Chapitre 4

Les structures itératives (Les boucles)

Sommaire

Objectif

L'objectif de ce chapitre est de construire des algorithmes comportant des traitements itératifs, qui consiste à répéter les mêmes instructions un certain nombre de fois à travers la boucle tant que, la boucle répéter et la boucle pour.

I. Introduction

Il arrive souvent des situations où on veut répéter une instruction ou un bloc d'instructions plusieurs fois. Pour ce faire on utile les structures répétitivités (boucles).

La plupart des langages de programmation proposent trois types de structures répétitives courantes :

- La boucle Pour
- La boucle Tant que
- La boucle Répéter

II. La boucle Pour (La boucle « for »)

La boucle ***Pour*** permet de répéter un bloc d'instructions un nombre de fois donné, dans ce cas on utilise un compteur et on s'arrête lorsque le compteur atteint sa valeur finale connue au préalable. Cette boucle a la structure suivante :

Syntaxe :

```
POUR I= valeur_initiale à valeur_finale pas=val FAIRE
        Ensemble des instructions à répéter
FINpour
```

Propriétés en langage algorithmique :

- La boucle POUR est utilisée lorsque le nombre de répétition est connu à l'avance.
- Pour chaque valeur de la variable de contrôle qui varie de la *valeur initiale* à *la valeur finale* avec un pas égale à *val*, le bloc sera exécuté.
- Le bloc est répété ***(valeur finale - valeur initiale+1)*** fois.
- La sortie de cette boucle s'effectuer lorsque le nombre souhaité de répétition est atteint.
- Dans cette boucle l'incrémentation (ajouter 1) est faite automatiquement.
- Lorsque le pas est Omis, il est supposé égal à (+1).

Syntaxe en Langage C :

```
for ( initialisation ; condition ; pas )
{
// Bloc d'instructions
}
```

Propriétés en langage C :

Il y a trois instructions condensées, chacune séparée par un point-virgule :

- La première est l'**initialisation** : cette première instruction est utilisée pour préparer notre variable compteur.
- La seconde est la **condition :** c'est la condition qui dit si la boucle doit être répétée ou non. Tant que la condition est vraie, la boucle for continue. Si la condition est fausse, on sort de la boucle et on continue avec l'instruction qui suit l'accolade fermante.
- Enfin, il y a **l'incrémentation :** cette dernière instruction est exécutée à la fin de chaque tour de boucle pour mettre à jour la variable compteur.

Exemple : Calculer la somme de N entiers

Algorithme	langage C

Algorithme Somme **;**
Var
N, X, SOM, i **: entier ;**
Debut
Lire (N) ;
SOM ← 0 ;
Pour i = 1 **à** N **Faire**
Lire (X) ;
SOM← SOM + X ;
Fin Pour
Ecrire ("LA SOMME =", SOM) ;
Fin
	#include<stdio.h>
	int main()
	{
	int N, X, SOM, i;
	printf("donner le nombre des entiers à additionner: ");
	scanf("%d",&N);
	SOM=0;
	for (i =1; i <N+1; i ++)
	{
	printf ("donner la valeur du nombre %d : " , i);
	scanf("%d", &X);
	SOM=SOM+X;
	}
	printf ("LA SOMME = %d \n", SOM);
	return o;
	}

La plupart du temps on fera une **incrémentation**, mais on peut aussi faire une **décrémentation (variable --)** ou encore n'importe quelle autre opération (variable += 2 ; pour avancer de 2 en 2 par exemple).

Il est fortement déconseillé de modifier la valeur du compteur (et/ou la valeur de finale) à l'intérieur de la boucle. En effet, une telle action :

- Perturbe le nombre d'itérations prévu par la boucle Pour
- Présente le risque d'aboutir à une boucle infinie

III. La boucle Tant que (La boucle « While »)

Une boucle ***Tant que*** permet de répéter plusieurs fois le même bloc d'instructions tant qu'une certaine condition reste vraie

Syntaxe :

```
TANT QUE COND FAIRE
    Bloc d'instructions
FIN TANT QUE
```

Propriétés en langage algorithmique :

- Le nombre de répétitions n'est pas connu à l'avance il dépend de la valeur de la condition **COND.**
- Au début de chaque itération, la condition est évaluée.
- Si **COND** est Vraie le bloc est exécuté sinon on sort de la boucle
- Le Bloc est répété tant que **COND** est Vraie
- Si **COND** est fausse au début on n'exécute aucune instruction. Donc on peut avoir des boucles qui ne sont jamais exécutées.

Syntaxe en Langage C:

```
while ( Condition )
{
  // Bloc d'instructions
}
```

Propriétés en langage C :

- La condition (dite condition d'arrêt) est évaluée avant chaque itération
- Si la condition est vraie, on exécute le bloc d'instructions (corps de la boucle), puis, on retourne tester la condition. Si elle est encore vraie, on répète l'exécution, ...
- Si la condition est fausse, on sort de la boucle et on exécute l'instruction qui se trouve juste après l'accolade fermante « } ».

Exemple 1 : On souhaite écrire un programme qui permet de contrôler la saisie d'un entier positif.

```
int entierPositif = 0;
while ( entierPositif <=0)
{
printf (" Tapez un entier positif ! ");
scanf ("%d", & entierPositif );
}
```

Exemple 2 : Calculer la somme de N entiers

Algorithme	langage C

Algorithme Somme **;**
Var
N, X, SOM, i **: entier ;**
Debut
Lire (N) ;
SOM ← 0 ;
i ← 1 ;
 	
Tant que (I <= N) Faire
Lire (X) ;
SOM← SOM + X ;
i←i+1 ;
FinTantque
Ecrire ("LA SOMME =", SOM) ;
Fin
	#include<stdio.h>
	int main()
	{
	int N, X, SOM, i;
	printf("donner le nombre des entiers à additionner: ");
	scanf("%d",&N);
	SOM=0;
	i = 1 ;
	while (i <= N)
	{
	printf ("donner la valeur du nombre %d : " , i);
	scanf("%d", &X);
	SOM=SOM+X;
	i++ .
	}
	printf ("LA SOMME = %d \n", SOM);
	return o;
	}

IV. La boucle Répéter (La boucle « do...while »)

Cette structure est très similaire à la structure ***Tant que*** du fait que leurs exécutions dépendent d'une condition. Elle permet de répéter une série d'instructions jusqu'à la vérification d'une certaine condition.

La différence entre ***Tant que*** et ***Répéter*** est que la boucle ***Tant que*** peut ne jamais être exécutée car la condition peut être non vérifiée au démarrage alors que le contenu de la boucle ***Répéter*** est exécuté au moins une fois pour arriver à la condition.

Syntaxe :

Répéter
Bloc d'instructions
jusqu'à COND

Propriétés en langage algorithmique:

- Le Bloc d'instructions est répété jusqu'à ce que la condition **COND** égale à Vraie.
- En fin de chaque itération l'expression logique **COND** est évaluée.
- Avec cette boucle le Bloc est répété au moins une seule fois.

Syntaxe en Langage C :

```
do
{
  // Bloc d'instructions
} while ( Condition );
```

Propriétés en langage C :

- La boucle « do...while » est très similaire à while
- La seule chose qui change par rapport à while, c'est la position de la condition. Au lieu d'être au début de la boucle, la condition est à la fin.
- Cette boucle s'exécutera donc toujours au moins une fois.
- Dans la boucle « do...while », n'oubliez pas de mettre un point-virgule à la fin.

Exemple : Calculer la somme de N entiers

Algorithme	langage C

Algorithme Somme **;**
Var
N, X, SOM, i **: entier ;**
Debut
Lire (N) ;
SOM ← 0 ;
i ← 1 ;
 	
Repeter
Lire (X) ;
SOM← SOM + X ;
i←i+1 ;
Jusqu'à (i > N)
Ecrire ("LA SOMME =", SOM) ;
Fin
	#include<stdio.h>
	int main()
	{
	int N, X, SOM, i;
	printf("donner le nombre des entiers à additionner: ");
	scanf("%d",&N);
	SOM=0;
	i = 1 ;
	do
	{
	printf ("donner la valeur du nombre %d : " , i);
	scanf("%d", &X);
	SOM=SOM+X;
	i++ .
	} **while** (I <= N);
	printf ("LA SOMME = %d \n", SOM);
	return o;
	}

V. Les boucles imbriquées

Le bloc d'instructions d'une boucle peut contenir lui-même une autre boucle. C'est ce qu'on appelle des boucles imbriquées

Le principe des boucles imbriquées est :

1. On rentre dans la boucle mère (qui englobe la deuxième boucle).
2. On rentre dans la boucle inclue et on la parcours jusqu'à arriver à sa condition de sortie.
3. On sort de la boucle inclue et on revient donc au niveau de la boucle mère.
4. On itère sur la boucle englobant et l'on revient donc sur la boucle inclue.
5. Et ainsi de suite.

Exemple 1 :

```
int i;
int j=1;
for (i = 1 ; i <= 3 ; i++)
 {
    j=1
    while (j <= 4)
    {
     printf ("i=%d et j=%d!\n", i, j);
     j++;
    }
 }
```

Historique d'exécution

Inst.	i	j	Affichage
1	1	1	i=1 et j=1
2	1	2	i=1 et j=2
3	1	3	i=1 et j=3
4	1	4	i=1 et j=4
5	2	1	i=2 et j=1
6	2	2	i=2 et j=2
7	2	3	i=2 et j=3
8	2	4	i=2 et j=4
9	3	1	i=3 et j=1
10	3	2	i=3 et j=2
11	3	3	i=3 et j=3
12	3	4	i=3 et j=4

Exemple 2 : Programme d'affichage des tables de multiplication de 1 à 9

Algorithme	langage C

Algorithme Table_Multipliction ;
Var
i, j, resultat : entier ;
Pour i Allant de 1 A 9 Faire
Pour j Allant de 1 A 9 Faire
resultat ← i*j ;
Ecrire (i,"*",j, "=",resultat) ;
FinPour
Fin pour
Fin.
	#include<stdio.h>
	int main ()
	{ int i, j;
	int resultat = 0;
	for (i = 1; i < 10; i++)
	for (j = 1; j <= 10; j++)
	{
	resultat = i * j;
	printf ("%d x %d = %d\n", i,j,resultat);
	}
	return 0;
	}

Quelle boucle puis-je utiliser pour mon programme ?

Utilisez la structure qui reflète le mieux l'idée du programme que vous voulez réaliser :

- Si le bloc d'instructions ne doit pas être exécuté si la condition est fausse, alors utilisez while ou for.
- Si le bloc d'instructions doit être exécuté au moins une fois, alors utilisez do - while.
- Si le nombre d'exécutions du bloc d'instructions est connu à l'avance alors utilisez de préférence for.
- Si le bloc d'instructions doit être exécuté aussi longtemps qu'une condition extérieure est vraie alors utilisez while.

Le choix entre for et while n'est souvent qu'une question de préférence ou d'habitudes.

VI. Conclusion

Les structures répétitives, également appelées structures itératives ou encore boucles, sont introduites dans ce chapitre. Techniquement, si le nombre de répétitions est connu à l'avance on préfère utiliser la boucle for. Sinon, dans le cas où le nombre de répétition est imprévisible et dépend de l'évaluation d'une condition il faut utiliser un while ou un do..while.