

Chapitre IV

LES STRUCTURES REPETITIVES

Objectifs

Général

⇒ Pouvoir écrire un programme utilisant les structures répétitives.

Spécifiques

- ❑ Comprendre les structures “*while*”, “*do – while*” et “*for*”;
 - ❑ Pouvoir faire le choix adéquat entre ces différentes structures dans le développement d'un programme C.
-

1. Introduction

Les boucles sont des structures qui permettent d'exécuter plusieurs fois la même série d'instructions jusqu'à ce qu'une condition ne soit plus réalisée. On appelle parfois ces structures *instructions répétitives* ou bien *itérations*. La façon la plus commune de faire une boucle, est de créer un compteur et de faire arrêter la boucle lorsque le compteur dépasse une certaine valeur.

En C, nous disposons de trois structures qui nous permettent la définition de boucles conditionnelles:

- 1) la structure : **while**
- 2) la structure : **do - while**
- 3) la structure : **for**

Théoriquement, ces structures sont interchangeables, c.-à-d. il serait possible de programmer toutes sortes de boucles conditionnelles en n'utilisant qu'une seule des trois structures.

2. La structure while

La structure **while** correspond tout à fait à la structure tant que du langage algorithmique.

La structure *tant que* en langage algorithmique

Tant que (<expression logique>) faire
| <bloc d'instructions>
Fin faire

- Tant que l'<expression logique> fournit la valeur *vrai*, le <bloc d'instructions> est exécuté.
- Si l'<expression logique> fournit la valeur *fausse*, l'exécution continue avec l'instruction qui suit *ftant*.
- Le <bloc d'instructions> est exécuté zéro ou plusieurs fois.

La structure **while** en C

while (<expression>)
 <bloc d'instructions>

- Tant que l'<expression> fournit une valeur différente de zéro, le <bloc d'instructions> est exécuté.
- Si l'<expression> fournit la valeur zéro, l'exécution continue avec l'instruction qui suit le bloc d'instructions.
- Le <bloc d'instructions> est exécuté zéro ou plusieurs fois.

La partie <expression> peut désigner :

- Une variable d'un type numérique,
- Une expression fournissant un résultat numérique.

La partie <bloc d'instructions> peut désigner :

- Un (vrai) bloc d'instructions compris entre accolades,
- Une seule instruction terminée par un point-virgule.

Exemple 1

```
/* Afficher les nombres de 0 à 9 */
int I = 0;
while (I<10)
{
    printf("%i \n", I); I++;
}
```

Exemple 2

```
int I;
/* Afficher les nombres de 0 à 9 */
I = 0;
while (I<10) printf("%i \n", I++);
```

Exemple 3

```
/* Afficher les nombres de 10 à 1 */
int I=10;
while (I) printf("%i \n", I--);
```

Exercice 4.1

Ecrivez un programme qui lit N nombres entiers au clavier et qui affiche leur somme, leur produit et leur moyenne. Choisissez un type approprié pour les valeurs à afficher. Le nombre N est à entrer au clavier. Résolvez ce problème, en utilisant **while**.

Solution

```
#include <stdio.h>
main( )
{
```

```
int N;      /* nombre de données */
int NOMB;   /* nombre courant   */
int I;      /* compteur */
long SOM;   /* la somme   des nombres entrés */
double PROD; /* le produit des nombres entrés */

printf("Nombre de données : ");
scanf("%d", &N);

SOM=0;
PROD=1;
I=1;
while(I<=N)
{
    printf("%d. nombre : ", I);
    scanf("%d", &NOMB);
    SOM += NOMB;
    PROD *= NOMB;
    I++;
}

printf("La somme   des %d nombres est %ld \n", N, SOM);
printf("Le produit des %d nombres est %.0f\n", N, PROD);
printf("La moyenne des %d nombres est %.4f\n", N, (float)SOM/N);
return 0;
}
```

3. La structure **do - while**

La structure **do - while** est semblable à la structure **while**, avec la différence suivante :

- **while** évalue la condition *avant* d'exécuter le bloc d'instructions,
- **do - while** évalue la condition *après* avoir exécuté le bloc d'instructions. Ainsi le bloc d'instructions est exécuté au moins une fois.

La structure **do - while** en C

```
do
    <bloc d'instructions>
while ( <expression> );
```

Le <bloc d'instructions> est exécuté au moins une fois et aussi longtemps que l'<expression> fournit une valeur différente de zéro.

En pratique, la structure **do - while** n'est pas si fréquente que **while**; mais dans certains cas, elle fournit une solution plus élégante. Une application typique de **do - while** est la saisie de données qui doivent remplir une certaine condition.

Exemple 1

```
float N;
do
{
    printf("Introduisez un nombre entre 1 et 10 :");
    scanf("%f", &N);
}
```

```
while (N<1 || N>10);
```

Exemple 2

```
int n, div;
printf("Entrez le nombre à diviser : ");
scanf("%d", &n);
do
{
    printf("Entrez le diviseur ( 0) : ");
    scanf("%d", &div);
}
while (!div);
printf("%d / %d = %f\n", n, div, (float)n/div);
```

do - while est comparable à la structure répéter du langage algorithmique *si la condition finale est inversée logiquement*.

Exemple 3

Le programme de calcul de la racine carrée :

programme RACINE_CARREE
réel N
répéter
écrire "Entrer un nombre (>=0) : "
lire N
jusqu'à (N >= 0)
écrire "La racine carrée de ",N,"vaut ", N
fprogramme (fin RACINE_CARREE *)*

se traduit en C par :

```
#include <stdio.h>
#include <math.h>
main( )
{
    float N;
    do
    {
        printf("Entrer un nombre (>= 0) : ");
        scanf("%f", &N);
    }
    while (N < 0);
    printf("La racine carrée de %f est %f\n", N, sqrt(N));
    return 0;
}
```

Exercice 4.2

Ecrivez un programme qui lit N nombres entiers au clavier et qui affiche leur somme, leur produit et leur moyenne. Choisissez un type approprié pour les valeurs à afficher. Le nombre N est à entrer au clavier. Résolvez ce problème, en utilisant **do - while**.

Solution

```
#include <stdio.h>
main( )
{
    int N;          /* nombre de données */
    int NOMB;       /* nombre courant   */
    int I;          /* compteur */
    long SOM;       /* la somme   des nombres entrés */
    double PROD;    /* le produit des nombres entrés */

    printf("Nombre de données : ");
    scanf("%d", &N);

    SOM=0;
    PROD=1;
    I=1;
    do
    {
        printf("%d. nombre : ", I);
        scanf("%d", &NOMB);
        SOM += NOMB;
        PROD *= NOMB;
        I++;
    }
    while(I<=N);

    printf("La somme   des %d nombres est %ld \n", N, SOM);
    printf("Le produit des %d nombres est %.0f\n", N, PROD);
    printf("La moyenne des %d nombres est %.4f\n", N, (float)SOM/N);
    return 0;
}
```

4. La structure for

La structure pour en langage algorithmique est utilisée pour faciliter la programmation de boucles de comptage. La structure **for** en C est plus générale et beaucoup plus puissante.

La structure for en C

for (<expr1> ; <expr2> ; <expr3>)
<bloc d'instructions>

est équivalent à :

```
<expr1>;
while ( <expr2> )
{
    <bloc d'instructions>
    <expr3>;
}
```

- **<expr1>** est évaluée une fois avant le passage de la boucle. Elle est utilisée pour initialiser les données de la boucle.
- **<expr2>** est évaluée avant chaque passage de la boucle. Elle est utilisée pour décider si la boucle est répétée ou non.
- **<expr3>** est évaluée à la fin de chaque passage de la boucle. Elle est utilisée pour réinitialiser les données de la boucle.

Le plus souvent, **for** est utilisé comme boucle de comptage :

```
for ( <init.> ; <cond. répétition> ; <compteur> )
    <bloc d'instructions>
```

Exemple 1

```
int I;
for (I=0 ; I<=20 ; I++) printf("Le carré de %d est %d \n", I, I*I);
```

En pratique, les parties **<expr1>** et **<expr2>** contiennent souvent plusieurs initialisations ou réinitialisations, *séparées par des virgules*.

Exemple 2

```
int n, tot;
for (tot=0, n=1 ; n<101 ; n++) tot+=n;
printf("La somme des nombres de 1 à 100 est %d\n", tot);
```

Exercice 4.3

Ecrivez un programme qui lit N nombres entiers au clavier et qui affiche leur somme, leur produit et leur moyenne. Choisissez un type approprié pour les valeurs à afficher. Le nombre N est à entrer au clavier. Résolvez ce problème, en utilisant **for**.

Solution

```
#include <stdio.h>
main( )
{
    int N;          /* nombre de données */
    int NOMB;       /* nombre courant   */
    int I;          /* compteur */
    long SOM;       /* la somme   des nombres entrés */
    double PROD;    /* le produit des nombres entrés */

    printf("Nombre de données : ");
    scanf("%d", &N);
    for (SOM=0, PROD=1, I=1 ; I<=N ; I++)
    {
        printf("%d. nombre : ", I);
        scanf("%d", &NOMB);
        SOM += NOMB;
        PROD *= NOMB;
    }
}
```

```
printf("La somme des %d nombres est %ld \n", N, SOM);
printf("Le produit des %d nombres est %.0f\n", N, PROD);
printf("La moyenne des %d nombres est %.4f\n", N,
(float)SOM/N);
return 0;
}
```

5. Choix de la structure répétitive

Dans ce chapitre, nous avons vu trois façons différentes de programmer des boucles (**while**, **do - while**, **for**).

Utiliser la structure qui reflète le mieux l'idée du programme que vous voulez réaliser, en respectant toutefois les directives suivantes :

- Si le bloc d'instructions ne doit pas être exécuté si la condition est fausse, alors utiliser **while** ou **for**,
- Si le bloc d'instructions doit être exécuté au moins une fois, alors utiliser **do – while**,
- Si le nombre d'exécutions du bloc d'instructions dépend d'une ou de plusieurs variables qui sont modifiées à la fin de chaque répétition, alors utiliser **for**,
- Si le bloc d'instructions doit être exécuté aussi longtemps qu'une condition extérieure est vraie, alors utiliser **while**.

Le choix entre **for** et **while** n'est souvent qu'une question de préférence ou d'habitudes:

- **for** nous permet de réunir avantageusement les instructions qui influencent le nombre de répétitions au début de la structure,
- **while** a l'avantage de correspondre plus exactement aux structures d'autres langages de programmation (**while**, **tant que**),
- **for** a le désavantage de favoriser la programmation de structures surchargées et par la suite illisibles,
- **while** a le désavantage de mener parfois à de longues structures, dans lesquelles il faut chercher pour trouver les instructions qui influencent la condition de répétition.

6. Exercices d'application

Exercice 4.4

Ecrire un programme C qui lit 20 notes et affiche la note maximale et celle minimale.

Exercice 4.5

Ecrire un programme C qui permet de lire 12 notes réelles et afficher le nombre de notes qui sont en dessous de la moyenne.

Exercice 4.6

Ecrire un programme C qui permet de lire une série de notes comprises entre 0 et 20, puis afficher la plus petite, la plus grande, le nombre de notes strictement inférieures, égales et strictement supérieures à 10.

Exercice 4.7

Ecrire un programme C qui détermine si un entier N est parfait ou non. Un entier est dit parfait s'il est égal à la somme de ses diviseurs stricts. Exemple : $6 = 1 + 2 + 3$.

Exercice 4.8

Ecrire un programme C qui demande un nombre N entier positif, puis calcule :

La somme $S1 = 1 + 2 + \dots + N$

La somme $S2 = 1^2 + 2^2 + \dots + N^2$

Le produit $P1 = 1 \times 2 \times \dots \times N$

Exercice 4.9

Ecrire un programme C qui permet de demander à l'utilisateur s'il est un homme ou une femme, puis le saluer (« Bonjour Monsieur » ou « Bonjour Madame ») en tenant compte de sa réponse. On exigera la réponse « H ou h » ou la réponse « f ou F ».
