IUT ANNECY UNIVERSITÉ SAVOIE MONT BLANC

Informatique

Sequence 2 : Boucles et fonctions

Cours 2.1

v2025-09-24

IUT d'Annecy, 9 rue de l'Arc en Ciel, 74940 Annecy

Boucles et premières fonctions

1 Boucles en C — while, for, do...while

1.1 La boucle While

Structure d'une boucle `while`

Le mot clef `while`signifie "tant que". Une boucle `while` répète un bloc d'instructions **tant qu'**une condition est vraie. La syntaxe est la suivante :

```
while (condition) {
    // bloc d'instructions à répéter
}
```

- La **condition** est une expression qui est évaluée avant chaque itération de la boucle. Si elle est vraie (non nulle), le bloc d'instructions est exécuté. Si elle est fausse (zéro), la boucle se termine.
- Le **bloc d'instructions** est le code qui sera répété tant que la condition est vraie.

Une fois la boucle terminé, le programme continue avec les instructions qui suivent la boucle.

1.1.1 Exemple rapide : Miauler 3 fois avec `while`

```
#include <stdio.h>
int main(void) {
    int i = 0;
    while (i < 3) {
        printf("Miaou !\n");
        i++;
    }
    return 0;
}
```

Ici, tant que `i` est inférieur à 3, on affiche "Miaou !" et on incrémente `i` de 1. Cela produit l'affichage suivant :

```
Miaou !
Miaou !
Miaou !
```

1.2 La boucle For

Une boucle `for` est une autre structure de boucle en C. Elle permet également de répéter un bloc d'instructions et est particulièrement utile lorsque le nombre d'itérations est connu à l'avance.

For Vs While

Il est totalement possible d'écrire le même programme avec une boucle `while` ou une boucle `for`. Le choix entre les deux dépend souvent de la préférence personnelle et du contexte du problème à résoudre. La boucle `for` est souvent adaptée pour compter un nombre fixe d'itérations, tandis que la boucle `while` est plus flexible pour des conditions basées sur des événements ou des états qui peuvent changer de manière imprévisible.

Structure d'une boucle `for`

La syntaxe d'une boucle `for` est la suivante :

```
for (initialisation; condition; mise_à_jour) {
    // bloc d'instructions à répéter
}
```

- Le **bloc d'instructions** est le code qui sera exécuté tant que la condition est vraie.
- La boucle `for` comprend trois parties dans ses parenthèses :

  initialisation : C'est l'endroit où vous initialisez une variable de compteur. Cette partie est exécutée une seule fois, au début de la boucle.

  condition : C'est une expression qui est évaluée avant chaque itération de la boucle. Si elle est vraie (non nulle), le bloc d'instructions est exécuté. Si elle est fausse (zéro), la boucle se termine.

  mise_à_jour : C'est l'endroit où vous mettez à jour la variable de compteur. Cette partie est exécutée à la fin de chaque itération de la boucle.

Exemple rapide : Aboyer 3 fois avec `for`

```
#include <stdio.h>
int main(void) {
    for (int i = 0; i < 3; i++) {
        printf("Ouaf !\n");
    }
    return 0;
}
```

Ici, la boucle `for` initialise `i` à 0, vérifie si `i` est inférieur à 3, et incrémente `i` de 1 après chaque itération.

Exemple de boucles imbriquées

Il est possible d'imbriquer des boucles, c'est-à-dire d'avoir une boucle à l'intérieur d'une autre. Cela est souvent utilisé pour parcourir des structures de données à plusieurs dimensions, comme des matrices.

```
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 2; j++) {
        printf("i = %d, j = %d\n", i, j);
    }
}
```

Exemple du cours - A compléter

1.3 La boucle Do...While

Une boucle `do...while` est une structure de boucle en C qui garantit que le bloc d'instructions sera exécuté au moins une fois, car la condition est évaluée après l'exécution du bloc.

Structure d'une boucle `do...while`

La syntaxe d'une boucle `do...while` est la suivante :

```
do {
    // bloc d'instructions à répéter
} while (condition);
```

- Le **bloc d'instructions** est le code qui sera exécuté au moins une fois.
- La **condition** est une expression qui est évaluée après chaque exécution du bloc. Si elle est vraie (non nulle), le bloc est répété. Si elle est fausse (zéro), la boucle se termine.

Exécutée au moins une fois

Contrairement aux boucles `while` et `for`, une boucle `do...while` garantit que le bloc d'instructions sera exécuté au moins une fois, même si la condition est fausse dès le départ. La condition est évaluée **après** l'exécution du bloc, ce qui signifie que le bloc sera toujours exécuté une première fois avant que la condition ne soit vérifiée.

2 Les fonctions void

Cette section se concentre sur les fonctions 'void', fontions qui produisent un effet sans retourner de valeur. Les fonctions d'affichages en sont des exemples typiques.

Une fonction
Une fonction sert à regrouper un ensemble d'instructions que l'on peut réutiliser plusieurs fois dans un programme. Cela permet de rendre le code plus lisible, plus modulaire et plus facile à maintenir.

2.1 Définir une fonction

Pour définir une fonction en C, on utilise la syntaxe suivante :

```
type_de_retour nom_de_fonction(paramètres) {
    // bloc d'instructions
}
```

- **type_de_retour** : C'est le type de donnée que la fonction renvoie après son exécution (par exemple, `int`, `float`, `char`, etc.). Si la fonction ne renvoie rien, on utilise le mot clef `void`.
- **nom_de_fonction** : C'est le nom que vous donnez à la fonction. Il doit être descriptif de ce que fait la fonction.
- **paramètres** : Ce sont les variables que la fonction peut prendre en entrée pour effectuer son travail. Elles sont optionnelles. Si la fonction ne prend pas de paramètres, on laisse les parenthèses vides.
- **bloc d'instructions** : C'est le code qui sera exécuté lorsque la fonction est appelée.

2.1.1 Exemple rapide : Une fonction void qui affiche un message de bienvenue

```
#include <stdio.h>

void afficher_bienvenue(); // Déclaration de la fonction

int main(void) {
    afficher_bienvenue(); // Appel de la fonction
    return 0;
}

void afficher_bienvenue() { // Définition de la fonction
    printf("Bienvenue dans le monde de la programmation en C !\n");
}
```

Trois choses importantes à noter dans cet exemple :

1. La fonction `afficher_bienvenue` est déclarée avant son utilisation dans `main`. Cela informe le compilateur qu'une fonction avec ce nom et cette signature existe.
2. La fonction est définie après `main` – mais elle pourrait aussi être définie avant.
3. La fonction est **appelée** dans `main` en utilisant son nom suivi de parenthèses.

2.1.2 Exemple du cours : Fonction qui affiche une ligne de `n` blocs