT
   Afficher(" Le nombre le plus grand est " , pgn, " et sa position est  ", pos)
FIN
```

Exercices

- Écrire un algorithme qui demande à l'utilisateur de saisir dix nombres ~~positifs~~ puis qui retourne le nombre le plus grand, le nombre le plus petit et leurs positions

La boucle « REPETER … JUSQU'A»

Boucle **RÉPÉTER … JUSQU'À**

- Permet de répéter une instruction jusqu'à ce que la condition d'arrêt soit vraie

RÉPÉTER

instructions

JUSQU'À *condition_arrêt*

Boucle **RÉPÉTER … JUSQU'À**

- Permet de répéter une instruction jusqu'à ce que la condition d'arrêt soit vraie

RÉPÉTER

instructions

JUSQU'À *condition_arrêt*

- Utilisée quand le nombre d'itérations n'est pas connu d'avance et qu'il faut exécuter les instructions **au moins une fois**

Boucle **RÉPÉTER … JUSQU'À**

- Exemple

```
ALGORITHME exemple_boucle_répéter
VAR n, i : entier
DEBUT
   n ← 0
  RÉPÉTER
     n ← n + 1
  JUSQU'À n=10
FIN
```

Exercices

- Écrire un algorithme qui demande à l'utilisateur de saisir des nombres positifs puis qui retourne le nombre le plus grand
  - Le nombre des nombres à saisir n'est pas connu à l'avance
  - La saisie s'arrête quand l'utilisateur saisit le nombre -1

```
ALGORITHME boucle_pgn
VAR n, i, pgn, pos : réel
DEBUT
   pgn , i ← 0 , 1
   RÉPÉTER
       Afficher(" Saisir le nombre numéro ", i)
       Lire(n)
       SI n > pgn ALORS
            pgn ← n
            pos ← i
       FINSI
       i ← i +1
   JUSQU'À n = −1
    Afficher(" Le nombre le plus grand est " , pgn, " et sa position est  ", pos)
FIN
```

مدرسة علوم المعلومات
ⵜⵉⵎⵔⵔⴰⵜ ⵏ ⵜⵓⵙⵙⵏⴰⵡⵉⵏ ⵉⵏⴼⵓⵔⵎⴰⵜⵉⵡⵏ
ECOLE DES SCIENCES DE L'INFORMATION
www.esi.ac.ma

Algorithmique

4. Les structures itératives