C - Les conditions

Dans la programmation en C, les conditions permettent de contrôler le comportement de votre programme en fonction des événements ou des valeurs mesurées. Elles permettent de prendre des décisions et de déclencher des actions différentes selon les situations. Dans cet article, nous allons explorer les instructions conditionnelles `if`, `else if`, et `else`, ainsi que les opérateurs logiques `&&` (ET) et `||` (OU), pour vous aider à mieux gérer les flux d'exécution dans vos projets.

5TTR 6TTR Découverte

Les Bases des Conditions : if, else if, et else

Utilisation de `if`

L'instruction `if` permet de tester une condition et d'exécuter un bloc de code seulement si cette condition est vraie. Si la condition est fausse, le programme ignore le bloc de code.

Syntaxe de base :

```c
if (condition) {
    // Code à exécuter si la condition est vraie
}
```

Exemple : Réussir son interro.

```c

```

Utilisation de `else`

L'instruction `else` est un cas de repli qui s'exécute si aucune des conditions précédentes (`if` ou `else if`) n'est vraie. Elle est souvent utilisée pour définir une action par défaut.

Syntaxe de base :

```c
if (condition1) {
    // Code si condition1 est vraie
} else if (condition2) {
    // Code si condition2 est vraie
} else {
```

```
    // Code si aucune condition n'est vraie
}
```

Exemple:

```c
#include <stdio.h>

void main() {
    int note = 13;

    if (note >= 10) {
        printf("Bravo, tu as réussi !\n");
    }
    else {
        printf("Dommage, tu as échoué.\n");
    }

}
```

Utilisation de `else if`

L'instruction `else if` permet de tester une condition supplémentaire si la première condition (`if`) est fausse. Vous pouvez chaîner plusieurs `else if` pour tester différentes conditions.

Syntaxe de base :

```c
#include <stdio.h>

void main() {
    int nombre = 1;

    if (nombre > 0) {
        printf("Le nombre est positif.\n");
    }
    else if (nombre < 0) {
        printf("Le nombre est négatif.\n");
    }
    else {
        printf("Le nombre est nul.\n");
    }

}
```

Opérateurs Logiques : `&&` (ET) et `||` (OU)

Pour combiner plusieurs conditions dans une même instruction `if`, vous pouvez utiliser les opérateurs logiques `&&` (ET) et `||` (OU).

Utilisation de `&&` (ET)

L'opérateur `&&` est utilisé lorsque vous voulez que **plusieurs conditions soient vraies en même temps** pour exécuter le code.

Exemple : vérifier si un nombre est compris entre 0 et 100

```c
#include <stdio.h>

int main() {
    int nombre;

    printf("Entre un nombre : ");
    scanf("%d", &nombre);

    if (nombre >= 0 && nombre <= 100) {
        printf("Le nombre est entre 0 et 100.\n");
    }
    else {
        printf("Le nombre n'est pas dans l'intervalle.\n");
    }

    return 0;
}
```

Utilisation de `||` (OU)

L'opérateur `||` est utilisé lorsque vous voulez que **au moins une des conditions soit vraie** pour exécuter le code.

```c
#include <stdio.h>

int main() {
    int nombre;

    printf("Entre un nombre : ");
    scanf("%d", &nombre);

    if (nombre < 0 || nombre > 100) {
        printf("Le nombre est en dehors de l'intervalle 0-100.\n");
    }
    else {
        printf("Le nombre est entre 0 et 100.\n");
    }

    return 0;
}

```

Explication

- L'expression `(nombre < 0 || nombre > 100)` est **vraie** si le nombre est plus petit que 0 **ou** plus grand que 100.
- Si le nombre est 50 → faux (dans l'intervalle).

- Si le nombre est -5 → vrai.
- Si le nombre est 150 → vrai.

Utiliser des parenthèses

> 💡 **ASTUCE :** Utilise toujours des **parenthèses** pour rendre les conditions plus lisibles :

```
1 | if ((a > 0) || (b < 10))
```

Conclusion

Les conditions `if`, `else if`, `else`, ainsi que les opérateurs logiques `&&` et `||`, sont essentiels pour prendre des décisions dans vos programmes. En combinant ces structures de contrôle, vous pouvez créer des comportements complexes et adaptés aux situations de vos projets, qu'il s'agisse de vérifier plusieurs capteurs ou de déclencher des actions en fonction d'événements spécifiques. Prendre le temps de bien comprendre ces concepts est crucial pour développer des programmes robustes et fonctionnels.