Université Félix Houphouët-Boigny
(UFHB)

Cours d'algorithme avancée
L3-Info

Enseignant :

Dr GBAME Gbede Sylvain

Assistant, enseignant chercheur à l'UFHB

gbamegbedesylvain@gmail.com

MODULE : Algorithme avancée

PLAN

Chapitre 3

Les structures répétitives

Chapitre 3 : Les structures répétitives

3.Introduction

Certains algorithmes du chapitre précédent présentent des failles. La solution à ces failles consiste à contraindre l'utilisateur à saisir des valeurs appartenant à une plage de valeurs.

En effet, on doit :

- soit lui demander de saisir une nouvelle valeur tant que l'actuelle n'est pas valide,
- soit répéter le refus de toute valeur invalide jusqu'à ce qu'il saisisse une valeur valide.

Les structures répétitives permettent de réaliser de telles contraintes.

Une structure répétitive permet d'exécuter un même bloc d'instructions plusieurs fois de suite.

On la désigne aussi par les termes ***structure de répétition*** ou ***structure itérative*** ou ***structure de boucle*** ou ***boucle***.

Les structures répétitives sont aussi **des structures de contrôle**.

Il existe trois catégories de structures de boucle :

- la structure **Tant Que** ;
- la structure **Répéter Jusqu'à** (appelée tout simplement structure Répéter) ;
- la structure **Pour**.

Ces structures jouent presque le même rôle mais elles ne s'utilisent pas de la même façon.

Chapitre 3 : Les structures répétitives

3.1 La structure *Tant que*

Elle a pour format général :

TantQue < condition > **Faire**
< action(s) >
FinTantQue

Comment fonctionne cette structure ?

La structure ***Tant Que*** permet d'exécuter une ou plusieurs actions plusieurs fois de suite tant que la condition spécifiée est vérifiée.

La condition est évaluée **avant** la première exécution du bloc d'actions définies.
Lorsque l'évaluation de la condition produit la valeur :

- ***Vrai*** : le bloc d'actions est exécuté, et la condition est évaluée à nouveau ;
- ***Faux*** : aucune action de la boucle ***Tant Que*** n'est exécutée, et l'instruction qui vient immédiatement après ***FinTantQue*** est exécutée. (On sort de la boucle.)

Chapitre 3 : Les structures répetitives

3.1 La structure *Tant que*

Puisque la condition est évaluée avant l'entrée dans la boucle, il peut alors arriver que le bloc d'actions ne soit pas exécuté. Cela se produit lorsque la première évaluation de la condition produit la valeur **Faux**.
Par conséquent, les actions définies dans la boucle ***Tant Que*** peuvent être exécutées **0** ou **1** ou plusieurs fois.
De plus, il faut qu'au moins une variable de l'expression d'évaluation soit susceptible de modification à l'intérieur de la boucle pour que l'on puisse en sortir.

Si la condition qui régit la boucle ne prend jamais la valeur **Faux** (donc si la condition prend toujours la valeur Vrai), les actions seront répétées indéfiniment. On aura alors un cas de "boucle infinie". (On ne sort jamais de la boucle)
Pour éviter ce genre d'erreur, il faut toujours rechercher la condition d'arrêt de la boucle, et exprimer sa proposition contraire.

Chapitre 3 : Les structures répetitives

3.1 La structure *Tant que*

Exercice d'application 1 : Saisie d'un nombre selon un critère

Écrire un algorithme qui demande la saisie d'un entier strictement négatif. L'algorithme valide la saisie si elle est correcte, sinon il invite l'utilisateur à recommencer.

Exercice d'application 2 : Affichage d'une suite de nombres selon un critère

Écrire un algorithme qui affiche les entiers naturels de n à 1, n étant un entier naturel strictement supérieur à 1, saisi au clavier.

```
Algorithme saisieNombreNegatif
// Déclaration de la variable
   Var entier <- 0

Début
// Boucle tant que l'entier n'est pas strictement négatif
   Tant que entier >= 0 faire
      Écrire ("Veuillez entrer un entier strictement négatif :   ")
      Lire (entier)

      // Si la saisie est incorrecte, demander à recommencer
      Si entier >= 0 Alors
         Écrire "Erreur : L'entier doit être strictement négatif. Veuillez recommencer."
      Fin Si
   Fin Tant que

   // Si la saisie est correcte
   Écrire "Saisie correcte : ", entier
Fin
```

Chapitre 3 : Les structures répetitives

3.1 La structure *Tant que*

Explication :

- La variable ***entier*** est initialisée à 0.

- La boucle "***Tant que***" continue tant que l'entier saisi est supérieur ou égal à 0.

- Si l'utilisateur entre un ***entier*** positif ou nul, un message d'erreur s'affiche, et il doit recommencer la saisie.

- Une fois un entier strictement négatif saisi, l'algorithme affiche que la saisie est correcte et sort de la boucle.

3.2 La structure *Répéter*

Elle a pour format général :

Répéter
< action(s) >
Jusqu'à < condition >

La structure ***Répéter*** permet d'exécuter une ou plusieurs actions plusieurs fois de suite jusqu'à ce que la condition spécifiée soit vérifiée.

La condition est évaluée ***après*** la première exécution du bloc d'actions définies.
Lorsque l'évaluation de la condition produit la valeur :

- **Faux** : le bloc d'actions est exécuté, et la condition est évaluée à nouveau
- **Vrai** : aucune action de la boucle ***Répéter*** n'est exécutée, et l'instruction qui vient immédiatement après le mot clé ***Jusqu'à*** est exécutée. (On sort de la boucle)

3.2 La structure *Répéter*

Puisque la condition est évaluée ***après l'exécution du bloc d'actions*** définies dans la boucle, alors le *bloc d'actions est exécuté au moins une fois*.
Donc les actions définies dans la boucle Répéter seront exécutées ***une*** ou ***plusieurs*** fois. De plus, il faut qu'au moins une variable de l'expression d'évaluation soit susceptible de modification à l'intérieur de la boucle pour qu'on puisse en sortir.

Si la condition qui régit la boucle ne prend jamais la valeur **Vrai** (donc si la condition prend toujours la valeur **Faux**), les actions sont répétées indéfiniment. On aurait alors un cas de **"boucle infinie"**.
Pour éviter ce genre d'erreurs, il faut toujours **rechercher la condition de continuité** de la boucle, et ***exprimer sa proposition contraire***.

3.2 La structure *Répéter*

Exercice d'application 1 : Affichage d'un intervalle de nombres selon un critère
Écrire un algorithme qui affiche les entiers naturels de n à 1, n étant un entier naturel strictement supérieur à 1, saisi au clavier.

Exercice d'application 2 : Résoudre plusieurs fois une équation du premier degré
Écrire un algorithme qui permet de résoudre une équation du premier degré, autant de fois que l'utilisateur le souhaite.

```
Algorithme EquationSecondDegre
// Déclaration des variables
   Var a, b, x : réel
   Var continuer : chaine de caractère

Début

   // Boucle principale pour résoudre l'équation plusieurs fois
   Répéter
      // Saisie des coefficients a et b
      Écrire ("Résolution de l'équation ax + b = 0 ")
      Écrire ("Veuillez entrer la valeur de a (non nul) :  " )
      Lire a

      Tant que a = 0 faire
         Écrire ("a ne doit pas être égal à 0. Veuillez entrer une autre valeur de a : " )
         Lire ( a)
      Fin Tant que

      Écrire  ("Veuillez entrer la valeur de b :  " )
      Lire b

      // Calcul de la solution
      x <- -b / a

      // Affichage de la solution
      Écrire  ("La solution de l'équation est : x = ", x )

      // Demander à l'utilisateur s'il veut résoudre une autre équation
      Écrire ("Voulez-vous résoudre une autre équation ? (O/N) :  ")
      Lire ( continuer )
   Jusqu'à continuer = 'N' ou continuer = 'n'

Fin
```

Explication :

1.L'algorithme demande à l'utilisateur de saisir les coefficients **a** et **b** de l'équation ax + b = 0.

2. Si **a** est nul, l'algorithme demande de ressaisir une valeur différente de zéro (car une équation de premier degré nécessite a ≠ 0

3. Il résout l'équation en calculant x = −b/ a.

4. Ensuite, l'utilisateur est invité à décider s'il veut résoudre une autre équation ou non. La boucle se répète tant que l'utilisateur saisit "O" ou "o" pour continuer. Ainsi, l'algorithme permet de résoudre autant d'équations du premier degré que l'utilisateur le souhaite.

3. 3 La structure *Pour*

Elle a pour format général :

Pour < identificateur > ← < valeur initiale > **à** < valeur finale > [Pas de < incrément >] **Faire**
    < action(s) >
FinPour

Avant d'étudier le fonctionnement de cette structure, il faut savoir que :

- < **identificateur** > est le nom d'une variable de type entier ou réel ;

- < **valeur initiale** > et < **valeur finale** > sont des **constantes** ou des **variables** ou des **expressions** du même type que < **identificateur** >, donc de type entier ou réel ;

- < **valeur initiale** > et < **valeur finale** > constituent les bornes de l'intervalle d'entiers ou de réels dans lequel < **identificateur** > va prendre ses valeurs;

3.3 La structure *Pour*

- Pas de < **incrément** > est une valeur entière ou réelle positive ou négative prédéfinie ajoutée à la valeur de < **identificateur** > à chaque exécution de < **action(s)** >. On peut simplement dire Pas ou incrément;

- < **action(s)** > peut être une instruction, un bloc d'**instructions** ou un **algorithme**.

Comment fonctionne la structure Pour ?

La structure Pour permet d'exécuter un bloc d'actions un nombre connu de fois.
< **identificateur** > reçoit la valeur de < **valeur initiale** >. La valeur de < **identificateur** > va progresser en fonction du pas (c'est-à-dire en fonction de l'incrément) jusqu'à atteindre la valeur de < **valeur finale** >. C'est la condition de continuité de la boucle Pour.
Chaque fois que < **identificateur** > change de valeur et progresse, < **action(s)** > est exécuté.

3.3 La structure *Pour*

Exemples :

- *valeur initiale*, *valeur finale* et *incrément* valent respectivement 1 ; 5 et 1.

< identificateur > progressera de 1 en 1, et prendra les valeurs 1 ; 2 ; 3 ; 4 et 5.
< action(s) > sera exécuté 5 fois.

- *valeur initiale*, *valeur finale* et *incrément* valent respectivement 2 ; 12 et 2.

< identificateur > progressera de 2 en 2, et prendra les valeurs 2 ; 4 ; 6 ; 8 ; 10 et 12.
< action(s) > sera exécuté 6 fois.

- *valeur initiale*, *valeur finale* et *incrément* valent respectivement 5 ; 0 et **-1.**

< identificateur > progressera de **-1** en **-1**, et prendra les valeurs 5 ; 4 ; 3 ; 2 ; 1 et 0.
< action(s) > sera exécuté 6 fois.

3.3 La structure *Pour*

- *valeur initiale*, *valeur finale* et *incrément* valent respectivement **-10** ; 2 et 2.

< identificateur > progressera de 2 en 2, et prendra les valeurs -10 ; -8 ; -6 ; -4 ; -2 ; 0 et 2.

< action(s) > est exécuté 7 fois.

- *valeur initiale*, *valeur finale* et *incrément* valent respectivement 2; **-10** et **-2**.

< identificateur > progressera de -2 en -2, et prendra les valeurs 2 ; 0 ; -2 ; -4 ; -6 ; -8 et -10.

< action(s) > est exécuté 7 fois.

3.3 La structure *Pour*

Cependant

- Si **valeur initiale** = **valeur finale** et **incrément** = **1**, alors le bloc d'actions définies sera exécuté une seule fois.

- Si **valeur initiale** > **valeur finale** et **incrément** > **0**, alors le bloc d'actions définies ne sera pas exécuté.

- Si **valeur initiale** < **valeur finale** et **incrément < 0**, alors le bloc d'actions définies ne sera pas exécuté.

La structure Pour s'exécute selon l'algorithme ci-dessous :

Chapitre 3 : Les structures répetitives

3.3 La structure *Pour*

1. < **identificateur** > reçoit la valeur de < valeur initiale >
2. < **identificateur** > vérifie les conditions de continuité
   a. Si **Vrai** :
      i. exécuter < **action(s)** >
      ii. exécuter < **identificateur** > ← < **identificateur** > + < **incrément** >
      iii. revenir au point 2
   b. Si **Faux** : sortir de la boucle

Cas particulier : l'incrément vaut 1

Lorsque l'incrément vaut 1, la structure ***Pour*** devient :

Pour < identificateur > ← < valeur initiale > **à** < valeur finale > **Faire**
    < action(s) >
FinPour

3.3 La structure *Pour*

Exercice d'application 1 : Affichage d'un intervalle de nombres selon un critère
Écrire un algorithme qui affiche les entiers naturels de 1 à n, n étant un entier naturel strictement supérieur à 1, saisi au clavier.

Exercice d'application 2 : Génération de la table de multiplication de 9
Écrire un algorithme qui génère la table de multiplication de 9.

Exercice d'application 3 : Génération d'une table de multiplication quelconque
Écrire un algorithme qui demande un entier naturel n à l'utilisateur, puis qui génère la table de multiplication de ce nombre.

3.4 Imbrication de structures de contrôle

Selon le problème à résoudre, on peut être amené à imbriquer des structures de contrôle.
En effet, on peut imbriquer :

- les structures conditionnelles entre elles ;
- les structures itératives entre elles ;
- une structure itérative dans une structure conditionnelle ;
- une structure conditionnelle dans une structure itérative ;

Soit l'énoncé ci-dessous :

Exercice d'application : Génération de tables de multiplication

Écrire un algorithme qui génère les dix tables de multiplication.

3.4 Imbrication de structures de contrôle

EXERCICES D'ENTRAINEMENT

Avec les structures de contrôle (structures conditionnelles et les boucles), on peut désormais écrire des algorithmes plus "intelligents".

La résolution des exercices proposés peut requérir des connaissances dans divers domaines, notamment en mathématiques, en comptabilité et en linguistique. Alors quelques recherches pourraient être utiles.

Pour tous les algorithmes de cette partie et les prochains à écrire, il est recommandé d'effectuer si nécessaire et si possible des contrôles de saisie.

Pour les exercices traitant des chaînes de caractères contenant des caractères alphabétiques, sauf indication contraire, on considérera que ces caractères sont des lettres minuscules sans tenir compte des signes diacritiques (accents, trémas, cédilles).

On admettra qu'un mot est une chaîne de caractères alphabétiques ou non ayant un sens ou non.

Maintenant, testons nos connaissances !

FIN DU COURS