Université Ahmed Zabana de Relizane

Spécialité : LMD MIAS

Année : 1ère (Semestre 1)

Module : Algorithmique et structure de données 1

Les structures de contrôle d'un algorithme :

Les structures itératives

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).

On distingue trois types de boucles courantes en C :

1. while
2. do. . . while
3. for

II. Types de Boucles

II.1 La boucle Pour :

Syntaxe :

POUR i ALLANT DE *valeur initiale* **à** *valeur finale* **FAIRE**
Bloc
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 1, le bloc sera exécuté.
- Le bloc est répéter *(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

Syntaxe en Langage C :

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

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.
- 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. par exemple).

Syntaxe :

```
int cpt;
for (cpt= initial ; cpt <= finale ; cpt=cpt+pas)
   {
        printf ("cpt vaut %d !\n", cpt);
   }
```

Propriétés en langage C :

1. La valeur initiale est affectée à la variable cpt ;
2. On compare la valeur du cpt et la valeur finale :
3. Si la condition est fausse, on sort de la boucle et on continue avec l'instruction qui suit l'accolade fermante.
4. Si la condition est vrai alors :
4.1 Les instructions de la boucle seront exécutées
4.2 Ensuite, la valeur de cpt est incrémentée de la valeur du pas (sinon 1 par défaut).
4.3 On recommence l'étape 2 : La comparaison entre cpt et finale est de nouveau effectuée, et ainsi de suite...

Exemple :

```
int compteur ;
for ( compteur = 0 ; compteur < 10 ; compteur ++)
{
        printf ("La variable compteur vaut %d !\n", compteur );
}
```

La plus part 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

II.2 La boucle Tant que : La boucle « While »

Syntaxe :

```
TANT QUE COND FAIRE
Bloc
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

Syntaxe en Langage C:

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

Propriétés en langage C :

- la condition (dite condition de contrôle de la boucle) 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 :

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 );
        }
```

II.3 La boucle Répéter : La boucle « do...while »

Syntaxe :

```
RÉPÉTER
Bloc
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 seul fois.

Exemple :

```
ALGORITHME TEST ;
VAR AGE : ENTIER ;
DÉBUT
RÉPÉTER
        ECRIRE ("ENTREZ SVP VOTRE AGE")
        LIRE (AGE)
JUSQU'À AGE >0
ECRIRE ("VOTRE AGE EST ", AGE)
FIN
```

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 :

```
int compteur = 0;
int somme = 0;
do
{
        somme =+ compteur   // equivalent a somme = somme + compteur
        compteur ++;
} while ( compteur <= 10);
```

II.4 Les boucles imbriqués

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

Exemple :

```
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

III. Quelle boucle puisse-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 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.