Les boucles

Les boucles en algorithmique sont utilisées lorsqu'on veut répéter un bloc d'instructions jusqu'à la vérification d'une certaine condition. Le langage C, comme en algorithmique, offre 3 types de boucles qui sont : la boucle *'for'*, la boucle *'while'* et la boucle *'do .. while'*.

1 La boucle 'For'

Algorithmique	Langage C

Pour <idf_compteur> de <BI> à <BS>
Faire
  //bloc d'instructions à exécuter ;
Fait ;
	for (<initialisation>;<condition>;<mise à jour>)
	{
	  //bloc d'instructions à exécuter ;
	}

La boucle 'for' fonctionne comme suit :

1. au début de l'exécution, l'initialisation de la variable (indice) se fait une et une seule fois ;
2. la condition est évaluée, et si elle est vraie, le bloc d'instructions est exécuté. Si la condition est fausse alors, le bloc ne sera pas exécuté ;
3. à la fin de l'exécution du bloc, la mise à jour de la variable aura lieu ;
4. aller à (2).

2 La boucle 'While'

Algorithmique	Langage C

Tant que (<condition>)
Faire
  //bloc d'instructions à exécuter ;
Fait ;
	while(<condition>)
	{
	  //bloc d'instructions à exécuter ;
	}

Comme la boucle 'for', l'évaluation de la condition est faite avant l'exécution du bloc d'instructions. L'évolution de l'évaluation de la condition doit se faire dans de corps de la boucle afin d'éviter une **boucle infinie**.

On appelle une **boucle infinie**, une boucle dont sa condition reste toujours vraie durant l'exécution.

Exemple : while(1){//bloc d'instructions à exécuter ;} est une boucle infinie.

3 La boucle ‘Do .. while’

Algorithmique	Langage C

Répéter
  //bloc d’instructions à exécuter ;
Tant que (<condition >) ;
	do{
	  //bloc d’instructions à exécuter ;
	} while(<condition>) **;**

Le bloc d’instructions dans la boucle ‘do .. while’ est exécuté au moins une fois. Contrairement aux boucles précédentes, l’évaluation de la condition est effectuée après l’exécution du bloc d’instructions. Le bloc est exécuté tant que la condition est vérifiée.

Exemples

La somme des N premiers nombres :

En utilisant l’instruction ‘for’ :

main.c

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int n, s, i;
    printf("Donner un nombre positif: ");
    scanf("%d", &n);

    s=0;
    for(i=0; i<= n; i++){
        s+=i;
    }
    printf("La somme des %d premiers nombres est:\t%d\n", n, s);
    return 0;
}
```

Remarque : il est plus juste d’initialiser i dans la boucle *‘for’* et la boucle *‘while’* à 1. Le résultat reste toujours vrai, mais on améliore la performance en termes de temps d’exécution : on évite l’exécution inutile de l’instruction **‘s+=i’ lorsque ‘i=0’ (s+=0)**. Toutefois, l’initialisation de i à 0 dans le cas de la boucle *‘do .. while’* est obligatoire !

En utilisant l'instruction 'while' :

main.c main.c

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int n, s, i;
    printf("Donner un nombre positif: ");
    scanf("%d", &n);

    s=0;
    i=0;
    while(i<= n){
        s+=i;
        i++;
    }
    printf("La somme des %d premiers nombres est:\t%d\n", n, s);
    return 0;
}
```

En utilisant l'instruction 'do .. while' :

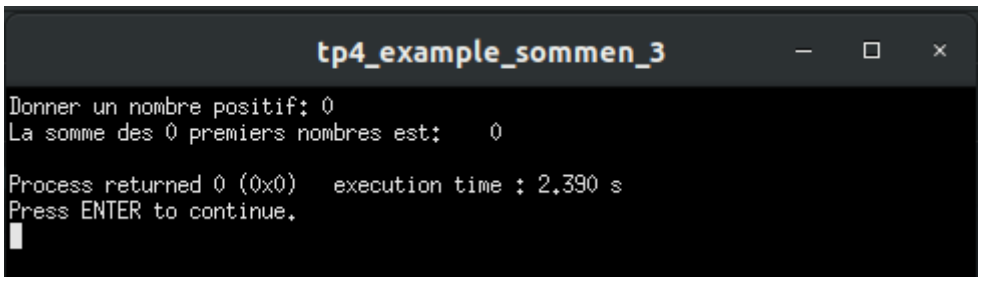
main.c

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int n, s, i;
    printf("Donner un nombre positif: ");
    scanf("%d", &n);

    s=0;
    i=0;
    do{
        s+=i;
        i++;
    }while(i<=n);
    printf("La somme des %d premiers nombres est:\t%d\n", n, s);
    return 0;
}
```

Exécution :



Solution plus complète :

Puisque la somme de N premiers nombres n'existe pas lorsque N est négatif, alors on assure que le N soit positif :

main.c

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int n, s, i;
    //lecture de n:
    do{
        printf("Donner un nombre positif: ");
        scanf("%d", &n);
    }while(n<0);
    // calcul de la somme:
    s=0;
    for(i=1; i<=n; i++){
        s+=i;
    }
    //affichage du resultat:
    printf("La somme des %d premiers nombres est:\t%d\n", n, s);
    return 0;
}
```

```
tp4_example_sommen_3
Donner un nombre positif: -5
Donner un nombre positif: -9
Donner un nombre positif: -5
Donner un nombre positif: -458
Donner un nombre positif: -68
Donner un nombre positif: 6
La somme des 6 premiers nombres est:    21
```

```
tp4_example_sommen_3
Donner un nombre positif: 0
La somme des 0 premiers nombres est:    0

Process returned 0 (0x0)   execution time : 1.754 s
Press ENTER to continue.
```

Exercices (Série TD)

1. Écrire l'algorithme qui affiche les tables de multiplication de 1 à 9 dans la plage de 1 à 9.
2. L'affichage de l'alphabet complet ('A' à 'Z') ou ('a' à 'z').
3. Déterminer si A est divisible par B. Avec A et B des entiers positifs.

4. Calculer A^n en utilisant uniquement l'opération de multiplication, N strictement positif.
5. Écrire l'algorithme permettant de déterminer le PGCD de deux nombres entiers A et B en utilisant la méthode euclidienne.
   - par soustractions successives.
   - par la division euclidienne.
6. Soit N nombres réels quelconques, calculer le nombre d'occurrence d'une valeur VAL donnée.
7. Recherche du minimum et du maximum dans un ensemble de N nombres.
8. Calcul du quotient et reste de la division de deux entiers A et B sans utiliser l'opération de division.
9. Déterminer tous les diviseurs d'un entier X donné.
10. Déterminer si un nombre entier X est premier ou non.
11. Écrire un algorithme qui pour un entier donné calcule la somme des chiffres qui le compose.
12. Écrire un algorithme qui calcule la somme d'ordre N de *Sn* définie comme suit en utilisant seulement les opérateurs de base (**sans l'utilisation de l'opérateur de puissance).**

$$Sn=\sum_{i=0}^{N}\frac{(-1)^{i+1}}{x^{i}}.$$