Université Abdelmalek Essaâdi
Ecole nationale des sciences appliquées d'Al Hoceima
(ENSAH)

Algorithmique

Préparé et présenté par
Mr. Ouazzani Chahdi

Année universitaire: 2018/2019

Structures de contrôle

1-Les structures alternatives

- En algorithmique, il y a deux types d'instructions alternatives qui sont :
  - L'instruction alternative simple.
  - L'instruction alternative complexe.
- Pour mieux comprendre les rôles de ces deux instructions, nous allons étudier deux problèmes :

Problème 1 : On veut élaborer un algorithme ***Division*** qui reçoit deux valeurs réelles et ensuite calcule et affiche le résultat de leur division.

Problème 2 : On veut élaborer un algorithme ***Valeur_Absolue*** qui reçoit une valeur réelle x et ensuite affiche sa valeur absolue (sans utiliser la fonction Abs()).

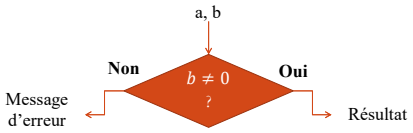
Discussion du problème 1 :

- La première solution qu'on peut proposer est la suivante :

```
Algorithme Division
Variable a, b, D : Réel
Début
Ecrire("Donnez les valeurs de a et b : ")
Lire(a, b)
D ← a/b
Ecrire("Le résultat de la division est :", D)
Fin
```

- Cet algorithme n'est pas complet, car il n'envisage pas le cas où l'utilisateur saisit une valeur nulle pour ***b***.
- Dans ce cas, l'algorithme doit être déjà préparé pour recevoir une valeur nulle pour ***b***, afin de faire intervenir le traitement correspondant.

- Alors, pour calculer a/b l'algorithme ***Division*** doit tester la valeur de b :
  - Si la valeur de b est non nulle, alors l'algorithme calcule a/b et affiche le résultat.
  - Si la valeur de b est nulle, on doit afficher un message d'erreur.
- Selon ce test on a deux traitements possibles de tel sorte que la réalisation de l'un exclue la réalisation de l'autre :



- Un tel test est effectué en utilisant une **structure alternative complexe.**

Discussion du problème 2 :

- Dans cet exemple, on a un seul cas à envisager, c'est le cas où l'utilisateur introduit une valeur négative pour **x**.
- Si tel est le cas alors on doit multiplier la valeur introduite par **-1**.
- Selon ce test, on a un seul traitement possible. Il sera réalisé si et seulement si **x** a une valeur négative.

- Un tel test est effectué en utilisant une **structure alternative simple.**

1.1- Structure alternative simple

Syntaxe :

```
Si Expression_Logique Alors
    Instruction
FinSi
```

Ou

```
Si Expression_Logique Alors
    Instruction1
    Instruction2
    ..
    InstructionN
FinSi
```

Bloc d'instructions (Ensemble d'instructions)

Si Expression_Logique renvoie une valeur *VRAI* alors le bloc d'instructions sera exécuté, sinon il sera ignoré.

- **L'algorithme *Valeur_Absolue*:**

```
Algorithme Valeur_Absolue
Variable x: Réel
Début
Ecrire("Donnez une valeur :")
Lire(x)
Si x < 0 Alors
    x ← -x
FinSi
Ecrire("La valeur absolue est : ", x)
Fin
```

Si **x** est négative, alors on change son signe, sinon on ne fait rien

1.2- Structure alternative complexe

Syntaxe :

```
Si Expression_Logique Alors
  Instruction1
Sinon
  Instruction2
FinSi
```

Ou

```
Si Expression_Logique Alors
  Bloc_Instructions_1
Sinon
  Bloc_Instructions_2
FinSi
```

Si Expression_Logique est VRAI alors **Bloc_Instructions1** sera exécuté et le **Bloc_Instructions2** sera ignoré, sinon le **Bloc_Instructions1** sera ignoré et le **Bloc_Instructions2** sera exécuté.

9 10-Dec-18

- **L'algorithme *Division***

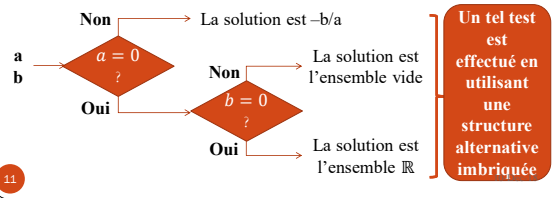
```
Algorithme Division
Variable a, b, D : Réel
Début
Ecrire("Donnez la valeur de a :")
Lire(a)
Ecrire("Donnez la valeur de b :")
Lire(b)
Si b = 0 Alors
  Ecrire("Division par 0 impossible")
Sinon
  D ← a/b
  Ecrire("Le résultat de la division est:",D)
FinSi
Fin
```

10 10-Dec-18

1.3- Imbrication des structures alternatives

Problème 1 : On veut élaborer un algorithme **Equation** qui résout une équation de type $ax + b = 0$.

- La solution de cette équation est **–b/a**, mais elle est valable que si a est différent de 0.
- Maintenant si la valeur de a est nulle, alors l'équation est ramené à l'égalité $b = 0$, dans ce cas la solution dépendra de **b** :
  - Si la valeur de b est nulle, alors l'ensemble de solution est $\mathbb{R}$
  - Sinon la solution est l'ensemble vide.



11

- **Ecrivons l'algorithme *Equation***

```
Algorithme Equation
Variable a, b, x : Réel
Début
Ecrire("Donnez respectivement les valeurs de a
  et b : ")
Lire(a, b)
  Si a = 0 Alors
    Si b = 0 Alors
      Ecrire("La solution est l'ensemble R")
    Sinon
      Ecrire("La solution est l'ensemble vide")
    FinSi
  Sinon
      x ← -b/a
      Ecrire("La solution est : ", x)
  FinSi
Fin
```

12 10-Dec-18

Problème 2 : On veut écrire un algorithme **Mention** qui permet de savoir la mention d'un étudiant à partir de sa note en suivant le schéma ci-dessous :

- Note < 10 : mauvais
- 10 <= Note < 12 : passable
- 12 <= Note < 14 : assez bien
- 14 <= Note < 16 : bien
- 16 <= Note <= 20 : très bien

❖ En utilisant la notion des instructions imbriquées, on a abouti à l'algorithme suivant :

13 10-Dec-18

```
Algorithme Mention
Variable Note : Réel
Début
  Ecrire("Donnez la note de l'étudiant :")
  Lire(Note)
  Si Note < 10 Alors
    Ecrire("Mauvais")
  Sinon
    Si Note < 12 Alors
      Ecrire("Passable")
    Sinon
      Si Note < 14 Alors
        Ecrire("Assez bien")
      Sinon
        Si Note < 16 Alors
          Ecrire("bien")
        Sinon
          Ecrire("Très bien")
        FinSi
      FinSi
    FinSi
  FinSi
Fin
```

14 10-Dec-18

❖ La syntaxe utilisée dans cet algorithme est un peu longue et peu lisible surtout si on ajoute encore la mention Excellent (18<Note<=20).

❖ Alors le pseudo-code algorithmique admet une simplification supplémentaire avec la syntaxe suivante :

```
Si Expression_Logique1 Alors
  Bloc_Instructions1
Sinon Si Expression_Logique2 Alors
  Bloc_Instructions2
…
…
Sinon Si Expression_LogiqueN Alors
  Bloc_InstructionsN

[Sinon  Bloc_Instructions_Final]
FinSi
```

15 10-Dec-18

❖ **Ainsi l'algorithme *Mention* devient :**

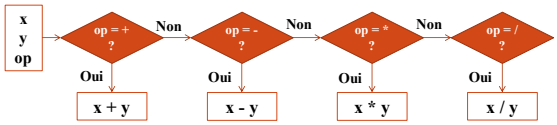
```
Algorithme Mention
Variable Note : Réel
Début
Ecrire("Donnez la note de l'étudiant :")
Lire(Note)
  Si Note < 10 Alors
    Ecrire("Mauvais")
  Sinon Si Note < 12 Alors
    Ecrire("Passable")
  Sinon Si Note < 14 Alors
     Ecrire("Assez bien")
  Sinon Si Note < 16 Alors
     Ecrire("Bien")
 Sinon
     Ecrire("Très bien")
  FinSi
Fin
```

16 10-Dec-18

2- Structure de choix multiple

Problème : On veut élaborer un algorithme ***Arithmétiques*** qui réalise au choix de l'utilisateur l'addition, la soustraction, la multiplication ou la division de deux valeurs réelles ***x*** et ***y*** saisies au clavier.

- Pour que l'algorithme soit capable de savoir l'opération arithmétique qu'il doit faire, l'utilisateur doit lui donner cette information.
- Selon cette information, l'algorithme va sélectionner l'opération correspondante.



- Un tel test est effectué en utilisant une structure de choix multiple.

Syntaxe:

```
Cas (Expression) Vaut
  Val1  : Bloc_Instructions1
  Val2  : Bloc_Instructions2
  ...
  ValN  : Bloc_InstructionsN
  [Autre : Bloc_Instructions]
FinCas
```

Où Vali (i=1,2,….,N) est une valeur unique, une liste de valeurs séparées par des virgules ou un intervalle de valeurs.

- Si Expression vaut une valeur ***Vali*** alors le bloc d'instruction ***i*** qui sera exécuté et tous les autres blocs seront ignorés.
- Si Expression n'a aucune valeur parmi ***Val1, Val2 ,…..,Valn***, c'est le bloc d'instruction après la rubrique ***Autre*** qui sera exécuté et les autres blocs seront ignorés.

- **Ecrivons l'algorithme *Arithmétique* :**

```
Algorithme Arithmétique
Variable x, y, résultat : Réel
         op: Caractère
Début
  Ecrire("+ : Addition")
  Ecrire("- : Soustraction")
  Ecrire("* : Multiplication")
  Ecrire("/ : Division")
  Ecrire("Donner deux valeurs :")
  Lire(x, y)
  Ecrire("Donner votre choix +, -, * ou / :")
  Lire(op)
```



```
Cas op Vaut
  '+' : résultat ← x + y
        Ecrire("Addition = ", résultat)
  '-' : résultat ← x - y
        Ecrire("Soustraction= ", résultat)
  '*' : résultat ← x * y
        Ecrire("Multiplication = ", résultat)
  '/' : Si y <> 0 Alors
          résultat ← x/y
          Ecrire("Division = ", résultat)
        Sinon
          Ecrire("La division par 0 est
          impossible")
        Finsi
  Autre : Ecrire("Choix incorrecte")
FinCas
Fin
```

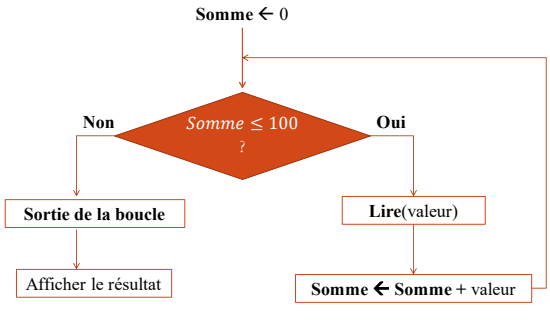
3-Les structures itératives

- En algorithmique une structure itérative est tout simplement **une boucle**, c'est-à-dire une **répétition d'instructions**.
- On l'utilise souvent quand on doit **exercer plusieurs fois** le **même traitement** sur un même objet.
- Mais son réel intérêt réside dans le fait que l'on peut modifier, à chaque répétition, les objets sur lesquels s'exerce l'action répétée.
- Alors dans une boucle on a :
  - Une instruction ou un ensemble d'instructions à répéter.
  - Cette répétition doit avoir un arrêt.
  - L'arrêt de cette répétition dépend d'une condition.
  - Cette condition est appelé **condition d'arrêt**.

3.1- La boucle TantQue … Faire

Problème : on veut élaborer un algorithme ***Somme_100*** qui calcule la somme d'une série de nombres entiers saisies au clavier jusqu'à ce que cette somme dépasse la valeur 100.

- L'exécution de cet algorithme déroulera comme suit :
  1. On initialise une variable ***Somme*** par 0. cette variable va contenir le résultat de la somme.
  2. On s'assure que la valeur de ***Somme*** est inférieur ou égale à 100. si ce n'est pas le cas on arrête, sinon :
     a. On invite l'utilisateur à saisir la première valeur.
     b. On fait **Somme ← Somme + valeur**
     c. On recommence à l'étape 2.
  3. Après l'arrêt de la boucle on affiche la valeur de ***Somme***.
- Ces étapes peuvent être schématisé en utilisant l'organigramme suivant :



Simulation

Syntaxe :

```
TantQue (Expression_Logique) Faire
    Instruction ou Bloc_Instructions
FinTantQue
```

Principe :

1. L'algorithme arrive sur la ligne du ***TantQue***. Il examine alors la valeur de Expression_Logique .
2. Si cette valeur est *VRAI*, l'algorithme exécute les instructions qui suivent, jusqu'à ce qu'il rencontre la ligne ***FinTantQue***.
3. Il retourne ensuite sur la ligne du ***TantQue***,
4. Il procède au même examen, et ainsi de suite.
5. On ne s'arrête que lorsque l' Expression_Logique prend la valeur *FAUX*.

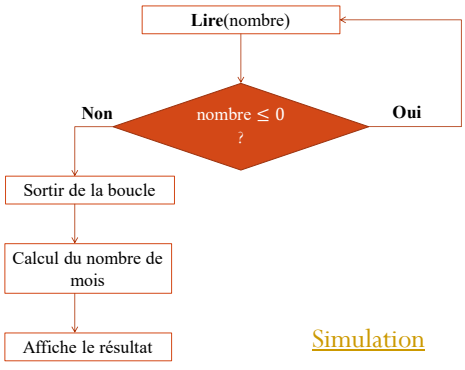
Ecrivons l'algorithme *Somme_100*

```
Algorithme Somme_100
Variable Somme, valeur : Entier
Début
Somme ← 0
TantQue (Somme <= 100) Faire
  Ecrire("Donnez une valeur entière : ")
  Lire(valeur)
  Somme ← Somme + valeur
FinTantQue
Ecrire("La somme est ", Somme)
Fin
```


3.2- La boucle Répéter ... Jusqu'à

Problème : On veut élaborer un algorithme ***Nombre_Mois*** qui demande à l'utilisateur son âge en nombre d'années et lui affiche le nombre de mois. Cet algorithme doit obliger l'utilisateur à saisir une valeur valide(supérieur strictement à 0).

- L'exécution de cet algorithme déroulera comme suit :
  1. On invite l'utilisateur à saisir un nombre d'années
  2. On s'assure que le nombre saisi est strictement positif. si ce n'est pas le cas on recommence à l'étape 1, sinon on arrête la boucle.
  3. Après l'arrêt de la boucle, on calcule et on affiche le résultat.
- Ces étapes peuvent être schématisé en utilisant l'organigramme suivant :



Simulation

Syntaxe :

```
Répéter
    Instruction ou bloc_instructions
Jusqu'à (Expression_Logique)
```

Principe :

Toutes les instructions écrites entre ***Répéter*** et ***jusqu'à(…)*** sont exécutées ***au moins une fois*** et leur exécution sera répétée jusqu'à ce que la valeur de Expression_Logique soit ***VRAI***.

Remarque :

Certains langages ne proposent pas de structure ***Répéter…Jusqu'à***, par contre il peuvent présenter une structure ***Répéter…TantQue***. Il suffit uniquement d'inverser l'expression Expression_Logique.

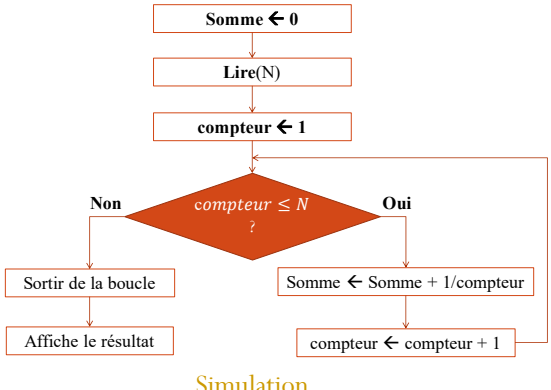
Ecrivons l'algorithme *Nombre_Mois*

```
Algorithme Nombre_Mois
Variable nbr_années, nbr_mois: Entier
Début
Répéter
  Ecrire("Donnez un nombre d'année valide :")
  Lire(nbr_annees)
Jusqu'à(nbr_années > 0)
Nbr_mois = nbr_années*12
Ecrire("Le nombre de mois est ", nbr_mois)
Fin
```


3.3- La boucle Pour... Jusqu'à ...

Problème : On veut élaborer un algorithme ***Somme_N*** qui calcule la somme $1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{N}$, N étant un entier positif saisi par l'utilisateur.

- L'exécution de cet algorithme déroulera comme suit :
  1. On initialise une variable Somme à 0. cette variable va contenir le résultat de la somme.
  2. On invite l'utilisateur à saisir un nombre positif N.
  3. On initialise un compteur à 1.
  4. On vérifie si la valeur de celui-ci ne dépasse pas la valeur de N. si c'est le cas, on arrête, sinon :
     a. On fait Somme ← Somme +1/compteur
     b. On fait compteur ← compteur + 1
     c. On passe à l'étape 4.
  5. Après l'arrêt de la boucle on affiche la valeur de Somme.
- Ces étapes peuvent être schématisé par l'organigramme suivant :



Simulation

Syntaxe :

```
Pour i Allant De Début jusqu'à Fin [PAS p] Faire
      Instruction ou Bloc_Instructions
  FinPour
```

Principe :

1. On initialise un compteur **i** par une valeur initiale ***Début***.
2. On teste si la valeur de ce compteur n'a pas dépassé la valeur ***Fin***. Si c'est le cas on sort de la boucle, sinon :
3. On exécute le bloc d'instructions.
4. On incrémente la valeur de i:
   - Si [Pas p] est absent, on incrémente la valeur de **i** par **1**(i ← i + 1), sinon :
   - **i ← i + p** : où **p** représente le **pas d'incrémentation**
5. On recommence à l'étape 2

Ecrivons l'algorithme *Somme_N*

```
Algorithme Somme_N
Variable N, compteur: Entier
         Somme : Réel
Début
Somme ← 0
Ecrire(" Donner une valeur entière positive :")
Lire(N)
Pour compteur allant de 1 jusqu'à N Faire
  Somme ← Somme + 1/compteur
FinPour
Ecrire("La somme est ", Somme)
Fin
```

33 10-Dec-18

3.3- Critères de choix d'une boucle selon le domaine d'utilisation

3.3.1-La boucle TantQue...Faire

- On l'utilise **quand on ne sait pas à l'avance** le nombre de fois que l'on doit répéter le même traitement.
- On l'utilise surtout pour la lecture d'un fichier, la recherche d'un élément dans un tableau, les calculs scientifiques.

3.3.2-La boucle Répéter...Jusqu'à

- On l'utilise **quand on ne sait pas à l'avance** le nombre de fois que l'on doit répéter le même traitement.
- Avec cette boucle on effectue au moins une fois l'instruction à répéter.

3.3.3-La boucle Pour...Jusqu'à

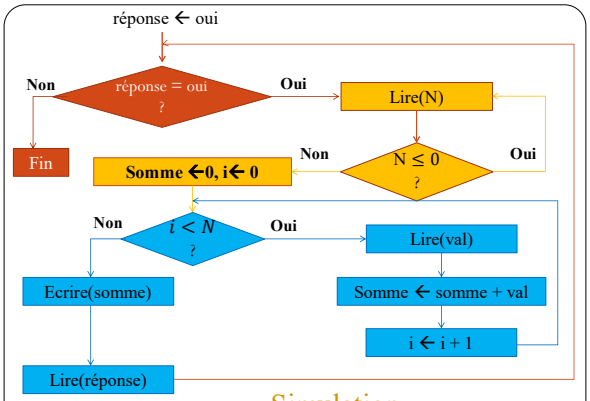
- On l'utilise quand on sait à l'avance le nombre de fois que l'on doit répéter le même traitement.
- C'est le cas des valeurs consécutives entre deux bornes données.

34 10-Dec-18

3.3- Imbrication des structures itératives

- Considérons le problème suivant :
  - On veut élaborer un algorithme ***Plusieurs_Sommes*** qui permet de calculer la somme d'une série de nombres saisis au clavier.
  - On veut aussi que cet algorithme offre à l'utilisateur la possibilité de répéter ce processus autant qu'il veut.
  - Et enfin, on doit s'assurer que l'utilisateur saisi une valeur strictement positive pour la taille de la série en question.
- Alors cet algorithme doit :
  - Lire la taille N de la série.
  - Vérifier que N > 0, sinon on oblige l'utilisateur à saisir une valeur valide.
  - Lire les nombres de cette série et en même temps calculer la somme.
  - Afficher le résultat de la somme.
  - Proposer à l'utilisateur le choix de répéter le processus ou l'arrêter.
  - L'algorithme va décider selon la réponse de l'utilisateur.

35 10-Dec-18



Simulation

36 10-Dec-18

```
Algorithme Plusieurs_Sommes
Variable somme, x: Réel
         N, i: Entier
         réponse : Caractère
Début
  réponse ← 'O'
  somme ← 0
  TantQque(réponse = 'O' Ou réponse = 'o')
    Répéter
      Ecrire("Donnez N :")
      Lire(N)
    Jusqu'à(N > 0)
    Pour i allant de 0 Jusqu'à N-1 Faire
       Ecrire("Donnez la valeur ", i+1)
       Lire(x)
       somme ← somme + x
    FinPour
    Ecrire("La somme est : ", somme)
    Ecrire("Voulez vous répéter le calcul (O/N): ")
    Lire(réponse)
  FinTantQque
Fin
```

37 10-Dec-18