Une boucle à quoi ça sert ?

Une boucle permet de répéter une instruction (ou une liste d'instructions) plusieurs fois.
Il y a principalement trois types de boucles :

- Les boucles pour répéter une instruction un certain nombre de fois, il s'agit de la boucle **Pour**
- Les boucles pour répéter une instruction jusqu'à une condition d'arrêt, il s'agit des boucles **Tant que** et **répéter**

Le passage dans une boucle est appelé itération
Soit l'exemple suivant :

Algorithme Exemple1
Début
Ecrire(1)
Ecrire(2)
Ecrire(3)
Ecrire(4)
Ecrire(5)
Fin

Et soit l'exemple : Ecrire un algorithme / programme qui affiche le nombre 2 à 1000 fois ?

1. La boucle Pour

L'instruction de boucle Pour permet de répéter l'exécution d'un bloc d'instructions. Elle utilise un compteur d'itération, une valeur initiale et une valeur finale du compteur. Le compteur est incrémenté automatiquement de **1.** Elle se caractérisent par le fait que l'on connait à l'avance le nombre d'itérations que l'on va devoir effectuer.

La syntaxe de l'instruction de la boucle Pour est :

Algorithme	**Code C**
1) **Pour** cpt de vi **à** vf **Faire** <instruction_1> **Fin Pour**	**1)** **For** (cpt=vi ; vi<=vf ; vi++) <instruction_1> ;
2) **Pour** cpt de vi **à** vf **Faire** <instruction_1> <instruction_2> …….. <instruction_n> **Fin Pour**	**2)** **For** (cpt=vi ; vi<=vf ; vi++) **{** <instruction_1> ; <instruction_2> ; …….. ; <instruction_n> ; **}**

Rq : cpt : le compteur, vi : Valeur initiale, vf : Valeur finale

Dans la boucle Pour, à chaque instant, on connait le nombre d'itérations déjà effectuées et aussi le nombre d'itérations restantes.

2. La boucle Tantque (TQ)

Les boucles tant que permettent d'effectuer des itérations tant qu'une certaine condition est vérifiée. On ne connait pas le nombre d'itérations à effectuer, mais à chaque itération, on vérifie si la condition est vraie ou fausse. Dès que cette condition est fausse, on sort de la boucle.

La syntaxe de l'instruction de la boucle Tant-que est :

Algorithme	**Code C**
1) **Tantque** <condition> **Faire** <instruction_1> **Fin Tantque** **2)** **Tantque** <condition> **Faire** <instruction_1> <instruction_2> …….. <instruction_n> **Fin Tantque**	**1)** **while** <condition> <instruction_1> ; **2)** **while** <condition> **{** <instruction_1> ; <instruction_2> ; …….. ; <instruction_n> ; **}**

Rq : le nombre de répétition des instruction dans la boucle tantque est [0..n]

3. La boucle Répéter

L'instruction de boucle Répéter permet de répéter l'exécution d'un bloc d'instructions. À chaque itération, une expression booléenne (condition) est réévaluée :

- Si l'expression donne TRUE : donc on arrête la boucle et on exécuter l'instruction qui vient après Répéter ;
- Si l'expression donne FALSE : on continue la boucle en exécutant l'itération suivante

La syntaxe de l'instruction de la boucle Répéter est :

Algorithme	**Code C**
1) **Répéter** <instruction_1> **Jusqu'à** <condition> **2)** **répéter** <instruction_1> <instruction_2> …….. <instruction_n> **Jusqu'à** <condition>	**1)** **Do** <instruction_1> ; **while** <condition> **2)** **Do** **{** <instruction_1> ; <instruction_2> ; …….. ; <instruction_n> ; **}** **while** <condition> ;

Rqs :

- Le nombre de répétition des instruction dans la boucle tantque est [1..n]
- En général, si la première itération est réalisée sans condition, on peut utiliser l'instruction **Répéter** au lieu de l'instruction **Tant-que.**
- L'instruction **Répéter** est équivalente à **Tantque.**
- L'instruction **Pour** est un cas spécial de l'instruction **Tantque**.

Remarques sur les instructions itératives

Une instruction peut être une instruction simple ou un bloc. Les différentes boucles peuvent être **imbriquées**, il faut donc s'astreindre à une certaine mise en page pour faciliter la lecture d'un programme.

4. Ruptures de séquence

Quand on veut sortir d'une boucle alors que la condition de passage est encore valide, on utilise des opérations appelées rupture de séquence.

Les ruptures de séquence sont utilisées lorsque plusieurs conditions (conditions multiples) conditionnent l'exécution d'un ensemble d'instructions.

En C, les ruptures d'instructions sont réalisées par quatre instructions qui correspondent à leur niveau de travail :

- **continue**
- **break**
- **goto**
- **return**

Deux appels de fonctions de la bibliothèque permettent aussi de modifier l'ordre d'exécution du programme, il s'agit de :

void exit (int status) : l'appel à cette fonction permet de terminer l'exécution du programme.

void longjmp(jmp_buf env, int val) : qui permet de sauter à un point de reprise mis dans le programme.

4.1. continue

Cette instruction permet le passage à l'itération suivante dans une boucle en sautant à la fin du bloc.

4.2. break

Il permet de sortir d'un bloc associé à une instructions répétitive ou alternative contrôlée par les instructions : if, for, while ou do while. Il ne permet de sortir que d'un niveau d'imbrication.

4.3. return

Elle provoque la terminaison de l'exécution de la fonction dans laquelle elle se trouve et le retour à la fonction appelante

4.4. goto

Permet d'aller n'importe où à l'intérieur d'une fonction. Il est associé à une étiquette appelé label, un label est une chaine de caractères suivie par du double points « : »

La commande de branchement goto et l'instruction repérer par étiquette doivent se trouver dans la même fonction.

Exemple :

Algorithme	**Code C**

Algorithme exemple
Variable x,i : entier
Début
x←0, i←1
loop : x←x+i
i←i+1
si (i<100) alors
aller à loop
fsi
ecrire(x)
Fin
	#include<stdio.h>
	main()
	{
	int i,x;
	i=1; x=0;
	loop : x=x+i;i++;
	if (i<100)
	goto loop;
	printf("%f",x);
	}

Exercice 1

Ecrire un algorithme qui demande un nombre compris entre 10 et 20, jusqu'à ce que la réponse convienne. En cas de réponse supérieure à 20, on fera apparaître un message : « Plus petit ! », et inversement, « Plus grand ! » si le nombre est inférieur à 10.

Exercice 2

Ecrire un algorithme qui demande un nombre de départ, et qui ensuite affiche les dix nombres suivants. Par exemple, si l'utilisateur entre le nombre 17, le programme affichera les nombres de 18 à 27.

Exercice 3

Ecrire un algorithme qui demande un nombre de départ, et qui ensuite écrit la table de multiplication de ce nombre, présentée comme suit (cas où l'utilisateur entre le nombre 7) :
Table de 7 :
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
…
7 x 10 = 70

Exercice 4

Ecrire un algorithme qui demande un nombre de départ, et qui calcule la somme des entiers jusqu'à ce nombre. Par exemple, si l'on entre 5, le programme doit calculer : 1 + 2 + 3 + 4 + 5 = 15 NB : on souhaite afficher uniquement le résultat, pas la décomposition du calcul.

Exercice 5

Ecrire un algorithme qui demande successivement 20 nombres à l'utilisateur, et qui lui dise ensuite quel était le plus grand parmi ces 20 nombres :

Entrez le nombre numéro 1 : 12

Entrez le nombre numéro 2 : 14

...

Entrez le nombre numéro 20 : 6

Le plus grand de ces nombres est : 14

Exercice 6

Ecrire un algorithme qui demande successivement 10 nombres à l'utilisateur, et qui affiche à la fin le plus grand de ces 10 nombres et son rang dans la liste saisie !

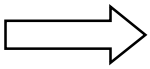
Exemple :

Entrez le nombre numéro 1 : 13

Entrez le nombre numéro 2 : 17

…..

Entrez le nombre numéro 10 : 5



Le plus grand de ces nombres est : 17
C'était le 2ème nombre saisi

Exercice 7

Ecrire l'algorithme qui vérifie si un nombre N positif est premier ou non. Un nombre est premier s'il n'est divisible que par 1 et par lui-même.

RQ : afficher un message d'erreur si N<=0

Exercice 8

Ecrire un algorithme qui Lit d'abord une valeur entière Ensuite il va lire 20 nombres entiers quelconques Ensuite il va déterminer combien de fois la première valeur a été saisie (sans compter la première saisie).

Exercice 9

Ecrire un algorithme mettant en œuvre le jeu suivant entre deux joueurs : Le premier utilisateur saisi un entier que le second doit deviner. Pour cela, il a le droit à autant de tentatives qu'il souhaite. A chaque échec, le programme lui indique si l'entier est plus grand ou plus petit que sa proposition.
Un score est affiché lorsque l'entier est trouvé.

Exercice 10

Ecrire un algorithme qui permet d'afficher tous les couples (x,y) , ou x est un entier compris entre 1 et p et y est un entier compris entre 1 et q .

p et q sont des entiers strictement positifs lus au clavier.

Exemple : p = 3 q= 4

Les couples affichés sont :

(1,1),(1,2),(1,3), (1,4)

(2,1),(2,2),(2,3),(2,4)

(3,1) , (3,2) , (3,3) , (3,4)