Introduction

Un programme a presque toujours pour rôle de répéter la même action un certain nombre de fois. Pour ce faire, on utilise une structure permettant de dire «Exécuter telles actions jusqu'à ce que telle condition soit remplie ».

Bien qu'une seule soit nécessaire, la plupart des langages de programmation proposent trois types de structures répétitives.

1. La boucle Tant que

Une boucle Tant que permet de répéter plusieurs fois le même bloc d'instructions tant qu'une certaine condition reste vraie. Le formalisme de cette structure dans un organigramme est comme suit :

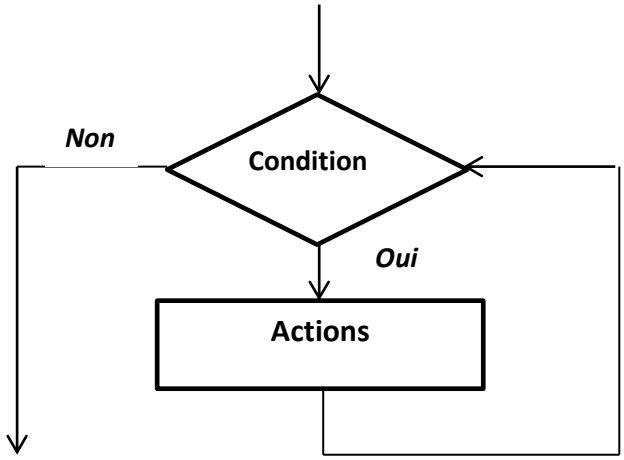

Figure1 : Structure de la boucle Tant que

Ce qui signifie tant que la condition est vraie, on exécute les actions.

Note :

- Le bloc d'instructions peut ne jamais être exécuté si dès le départ la condition de la boucle n'est pas satisfaite. Donc on peut avoir des boucles qui ne sont jamais exécutées.
- A l'intérieur de la boucle il est impératif de faire des changements de manière à ce que le test soit erroné à un moment ou l'autre sinon si la condition reste toujours vraie on se retrouve dans une situation de boucle infinie qui est formellement interdite en algorithmique.

Vocabulaire et syntaxe:

Algorithme	***Langage C***
La syntaxe de cette structure en langage algorithmique est la suivante : **Tant Que** Condition **Faire** Bloc d'instructions **Fin Tant Que**	***while*** (expression) ***{*** Bloc d'instructions ***}***

Exemple :

Calculer la somme de N entiers

Algorithme Somme ;
Variables
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 (SOM)
Fin
	#include<stdio.h>
	int main()
	{
	int N, X, SOM, I;
	printf("DONNER LE NOMBRE DES ENTIERS A 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 0;
	}

2. La boucle Faire Tant que

Comme la boucle "tant que", ce type de répétitive est utilisé lorsque le nombre de fois que la séquence d'instructions à répéter est inconnu au moment où cette séquence est abordée pour la première fois mais le corps de la boucle est **toujours exécuté au moins une fois**.

La différence entre *Tant que* et *Faire tant que* est que la boucle Tantque peut ne jamais être exécutée car la condition peut être non vérifiée au démarrage alors que le contenu de la boucle Faire Tant que est exécuté au moins une fois pour arriver à la condition. Le formalisme de cette structure dans un organigramme est comme suit :

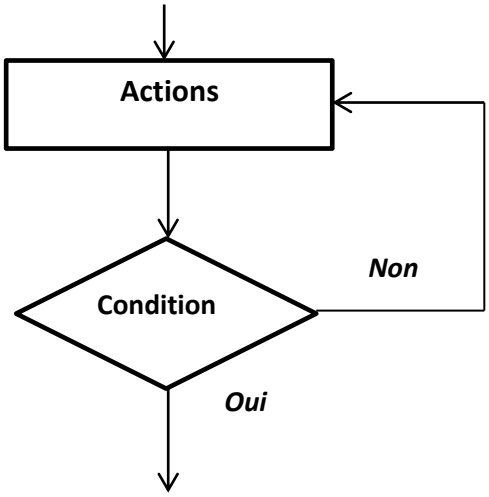

Figure 2: Structure de la boucle Faire Tant que

Vocabulaire et syntaxe:

Algorithme	***Langage C***
Faire <Traitement> **Tant que** (condition de refaire)	***do*** *{* Instructions *}* ***while (expression);***

Les étapes d'exécution de la boucle Faire Tant que:

1) Exécution du <Traitement>

2) Test de la valeur de la <condition d'arrêt>. Si elle est vérifiée Alors la boucle s'arrête, sinon retour à l'étape 1.

Cet ordre d'itération permet de répéter le <Traitement> une ou plusieurs fois et de s'arrêter sur une condition. En effet, lorsque la condition est vérifiée, la boucle s'arrête, sinon elle ré-exécute le <Traitement>.

Remarques

1. Dans cette boucle, le traitement est exécuté au moins une fois avant l'évaluation de la condition d'arrêt.

2. Il doit y avoir une action dans le <Traitement> qui modifie la valeur de la condition.

Exemple :

Calculer la somme de N entiers

Algorithme Somme ;
Variables
N, X, SOM, I : entier ;
Debut
Lire (N)
SOM ← 0 ;
I← 1
Faire
Lire (x) ;
SOM← SOM + X
I←I+1
Tant que (I ≤ N)
Ecrire (SOM)
Fin
	#include<stdio.h>
	int main()
	{
	int N, X, SOM, I;
	printf("DONNER LE NOMBRE DES ENTIERS A 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 0;
	}

3. La boucle Pour

Très souvent, on utilise une structure répétitive avec un nombre de répétitions connu, dans ce cas un utilise un compteur et on s'arrête lorsque le compteur atteint sa valeur finale connue au préalable.

C'est pourquoi la plupart des langages de programmation offrent une structure permettant d'écrire cette répétitive plus simplement

Vocabulaire et syntaxe:

Algorithme	***Langage C***
Pour i **Allant de** v1 **A** v2 **Pas** p instructions **FinPour**	***for*** (*expr 1* ; *expr 2* ; *expr 3*) ***{*** *Instructions* ***}***

Remarque : Lorsque le pas est Omis, il est supposé égal à (+1).

Exemple :

Calculer la somme de N entiers

Algorithme Somme ;
Variables
N, X, SOM, I : entier ;
Debut
Lire (N)
SOM ← 0 ;
Pour i **Allant de** 1 **A** N **PAS** 1 **Faire**
Lire (x) ;
SOM← SOM + X
Fin Pour
Ecrire (SOM)
Fin
	#include<stdio.h>
	int main()
	{
	int N, X, SOM, I;
	printf("DONNER LE NOMBRE DES ENTIERS A 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 0;
	}

4. Les boucles imbriquées

Comme nous avons vu précédemment, les boucles exécutent une ou plusieurs instructions (bloc d'instruction). Ces blocs d'instructions peuvent contenir de leurs tours des boucles. Dans ce cas on parle de boucles imbriquées.

Exemple :

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 :

Programme d'affichage des tables de multiplication de 1 à 9

Algorithme Table_Multipliction
Variables
i,j,resultat : entier
Pour i **Allant de** 1 **A** 9 **Faire**
 	
Pour j **Allant de** 1 **A** 9 **Faire**
 	
resultat ← i*j
Ecrire (resultat)
 	
FinPour
Fin pour
	#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;
	}

Conclusion

Les structure 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.*