Apprenez à programmer en C !

40 heures Moyenne

Mis à jour le 29/07/2019

Les boucles

Après avoir vu comment réaliser des conditions en C, nous allons découvrir les boucles. Qu'est-ce qu'une boucle ? C'est une technique permettant de répéter les mêmes instructions plusieurs fois. Cela nous sera bien utile par la suite, notamment pour le premier TP qui vous attend après ce chapitre.

Relaxez-vous : ce chapitre sera simple. Nous avons vu ce qu'étaient les conditions et les booléens dans le chapitre précédent, c'était un gros morceau à avaler. Maintenant ça va couler de source et le TP ne devrait pas vous poser trop de problèmes.

Enfin profitez-en, parce qu'ensuite nous ne tarderons pas à entrer dans la partie II du cours, et là vous aurez intérêt à être bien réveillés !

Qu'est-ce qu'une boucle ?

Je me répète : une boucle est une structure qui permet de répéter les mêmes instructions plusieurs fois.

Tout comme pour les conditions, il y a plusieurs façons de réaliser des boucles. Au bout du compte, cela revient à faire la même chose : répéter les mêmes instructions un certain nombre de fois.
Nous allons voir trois types de boucles courantes en C :

- `while`
- `do… while`
- `for`

Dans tous les cas, le schéma est le même (fig. suivante).

Voici ce qu'il se passe dans l'ordre :

1. l'ordinateur lit les instructions de haut en bas (comme d'habitude) ;
2. puis, une fois arrivé à la fin de la boucle, il repart à la première instruction ;
3. il recommence alors à lire les instructions de haut en bas...
4. ... et il repart au début de la boucle.

Le problème dans ce système c'est que si on ne l'arrête pas, l'ordinateur est capable de répéter les instructions à l'infini ! Il n'est pas du genre à se plaindre, vous savez : il fait ce qu'on lui dit de faire... Il pourrait très bien se bloquer dans une boucle infinie, c'est d'ailleurs une des nombreuses craintes des programmeurs.

Et c'est là qu'on retrouve... les conditions ! Quand on crée une boucle, on indique toujours une condition. Cette condition signifiera « Répète la boucle tant que cette condition est vraie ».

Comme je vous l'ai dit, il y a plusieurs manières de s'y prendre. Voyons voir sans plus tarder comment on réalise une boucle de type `while` en C.

La boucle while

Voici comment on construit une boucle `while` :

c

```c
while (/* Condition */)
{
    // Instructions à répéter
}
```

C'est aussi simple que cela. `while` signifie « Tant que ». On dit donc à l'ordinateur « Tant que la condition est vraie, répète les instructions entre accolades ».

Je vous propose de faire un test simple : on va demander à l'utilisateur de taper le nombre 47. Tant qu'il n'a pas tapé le nombre 47, on lui redemande le nombre. Le programme ne pourra s'arrêter que si l'utilisateur tape le nombre 47 (je sais, je sais, je suis diabolique) :

c

```c
int nombreEntre = 0;

while (nombreEntre != 47)
{
    printf("Tapez le nombre 47 ! ");
    scanf("%d", &nombreEntre);
```

```
7 }
```

Voici maintenant le test que j'ai fait. Notez que j'ai fait exprès de me tromper 2-3 fois avant de taper le bon nombre.

```
Tapez le nombre 47 ! 10
Tapez le nombre 47 ! 27
Tapez le nombre 47 ! 40
Tapez le nombre 47 ! 47
```

Le programme s'est arrêté après avoir tapé le nombre 47.
Cette boucle `while` se répète donc tant que l'utilisateur n'a pas tapé 47, c'est assez simple.

Maintenant, essayons de faire quelque chose d'un peu plus intéressant : on veut que notre boucle se répète un certain nombre de fois.
On va pour cela créer une variable `compteur` qui vaudra 0 au début du programme et que l'on va **incrémenter** au fur et à mesure. Vous vous souvenez de l'incrémentation ? Ça consiste à ajouter 1 à la variable en faisant `variable++;` .

Regardez attentivement ce bout de code et, surtout, essayez de le comprendre :

c

```c
int compteur = 0;

while (compteur < 10)
{
    printf("Salut les Zeros !\n");
    compteur++;
}
```

Résultat :

```
Salut les Zeros !
Salut les Zeros !
Salut les Zeros !
Salut les Zeros !
Salut les Zeros !
Salut les Zeros !
Salut les Zeros !
Salut les Zeros !
Salut les Zeros !
Salut les Zeros !
```

Ce code répète 10 fois l'affichage de « Salut les Zeros ! ».

Comment ça marche exactement ?

1. Au départ, on a une variable `compteur` initialisée à 0. Elle vaut donc 0 au début du programme.
2. La boucle `while` ordonne la répétition TANT QUE `compteur` est inférieur à 10. Comme `compteur` vaut 0 au départ, on rentre dans la boucle.
3. On affiche la phrase « Salut les Zeros ! » via un `printf`.
4. On **incrémente** la valeur de la variable `compteur`, grâce à `compteur++;`. `compteur` valait 0, elle vaut maintenant 1.
5. On arrive à la fin de la boucle (accolade fermante) : on repart donc au début, au niveau du `while`. On refait le test du `while` : « *Est-ce que `compteur` est toujours inférieure à 10 ?* ». Ben oui, `compteur` vaut 1 ! Donc on recommence les instructions de la boucle.

Et ainsi de suite... `compteur` va valoir progressivement 0,
1, 2, 3, ..., 8, 9, et 10. Lorsque `compteur` vaut 10, la condition `compteur < 10` est fausse. Comme l'instruction est fausse, on sort de la boucle.

On pourrait d'ailleurs voir que la variable `compteur` augmente au fur et à mesure dans la boucle, en l'affichant dans le `printf` :

c

```c
int compteur = 0;

while (compteur < 10)
{
    printf("La variable compteur vaut %d\n", compteur);
    compteur++;
}
```

```
La variable compteur vaut 0
La variable compteur vaut 1
La variable compteur vaut 2
La variable compteur vaut 3
La variable compteur vaut 4
La variable compteur vaut 5
La variable compteur vaut 6
La variable compteur vaut 7
La variable compteur vaut 8
La variable compteur vaut 9
```

Voilà : si vous avez compris ça, vous avez tout compris !
Vous pouvez vous amuser à augmenter la limite du nombre de boucles (`< 100` au lieu de `< 10`).
Cela m'aurait été d'ailleurs très utile plus jeune pour rédiger les punitions que je devais réécrire 100 fois.

Attention aux boucles infinies

Lorsque vous créez une boucle, **assurez-vous toujours qu'elle peut s'arrêter à un moment** ! Si la condition est toujours vraie, votre programme ne s'arrêtera jamais ! Voici un exemple de boucle infinie :

c

```c
while (1)
{
    printf("Boucle infinie\n");
}
```

Souvenez-vous des booléens : 1 = vrai, 0 = faux. Ici, la condition est toujours vraie, ce programme affichera donc « Boucle infinie » sans arrêt !

> Pour arrêter un tel programme sous Windows, vous n'avez pas d'autre choix que de fermer la console en cliquant sur la croix en haut à droite. Sous Linux, faites `Ctrl + C` .

Faites donc très attention : évitez à tout prix de tomber dans une boucle infinie. Notez toutefois que les boucles infinies peuvent s'avérer utiles, notamment, nous le verrons plus tard, lorsque nous réaliserons des jeux.

La boucle do... while

Ce type de boucle est très similaire à `while` , bien qu'un peu moins utilisé en général.
La seule chose qui change en fait 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 :

c

```c
int compteur = 0;

do
{
    printf("Salut les Zeros !\n");
    compteur++;
} while (compteur < 10);
```

Qu'est-ce que ça change ?
C'est très simple : la boucle `while` pourrait très bien ne jamais être exécutée si la condition est fausse dès le départ. Par exemple, si on avait initialisé le compteur à 50, la condition aurait été fausse dès le début et on ne serait jamais rentré dans la boucle.
Pour la boucle `do... while` , c'est différent : **cette boucle s'exécutera toujours au moins une fois**. En effet, le test se fait à la fin comme vous pouvez le voir. Si on initialise `compteur` à 50, la boucle s'exécutera une fois.

Il est donc parfois utile de faire des boucles de ce type, pour s'assurer que l'on rentre au moins une fois dans la boucle.

Il y a une particularité dans la boucle `do... while` qu'on a tendance à oublier quand on débute : il y a un point-virgule tout à la fin ! N'oubliez pas d'en mettre un après le `while`, sinon votre programme plantera à la compilation !

La boucle for

En théorie, la boucle `while` permet de réaliser toutes les boucles que l'on veut.
Toutefois, tout comme le `switch` pour les conditions, il est dans certains cas utile d'avoir un autre système de boucle plus « condensé », plus rapide à écrire.

Les boucles `for` sont très très utilisées en programmation. Je n'ai pas de statistiques sous la main, mais sachez que vous utiliserez certainement autant de `for` que de `while`, si ce n'est plus, il vous faudra donc savoir manipuler ces deux types de boucles.

Comme je vous le disais, les boucles `for` sont juste une autre façon de faire une boucle `while`.
Voici un exemple de boucle `while` que nous avons vu tout à l'heure :

c

```c
int compteur = 0;

while (compteur < 10)
{
    printf("Salut les Zeros !\n");
    compteur++;
}
```

Voici maintenant l'équivalent en boucle `for` :

c

```c
int compteur;

for (compteur = 0 ; compteur < 10 ; compteur++)
{
    printf("Salut les Zeros !\n");
}
```

Quelles différences ?

- Vous noterez que l'on n'a pas initialisé la variable `compteur` à 0 dès sa déclaration (mais on aurait pu le faire).
- Il y a beaucoup de choses entre les parenthèses après le `for` (nous allons détailler ça après).
- Il n'y a plus de `compteur++;` dans la boucle.

Intéressons-nous à ce qui se trouve entre les parenthèses, car c'est là que réside tout l'intérêt de la boucle `for`. 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`. Dans notre cas, on initialise la variable à 0.
- La seconde est **la condition** : comme pour la boucle `while`, 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`. La quasi-totalité 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).

Bref, comme vous le voyez la boucle `for` n'est rien d'autre qu'un condensé. Sachez vous en servir, vous en aurez besoin plus d'une fois !

En résumé

- Les **boucles** sont des structures qui nous permettent de répéter une série d'instructions plusieurs fois.
- Il existe plusieurs types de boucles : `while`, `do… while` et `for`. Certaines sont plus adaptées que d'autres selon les cas.
- La boucle `for` est probablement celle qu'on utilise le plus dans la pratique. On y fait très souvent des incrémentations ou des décrémentations de variables.

☐ J'AI TERMINÉ CE CHAPITRE ET JE PASSE AU SUIVANT

‹ LES CONDITIONS

TP : PLUS OU MOINS, VOTRE PREMIER JEU ›

Le professeur

Mathieu Nebra

Entrepreneur à plein temps, auteur à plein temps et co-fondateur d'OpenClassrooms :o)

Découvrez aussi ce cours en...

Livre PDF