Chapitre 3 : Les structures de contrôle itératives (boucles)

Que faire lorsqu'un algorithme doit répéter plusieurs fois le même traitement ? Comment éviter d'écrire plusieurs fois les mêmes instructions ? Peut-on demander à un algorithme de recommencer un calcul tant qu'une condition est remplie ou jusqu'à ce qu'elle ne le soit plus ?

Ce chapitre répondra à ces questions.

3.1 Introduction

Dans les chapitres précédents, nous avons vu comment les instructions pouvaient être exécutées de manière séquentielle ou conditionnelle, c'est-à-dire en suivant un ordre précis ou en choisissant entre plusieurs alternatives. Cependant, dans de nombreuses situations, un algorithme doit répéter un traitement plusieurs fois. Par exemple, pour additionner plusieurs nombres, parcourir les notes d'une classe ou rechercher un élément dans une liste. Répéter manuellement les instructions rendrait l'algorithme long, lourd et difficile à modifier.

Pour répondre à ce besoin, nous introduisons ici les structures itératives (répétitives), plus connues sous le nom de boucles.

3.2 Définition

Une boucle est une structure de contrôle qui permet à un algorithme d'exécuter plusieurs fois un même bloc d'instructions, selon une condition donnée. Elle sert à automatiser des traitements répétitifs sans avoir à écrire plusieurs fois les mêmes instructions.

3.3 Types de boucles

En algorithmique et en langage C, on distingue principalement trois types de boucles :

1. Tant-que (while)
2. Répéter...jusqu'à (do...while)
3. Pour (for)

Copyright © BENADDA Meriem

3.4 La boucle “Tant que” (while)

La boucle tant-que répète un bloc d’instructions tant qu’une condition est vraie. La condition est testée avant chaque itération (répétitions).

Syntaxe

Algorithmique :

Tant-que (condition) **faire**
  Instruction(s) ;
Fin-Tant-Que

En langage C :

```
while (condition) {
    // instructions ;
}
```

Exemple : Affichage des dix premiers nombres entiers positifs

Algorithme nombre **;**
Variables i : entier;

Début
i ← 1 ;
Tant-que (i <= 10) **faire**
  Ecrire(i);
  i ← i+1 ;
Fin-Tant-Que ;
Fin.

```
#include<stdio.h>

int main(){
int i = 1;
while (i <= 10) {
    printf("%d\n", i);
    i++;
}
return 0 ;}
```

Remarques :

1. Si la condition est fausse dès le départ, le bloc ne s'exécute jamais.
2. Toujours s'assurer que la condition soit susceptible de devenir fausse, sinon on risque une boucle infinie.

3.5 La boucle “Répéter...jusqu’à” (do...while)

Contrairement à la boucle « Tant-que », la boucle « Répéter » exécute d’abord le bloc d’instructions, puis teste la condition. Elle s'exécute donc au moins une fois.

Syntaxe

Algorithmique :

Répéter
  Instruction(s) ;
Jusqu’à (condition) ;

En langage C :

```
do {
    // instructions ;
} while (condition) ;
```

Copyright © BENADDA Meriem

Exemple : Reprenons l'exemple précédent

Algorithme nombre **;**
Variables i : entier;

Début
i ← 1 ;
Répéter
  Ecrire(i);
  i ← i+1 ;
Jusqu'à (i > 10) **;**
Fin.

```
#include<stdio.h>

int main(){
int i = 1;
do {
    printf("%d\n", i);
    i++;
} while (i <= 5);
return 0 ;}
```

Remarque :

1. La boucle Répéter est à utiliser lorsqu'on veut que le traitement se fasse **au moins une fois**, peu importe la condition.
2. En C, *while* et *do.. while* ont la même condition contrairement à l'algorithmique.

3.6 La boucle "Pour" (for)

La boucle pour est utilisée lorsque le nombre d'itérations est connu à l'avance.

Syntaxe

En algorithmique :

Pour *id_compteur* ← *val_initiale* **à** *val_finale* **faire**
  Instruction(s) ;
Fin-Pour ;

id_compteur : Variable qui va compter les itérations
val_initiale : La première valeur que prend le compteur
val_finale : La dernière valeur que prend le compteur

En langage C :

```
for (initialisation; condition;
incrémentation) {

    // instructions;
}
```

Exemple : Reprenons l'exemple précédent

Algorithme nombre **;**
Variables i : entier;

Début
Pour i ← 1 **à** 10 **faire**
  Ecrire(i);
Fin-pour;
Fin.

```
#include<stdio.h>

int main(){
for (int i = 1; i <= 10; i++)
{
    printf("%d\n", i);
}
return 0 ;}
```

Copyright © BENADDA Meriem

Dans cet exemple, le traitement est statique, c'est-à-dire que l'algorithme affiche toujours les nombres de 1 à 10. Pour le rendre dynamique, on demande à l'utilisateur d'introduire un nombre, et l'algorithme affichera les nombres de 1 jusqu'au nombre saisi.

Exemple : Reprenons l'exemple précédent

Algorithme nombre **;**
Variables i, n : entier;

Début
Ecrire("Veuillez introduire un entier strictement positif");
Lire(n) ;
Pour i ← 1 **à** n **faire**
  Ecrire(i);
Fin-pour;
Fin.

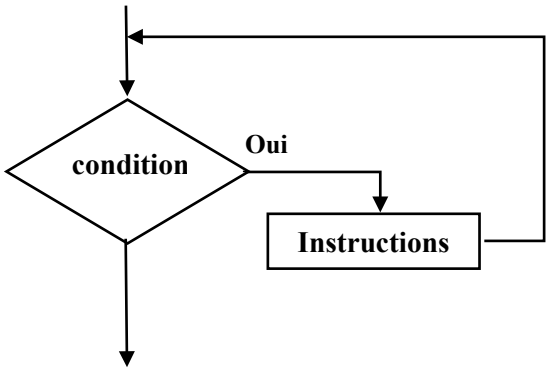
```
#include<stdio.h>

int main(){

printf("Veuillez introduire un entier
strictement positif \n");
scanf("%d", &n);

for (int i = 1; i <= n; i++) {
    printf("%d\n", i);
}
return 0 ;
}
```

Remarque:

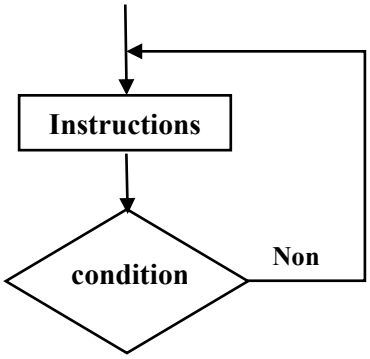
Dans la boucle pour, Le compteur est automatiquement incrémenté de 1 à chaque itération, sauf si une autre valeur d'incrémentation est spécifiée.

3.7 Les structures itératives en organigramme

a. La boucle "Tant que"

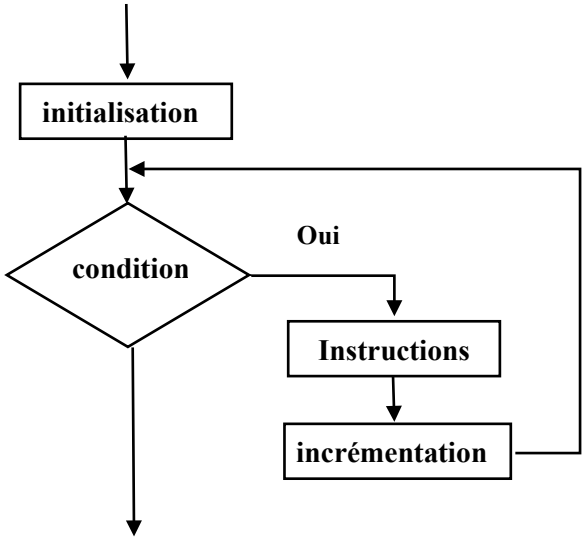


b. La boucle "Répéter"



Copyright © BENADDA Meriem

c. La boucle "Pour"



Remarques :

1. La boucle *Pour* peut être transformée en une boucle *Tant-que* ou *Répéter*. Mais, le contraire n'est pas toujours réalisable.

2. Si on transforme une boucle *Pour* en une boucle *Tant-que*, l'initialisation et l'incrémentation du compteur de répétitions doivent se faire manuellement dans la nouvelle forme de la boucle.

3.8 La boucle pour le contrôle d'entrée (saisie)

Lorsqu'un programme attend une valeur précise, l'utilisateur peut se tromper en saisissant une donnée incorrecte. Prenons par exemple la saisie de l'âge d'une personne : l'utilisateur pourrait entrer une valeur négative ou un texte à la place d'un nombre.

Ce type d'erreur peut provoquer :

- Une erreur d'exécution, si le type saisi ne correspond pas à celui attendu (ex. : l'utilisateur tape "vingt" au lieu de 20).

- Une erreur logique, si la valeur est du bon type mais hors des limites prévues (ex. : −5 ou 250).

Pour éviter ces situations, on doit mettre en place un contrôle d`entrée, c'est-à-dire une vérification des données saisies avant de continuer l'exécution du programme.

Le principe consiste à placer l'instruction de lecture dans une boucle qui se répète jusqu'à ce que la valeur saisie soit acceptable.

La boucle *Répéter* est particulièrement adaptée à ce type de contrôle même si elle peut bien être remplacée par la boucle *Tant-que*.

Copyright © BENADDA Meriem

Reprenons l'exemple précédent où nous demandions à l'utilisateur d'introduire un nombre strictement positif, nous allons donc contrôler n pour éviter toute mauvais saisie.

Exemple : Reprenons l'exemple précédent

Algorithme nombre **;**
Variables i, n : entier;

Début
Répéter
Ecrire("Veuillez introduire un entier strictement positif");
Lire(n) ;
Jusqu'à (n>0) ;
Pour i ← 1 **à** n **faire**
Ecrire(i);
Fin-pour;
Fin.

```
#include<stdio.h>

int main(){

do
{printf("Veuillez introduire un
entier strictement positif \n");
scanf("%d", &n);}
while(n<=0);

for (int i = 1; i <= n; i++) {
    printf("%d\n", i);
}
return 0 ;
}
```

3.9 Conclusion

Dans ce chapitre, nous avons abordé les structures de contrôle itératives qui sont fondamentales en algorithmique pour traiter des données en série, effectuer des calculs répétitifs ou parcourir des structures comme des tableaux.

Test d'acquisition des connaissances – Chapitre 3

QCM :

1. Quelle boucle garantit **au moins une exécution** ?
a) while
b) for
c) do...while
d) aucune

2. Quelle boucle est idéale si on connaît le nombre d'itérations dès le départ ?
a) for
b) while
c) do...while
d) switch

Vrai ou faux :

3. La boucle while s'exécute toujours au moins une fois.
4. Une boucle mal conçue peut provoquer une boucle infinie.
5. La boucle for peut être utilisée avec une décrémentation (ex : i--).

Copyright © BENADDA Meriem

Exercice de codage :

6. Écris une boucle en C qui affiche les nombres pairs entre 2 et 10 inclus.

Corrigé rapide du test :

- QCM :
  1 → c) do...while
  2 → a) for
- Vrai / Faux :
  3 → Faux
  4 → Vrai
  5 → Vrai

Copyright © BENADDA Meriem