Algorithmique et Programmation :
Chap 5 : Structures itératives

E. M. Souidi

Faculté des Sciences - Rabat
SVI4 –STU4 2013-14

Introduction

Les boucles sont aussi appelées structures répétitives ou itératives.

Introduction

Les boucles sont aussi appelées structures répétitives ou itératives.
Il arrive très souvent qu'une partie d'un problème soit solutionnée en répétant une même séquence d'instructions un certain nombre de fois mais avec des paramètres différents.

Par exemple :

Par exemple :
- Calculer les termes d'une suite u_n, seul le paramètre n qui change.

Par exemple :
- Calculer les termes d'une suite u_n, seul le paramètre n qui change.
- Un robot qui monte des escaliers.

Par exemple :
- Calculer les termes d'une suite u_n, seul le paramètre n qui change.
- Un robot qui monte des escaliers.
- Attribution de mentions aux étudiants.

Deux solutions :

Deux solutions :
1) Écrire cette séquence d'instructions, pour chaque cas, autant de fois que nécessaire, encore faut-il connaitre le nombre de fois.

Deux solutions :
1) Écrire cette séquence d'instructions, pour chaque cas, autant de fois que nécessaire, encore faut-il connaitre le nombre de fois.
2) Utiliser une structure itérative, valable même si on ne connaît pas le nombre de fois à effectuer.

Une itération (ou boucle) consiste à exécuter un bloc d'instructions un certains nombre de fois avec différents paramètres.

Une itération (ou boucle) consiste à exécuter un bloc d'instructions un certains nombre de fois avec différents paramètres.
Le nombre d'itération (ou de passage dans la boucle) peut être connu ou non.

Une variable appelée compteur sert à compter le nombre d'itération.

Une variable appelée compteur sert à compter le nombre d'itération.
Ce compteur est initialisée avant la première itération,

incrémentée à chaque itération jusqu'à ce qu'il atteint la valeur maximale pré-définie ou calculée.

incrémentée à chaque itération jusqu'à ce qu'il atteint la valeur maximale pré-définie ou calculée.
L'exécution du programme se poursuit à partir de la première instruction qui suit immédiatement la boucle .

Instruction `TantQue`

Est utilisée quand on connait pas le nombre d'itérations. Elle s'écrit en pseudo-code :

Instruction TantQue

Est utilisée quand on connait pas le nombre d'itérations. Elle s'écrit en pseudo-code :

```
TantQue (expression logique)
        (séquence d'instructions)
FinTantQue
```

Tant que (expression logique) est VRAI (séquence d'instructions) est exécutée.

Tant que (expression logique) est VRAI (séquence d'instructions) est exécutée.
Quand (expression logique) devient FAUX, l'exécution passe après l'instruction `FinTantQue`.

Tant que (expression logique) est VRAI (séquence d'instructions) est exécutée.
Quand (expression logique) devient FAUX, l'exécution passe après l'instruction `FinTantQue`.

Exemple

```
Taper O/N
Lire(Frape)
TantQue (Frape <> O et Frape <> N)
        Ecrire (Taper O/N)
FinTantQue
Si Frape = "O"
        Instructions1
Sinon
        Instructions2
```

Instruction `Pour`

Parfois on connait le nombre exact d'exécutions de la boucle.

Instruction `Pour`

Parfois on connait le nombre exact d'exécutions de la boucle.
Dans ce cas, on peut utiliser l'instruction `Pour`.

Instruction `Pour`

Parfois on connait le nombre exact d'exécutions de la boucle.
Dans ce cas, on peut utiliser l'instruction `Pour`.
En pseudo-code cette instruction s'écrit :

Instruction `Pour`

Parfois on connait le nombre exact d'exécutions de la boucle.
Dans ce cas, on peut utiliser l'instruction `Pour`.
En pseudo-code cette instruction s'écrit :

Boucle Pour :

`Pour` *i* ← Val_Init à *i* ← Val_fin [par pas] `faire`
(séquence d'instructions)
`FinPour`

(séquence d'instructions) est exécutée pour chaque valeur de *i* allant de valeur_Init à valeur_fin.

(séquence d'instructions) est exécutée pour chaque valeur de *i* allant de valeur_Init à valeur_fin.
Puis l'exécution passe aux instructions se trouvant après `FinPour`.

(séquence d'instructions) est exécutée pour chaque valeur de *i* allant de valeur_Init à valeur_fin.
Puis l'exécution passe aux instructions se trouvant après `FinPour`.
Par défaut le pas est égale à un. Si on veut préciser un pas différent de 1, on utilise `[par pas]` .

Le pas négatif est aussi permis, mais dans ce cas valeur_Init doit être supérieur à valeur_fin.

Example

```
Pour i ← 1 à i ← 20 pas ← 2 faire
Ecrire i × i
FinPour
```

Une autre utilisation de `Pour`

En pseudo-code elle s'écrit :

Une autre utilisation de `Pour`

En pseudo-code elle s'écrit :

Boucle Pour :

`Pour` var `dans` [ensemble ou liste] `faire`
    (séquence d'instructions)
`Finpour`

La séquence d'instructions est exécutée pour chaque élément dans l'ensemble ou la liste.

La séquence d'instructions est exécutée pour chaque élément dans l'ensemble ou la liste.
Cet ensemble peut être dynamique, c'est à dire il peut être modifié au cours des itérations.

Instruction `Répéter ... TantQue`

Il arrive que l'on soit certain que la boucle s'exécutera au moins une fois.

Instruction `Répéter ... TantQue`

Il arrive que l'on soit certain que la boucle s'exécutera au moins une fois.
Dans ce cas, on privilégie l'emploi de la boucle `Répéter ... TantQue`.

Instruction `Répéter ... TantQue`

Il arrive que l'on soit certain que la boucle s'exécutera au moins une fois.
Dans ce cas, on privilégie l'emploi de la boucle `Répéter ... TantQue`.
En pseudo-code cette instruction s'écrit :

Instruction `Répéter ... TantQue`

Il arrive que l'on soit certain que la boucle s'exécutera au moins une fois.
Dans ce cas, on privilégie l'emploi de la boucle `Répéter ... TantQue`.
En pseudo-code cette instruction s'écrit :

Répéter ... TantQue :

```
Répéter
        (séquence d'instructions)
TantQue
        (expression logique)
```

1) (séquence d'instructions) est exécutée.

1) (séquence d'instructions) est exécutée.
2) (expression logique) est évaluée :

1) (séquence d'instructions) est exécutée.
2) (expression logique) est évaluée :
    - Si elle est à VRAI, on retourne au point 1

1) (séquence d'instructions) est exécutée.
2) (expression logique) est évaluée :
   - Si elle est à VRAI, on retourne au point 1
   - Si elle est à FAUX, on exécute l'instruction qui suit

`TantQue`.

Exemple

```
...
Répéter Ecrire "Voulez vous sauvegarder: O/N"
TantQue rep  <>  O ET Rep  <>  N
...
```

Remarques

Attention aux boucles infinies : Elle consiste à écrire une boucle dans laquelle le booléen ne devient jamais FAUX. Dans ce cas l'ordinateur exécute la boucle sans jamais s'arrêter !

Remarques

Attention aux boucles infinies : Elle consiste à écrire une boucle dans laquelle le booléen ne devient jamais FAUX. Dans ce cas l'ordinateur exécute la boucle sans jamais s'arrêter !
Il faut donc toujours s'assurer que la boucle est finie.

La structure `Pour` ... `FinPour` n'est pas indispensable ; on peut bien programmer toutes les situations de boucles uniquement avec la structure `TantQue`...`FinTantQue`.

La structure `Pour ... FinPour` n'est pas indispensable ; on peut bien programmer toutes les situations de boucles uniquement avec la structure `TantQue...FinTantQue`.
La structure `Pour ... FinPour` est un cas particulier de `TantQue ...FinTantQue` : celui où on peut dénombrer à l'avance le nombre d'itérations nécessaires.

Le seul intérêt de la forme `Pour` ... `FinPour` est d'éviter de gérer la progression de la variable qui sert de compteur.

Le seul intérêt de la forme `Pour` ... `FinPour` est d'éviter de gérer la progression de la variable qui sert de compteur.
Les structures TantQue sont employées dans les situations où l'on doit procéder à un traitement systématique sur les éléments d'un ensemble dont on ne connaît pas d'avance la quantité,

comme par exemple :

comme par exemple :
- le contrôle d'une saisie ;

comme par exemple :
- le contrôle d'une saisie ;
- la gestion des tours d'un jeu (tant que la partie n'est pas finie, on recommence)

Boucles imbriquées

De même qu'une structure SI ... ALORS peut contenir d'autres structures SI ... ALORS, une boucle peut tout à fait contenir d'autres boucles. Exemple :

`Pour` i variant de 1 à 9 `Faire`

```
Pour i variant de 1 à 9 Faire
    Pour j variant 1 à 9 Faire
```

```
Pour i variant de 1 à 9 Faire
    Pour j variant 1 à 9 Faire
        écrire i'x'j=i*j
```

```
Pour i variant de 1 à 9 Faire
    Pour j variant 1 à 9 Faire
        écrire i'x'j=i*j
    FinPour
```

```
Pour i variant de 1 à 9 Faire
    Pour j variant 1 à 9 Faire
        écrire i'x'j=i*j
    FinPour
FinPour
```

Exemple

```
i ← 1, j ← 1
TantQue i< 10 Faire
    TantQue j<10 Faire
        écrire i'x'j=i*j
        j ← j+1
    FinTantQue
    i ← i+1
FinTantQue
```

Instructions break et continue

Les instructions break et continue permettent de modifier l'exécution des instructions dans une boucle.

L'instruction break quitte la boucle en cours d'exécution.

L'instruction break quitte la boucle en cours d'exécution.
L'instruction continue saute à la fin de la boucle la plus imbriquée (itération suivante).

L'instruction pass

L'instruction pass ne fait rien. Elle peut être utilisée lorsqu'une instruction est requise syntaxiquement mais que le programme ne nécessite aucune action. Par exemple :

L'instruction pass

L'instruction pass ne fait rien. Elle peut être utilisée lorsqu'une instruction est requise syntaxiquement mais que le programme ne nécessite aucune action. Par exemple :

```
>>> while 1 :
... pass ...
```

Les boucles en Python

Deux types de boucles sont disponibles : les boucles for (pour) et while (TantQue).

Pour chacune : un en-tête qui décrit l'évolution de la boucle, un ensemble d'instructions qui sont évaluées à chaque itération de boucle et une partie optionnelle qui est évaluée en sortie de boucle (introduite par le mot-clé else).

Les boucles peuvent contenir les branchements continue pour passer à l'itération suivante et break pour sortir de la boucle (dans ce cas la clause else n'est pas évaluée).

Boucles en Python : pour = for

En Python l'instruction for s'écrit :

Boucles en Python : pour = for

En Python l'instruction for s'écrit :

for

```
for var in (chaine, liste, tuple ou dictionnaire)  :
        Instructions1
else
        Instructions2
```

pour = for
faire = :
dans = in
FinPour = retour la ligne

Exemple

```
>>> L=['veau', 'beau', 'chameau']
```

Exemple

```
>>> L=['veau', 'beau', 'chameau']
>>> for i  in L  :
```

Exemple

```
>>> L=['veau', 'beau', 'chameau']
>>> for i  in L  :
        print i+'x'
```

Exemple

```
>>> L=['veau', 'beau', 'chameau']
>>> for i  in L  :
        print i+'x'
>>> veaux
```

Exemple

```
>>> L=['veau', 'beau', 'chameau']
>>> for i  in L  :
        print i+'x'
>>> veaux
>>> beaux
```

Exemple

```
>>> L=['veau', 'beau', 'chameau']
>>> for i  in L  :
        print i+'x'
>>> veaux
>>> beaux
>>> chameaux
```

Fonction range()

Sert pour créer des listes de nombres entiers, croissantes ou décroissantes avec un pas. Sa syntaxe

Fonction range()

Sert pour créer des listes de nombres entiers, croissantes ou décroissantes avec un pas. Sa syntaxe

>>> range([début,] fin [,pas])

Fonction range()

Sert pour créer des listes de nombres entiers, croissantes ou décroissantes avec un pas. Sa syntaxe
>>> range([début,] fin [,pas])
(les arguments entre crochets sont optionnels).

Fonction range()

Sert pour créer des listes de nombres entiers, croissantes ou décroissantes avec un pas. Sa syntaxe

>>> range([début,] fin [,pas])

(les arguments entre crochets sont optionnels).

>>> range(8)

```
>>> range(8.4)
```

```
>>> range(8.4)
>>> range(8,16)
```

```
>>> range(8.4)
>>> range(8,16)
>>> range(8,16,2)
```

```
>>> range(8.4)
>>> range(8,16)
>>> range(8,16,2)
>>> range(8,16,3)
```

```
>>> range(8.4)
>>> range(8,16)
>>> range(8,16,2)
>>> range(8,16,3)
>>> range(28,12,-2)
```

```
>>> range(8.4)
>>> range(8,16)
>>> range(8,16,2)
>>> range(8,16,3)
>>> range(28,12,-2)
for i in range(10,100,2) :
    Print i**2
```

TantQue=while

while

while expression logique :

TantQue=while

while

while expression logique :
    Instructions1

TantQue=while

while

```
while expression logique  :
        Instructions1
else  :
```

TantQue=while

while

```
while expression logique  :
        Instructions1
else  :
        Instructions2
```

La clause else est évaluée losrque la condition est fausse et qu'il n'y a pas de break.

La clause else est évaluée losrque la condition est fausse et qu'il n'y a pas de break.
TantQue = while

La clause else est évaluée losrque la condition est fausse et qu'il n'y a pas de break.
TantQue = while
faire = :

La clause else est évaluée losrque la condition est fausse et qu'il n'y a pas de break.
TantQue = while
faire = :
FinTantQue = retour à la ligne

Exemple

i = 1

Exemple

```
i = 1
while i<= 10  :
```

Exemple

```
i = 1
while i<= 10  :
    print i**2
```

Exemple

```
i = 1
while i<= 10  :
    print i**2
    i = i + 1
```

Exemple

```
i = 1
while i<= 10  :
    print i**2
    i = i + 1
```

Exemple

```
i = 1
while i<= 10  :
    print i**2
    i = i + 1
```

TantQue=while

while

while test1 :

TantQue=while

while

while test1 :
    instructions

TantQue=while

while

```
while test1  :
    instructions
    if test2  : break
```

TantQue=while

while

```
while test1  :
    instructions
    if test2  : break
    if test3  : continue
```

TantQue=while

while

while test1 :
    instructions
    if test2 : break
    if test3 : continue
    autres instructions

TantQue=while

while

```
while test1  :
    instructions
    if test2  : break
    if test3  : continue
    autres instructions
else  :
```

TantQue=while

while

```
while test1  :
    instructions
    if test2  : break
    if test3  : continue
    autres instructions
else  :
    plus d'instructions
```

On peut utiliser break, continue, pass dans for comme dans un while