Support de cours

Chapitre 4 : Les boucles

Les structures répétitives appelées souvent les boucles permettent de répéter l'exécution d'une instruction ou d'un bloc d'instructions plusieurs fois. Les boucles sont très utiles pour la programmation, ils permettent l'automatisation des calculs. Les traitements répétitifs peuvent être réalisés en utilisant les branchements avec étiquettes mais un programme qui utilise les étiquettes est difficile à maintenir. La solution idéale pour implémenter les traitements répétitifs est l'utilisation des boucles. On distingue trois type de boucles, à savoir : la boucle *tantque*, la boucle *repeter* et la boucle *pour*.

1) La boucle Pour

Elle permet de répéter l'exécution d'un bloc d'instructions un certain nombre de fois connu à l'avance. Une variable compteur de type entier est utilisée pour contrôler le nombre d'itérations de la boucle. Le nombre d'itérations d'une boucle est le nombre de fois que les instructions de la boucle sont exécutées.

a) **La boucle pour en algorithmique** :

Syntaxe :

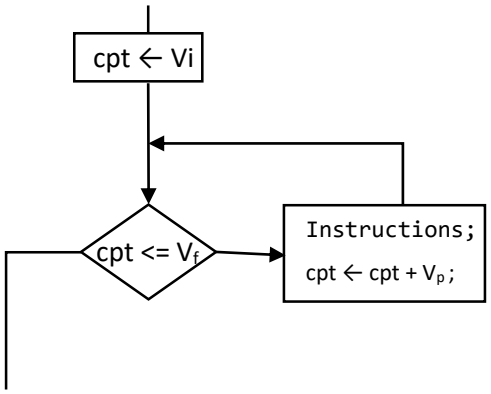
```
Pour cpt allant de Vi jusqua Vf Pas=Vp faire
   instructions;
finpour;
```

Fonctionnement :

La variable compteur `cpt` est initialisée à la valeur V_i. Si cpt est inférieur ou égale à la valeur V_f, les instructions de la boucle sont exécutées et la variable `cpt` prend la valeur `cpt+`V_p. Le processus recommence ensuite pour exécuter la prochaine itération. Si `cpt>Vf`, l'exécution de la boucle s'arrête et l'exécution de l'algorithme continue après `finpour`. Le champ `Pas` est facultatif, lorsque il est omis, sa valeur par défaut est 1.

Ci-dessous l'implémentation de la boucle **pour** en utilisant les étiquettes ainsi que sa modélisation graphique par un organigramme :

```
cpt ← Vi ;
E1 :   Si (cpt <= Vf) alors
       Instructions ;
       cpt ← cpt + Vp ;
       Aller a E1 ;
       Finsi;
```



Support de cours

Exemples :

- L'algorithme suivant permet d'afficher les nombres entiers compris entre la valeur 1 et la valeur d'une variable n lue au clavier.

```
Algorithme affiche ;
Var i : entier ;
Debut
     Lire(n);
     Pour i allant de 1 a n faire
        Ecrire(i);
     Finpour;
Fin.
```

L'exécution de cet algorithme avec une valeur de n = 10, affiche à l'écran la suite 1 2 3 4 5 6 7 8 9 10. Dans cet algorithme le champ Pas est absent alors sa valeur par défaut est 1.

- Un autre algorithme qui affiche les nombres impairs inferieur à une valeur n lue au clavier.

```
Algorithme affiche ;
Var i : entier ;
Debut
     Lire(n);
     Pour i allant de 1 a n Pas=2 faire
        Ecrire(i);
     Finpour;
Fin.
```

L'exécution de cet algorithme avec une valeur de n = 10, affiche à l'écran la suite 1 3 5 7 9.

b) La boucle pour en langage C

Syntaxe :

for (expr 1 ; expr 2 ; expr 3)
instruction ;
	for (expr 1 ; expr 2 ; expr 3){
	instruction 1;
	instruction 2;
	instruction 3;
	. . .
	}

Fonctionnement :

La boucle for en langage C est composée de trois expressions. `expr1` est l'expression d'initialisation, elle est exécutée uniquement à la première itération. `expr2` est une expression logique qui contient la condition d'arrêt de la boucle, lorsque sa valeur devient faux, l'exécution des instructions de la boucle s'arrête. `expr3` est exécuté à la fin de chaque itération de la boucle, elle est utilisée pour incrémenter la variable compteur de la boucle. `expr1` et `expr3` de la boucle for peuvent contenir plusieurs instructions séparées par des virgules.

Support de cours

Exemples :

Ci-dessous trois version d'un programme permettant de calculer la somme : S = 1 + 2 + 3+ …+ n.

```
#include <stdio.h>
int main(){
int n,s,i;
s=0;
printf("n=");
scanf("%d",&n);
for(i=1;i<=n;i++)
    s=s+i;
printf("s = %d ",s);
}
```

```
#include <stdio.h>
int main(){
int n,s,i;
printf("n=");
scanf("%d",&n);
for(i=1,s=0;i<=n;i++)
    s=s+i;
printf("s = %d ",s);
}
```

```
#include <stdio.h>
int main(){
int n,s,i;
printf("n=");
scanf("%d",&n);
for(i=1,s=0;i<=n;s=s+i,i++);
printf("s = %d ",s);
}
```

2) La boucle tantque :

La boucle *tantque* permet de répéter l'exécution d'une ou de plusieurs instruction tant qu'une condition est vérifiée. Le contrôle de la boucle *tantque* est assuré avec une variable booléenne ou une expression logique.

Syntaxe :

Algorithmique	Langage C

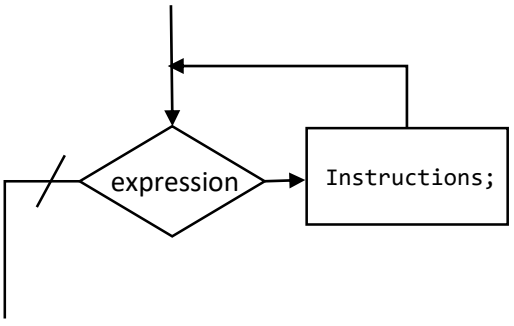
Tantque (expression) **faire**
Instructions ;
finTantque ;
	While (expression) {
	Instructions ;
	}

Fonctionnement :

Les instructions de la boucle sont exécutées tant que la valeur de (expression) est vrai. L'exécution de l'algorithme continue après finTantque lorsque la valeur de (expression) devient faux. La boucle est très utile lorsque le nombre d'itération n'est pas connu à l'avance.

Ci-dessous l'implémentation de la boucle **tantque** en utilisant les étiquettes ainsi que sa modélisation graphique par un organigramme :

```
E:   Si (expression=vrai) alors
       Instructions ;
       Aller a E;
       Finsi;
```



Exemple :

Ci-dessous un algorithme qui calcule et affiche le chiffre maximum d'un entier N lue au clavier :

```
Algorithme chiffre_max;
Var n : entier ;
Debut
     Lire(n); m ← 0 ;
     Tantque(n <> 0)faire
       Si(n mod 10 > m)alors
         m ← n mod 10 ;
         n ← n div 10 ;
       finSi ;
     FinTantque;
     Ecrire('le chiffre max est:', m);
Fin.
```

3) La boucle repeter :

Syntaxe :

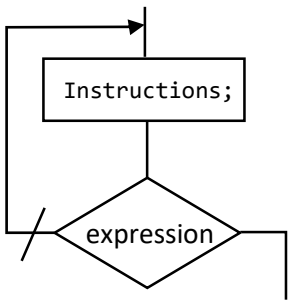
Algorithmique	Langage C
repeter Instructions ; **Jusqua(expression);**	do{ Instructions ; } While (expression);

Fonctionnement :

Contrairement à la boucle **tantque**, l'expression logique de la boucle **repeter** se trouve à la fin de la boucle. De ce fait, les instructions de la boucle sont exécutées au moins une fois. L'algorithme continu à exécuter les instructions de la boucle lorsque la valeur de (expression) est fausse. L'exécution de la boucle s'arrête dès que (expression) devient vrai.

Ci-dessous l'implémentation de la boucle **repeter** en utilisant les étiquettes ainsi que sa modélisation graphique par un organigramme :

```
E:   Instructions ;
       Si (expression=faux) alors
         Aller a E;
       Finsi;
```



Exemple :

Ci-dessous un algorithme qui calcule la somme d'une suite de nombre entier saisis au clavier, la saisie s'arrête lorsque l'utilisateur saisie la valeur 0.

Support de cours

Algorithme somme;

```
Var n : entier ;
Debut
     repeter
       lire(n);
     s ← s + n ;
     jusqua(n=0);
     Ecrire('s = ', s);
Fin.
```