Les instructions conditionnelles

En C++, les conditions permettent de contrôler l'exécution d'un programme en fonction de la valeur de certaines expressions. Les conditions sont généralement utilisées avec les instructions `if`, `else if` et `else` pour exécuter des blocs de code en fonction de la valeur de vérité d'une expression.

Voici un exemple simple qui utilise une condition pour vérifier si un nombre est pair ou impair :

```
#include <iostream>
using namespace std;

int main() {
    int x;
    cout << "Entrez un nombre entier: ";
    cin >> x;

    if (x % 2 == 0) {
        cout << x << " est un nombre pair." << endl;
    } else {
        cout << x << " est un nombre impair." << endl;
    }

    return 0;
}
```

Dans cet exemple, nous demandons à l'utilisateur d'entrer un nombre entier, puis nous utilisons l'opérateur modulo (`%`) pour vérifier si ce nombre est divisible par 2. Si c'est le cas, nous affichons un message indiquant que le nombre est pair, sinon nous affichons un message indiquant qu'il est impair.

En plus des instructions `if`, `else if` et `else`, C++ fournit également l'instruction `switch` pour effectuer des tests multiples sur une seule variable. Voici un exemple qui utilise l'instruction `switch` pour afficher le nom du jour de la semaine en fonction d'un nombre entier saisi par l'utilisateur :

```cpp
#include <iostream>
using namespace std;

int main() {
    int jour;
    cout << "Entrez un nombre entre 1 et 7: ";
    cin >> jour;

    switch (jour) {
        case 1:
            cout << "Lundi" << endl;
            break;
        case 2:
            cout << "Mardi" << endl;
            break;
        case 3:
            cout << "Mercredi" << endl;
            break;
        case 4:
            cout << "Jeudi" << endl;
            break;
        case 5:
            cout << "Vendredi" << endl;
            break;
        case 6:
            cout << "Samedi" << endl;
            break;
        case 7:
            cout << "Dimanche" << endl;
            break;
        default:
            cout << "Nombre invalide!" << endl;
    }

    return 0;
}
```

Dans cet exemple, nous demandons à l'utilisateur d'entrer un nombre entier entre 1 et 7, puis nous utilisons l'instruction `switch` pour afficher le nom du jour de la semaine correspondant. Si l'utilisateur entre un nombre qui n'est pas compris entre 1 et 7, nous affichons un message d'erreur.

Les opérateurs de comparaison

Les opérateurs de comparaison en C++ sont utilisés pour comparer deux valeurs. Il existe six opérateurs de comparaison en C++ :

- `==` (égal à) : renvoie `true` si les deux opérandes sont égales, sinon renvoie `false`.
- `!=` (différent de) : renvoie `true` si les deux opérandes sont différentes, sinon renvoie `false`.
- `<` (inférieur à) : renvoie `true` si l'opérande de gauche est inférieure à l'opérande de droite, sinon renvoie `false`.

- `>` (supérieur à) : renvoie `true` si l'opérande de gauche est supérieure à l'opérande de droite, sinon renvoie `false`.
- `<=` (inférieur ou égal à) : renvoie `true` si l'opérande de gauche est inférieure ou égale à l'opérande de droite, sinon renvoie `false`.
- `>=` (supérieur ou égal à) : renvoie `true` si l'opérande de gauche est supérieure ou égale à l'opérande de droite, sinon renvoie `false`.

Voici un exemple qui utilise les opérateurs de comparaison pour vérifier si un nombre est compris entre deux autres nombres :

```
#include <iostream>
using namespace std;

int main() {
    int x = 5;
    int a = 3;
    int b = 7;

    if (x >= a && x <= b) {
        cout << x << " est compris entre " << a << " et " << b << endl;
    } else {
        cout << x << " n'est pas compris entre " << a << " et " << b << endl;
    }

    return 0;
}
```

Dans cet exemple, nous utilisons les opérateurs de comparaison `>=` et `<=` pour vérifier si la variable `x` est comprise entre les variables `a` et `b`. Si c'est le cas, nous affichons un message indiquant que `x` est compris entre `a` et `b`, sinon nous affichons un message indiquant que ce n'est pas le cas.

```cpp
#include <iostream>
using namespace std;

int main() {
    int x;
    cout << "Entrez un nombre entier: ";
    cin >> x;

    if (x % 2 == 0) {
        cout << x << " est un nombre pair." << endl;
    } else {
        cout << x << " est un nombre impair." << endl;
    }

    return 0;
}
```

Dans cet exemple, nous demandons à l'utilisateur d'entrer un nombre entier, puis nous utilisons l'opérateur modulo (`%`) pour vérifier si ce nombre est divisible par 2. Si c'est le cas, nous utilisons l'instruction `if` pour afficher un message indiquant que le nombre est pair. Sinon, nous utilisons l'instruction `else` pour afficher un message indiquant que le nombre est impair.

L'instruction `else` est utilisée pour exécuter un bloc de code si la condition de l'instruction `if` précédente est fausse. Dans cet exemple, si la condition `x % 2 == 0` est fausse (c'est-à-dire si `x` n'est pas divisible par 2), alors le bloc de code associé à l'instruction `else` sera exécuté et le message "x est un nombre impair" sera affiché.

➢ Else if

```cpp
#include <iostream>
using namespace std;

int main() {
    int note;
    cout << "Entrez une note entre 0 et 100: ";
    cin >> note;

    if (note >= 90) {
        cout << "Excellent!" << endl;
    } else if (note >= 80) {
        cout << "Très bien!" << endl;
    } else if (note >= 70) {
        cout << "Bien!" << endl;
    } else if (note >= 60) {
        cout << "Assez bien!" << endl;
    } else {
        cout << "Peut mieux faire!" << endl;
    }

    return 0;
}
```

Dans cet exemple, nous demandons à l'utilisateur d'entrer une note entre 0 et 100, puis nous utilisons plusieurs instructions `else if` pour afficher un message en fonction de la note saisie. Si la note est supérieure ou égale à 90, nous affichons le message "Excellent!", sinon si elle est supérieure ou égale à 80, nous affichons le message "Très bien!", et ainsi de suite. Si aucune des conditions précédentes n'est vraie, nous utilisons l'instruction `else` pour afficher le message "Peut mieux faire!".

L'instruction `else if` est utilisée pour tester plusieurs conditions en cascade. Dans cet exemple, si la première condition (`note >= 90`) est fausse, alors la condition suivante (`note >= 80`) sera testée, et ainsi de suite jusqu'à ce qu'une condition soit vraie ou que toutes les conditions aient été testées.

➢ if avec des booléens

Voici un exemple simple qui utilise des conditions `true` et `false` en C++ :

```cpp
#include <iostream>
using namespace std;

int main() {
    bool condition = true;

    if (condition) {
        cout << "La condition est vraie." << endl;
    } else {
        cout << "La condition est fausse." << endl;
    }

    condition = false;

    if (condition) {
        cout << "La condition est vraie." << endl;
    } else {
        cout << "La condition est fausse." << endl;
    }

    return 0;
}
```

Dans cet exemple, nous déclarons une variable booléenne `condition` et lui affectons la valeur `true`. Nous utilisons ensuite l'instruction `if` pour vérifier si la valeur de `condition` est `true`. Si c'est le cas, nous affichons le message "La condition est vraie.", sinon nous utilisons l'instruction `else` pour afficher le message "La condition est fausse.".

Ensuite, nous modifions la valeur de `condition` pour lui affecter la valeur `false`, puis nous répétons le même test avec l'instruction `if`. Cette fois, comme la valeur de `condition` est `false`, c'est le bloc de code associé à l'instruction `else` qui sera exécuté et le message "La condition est fausse." sera affiché.

➢ Condition “and” , “or”, “not”

Voici un exemple qui utilise les opérateurs logiques `&&` (et), `||` (ou) et `!` (non) pour combiner des conditions en C++ :

```cpp
#include <iostream>
using namespace std;

int main() {
    int x = 5;
    int y = 10;

    if (x > 0 && y > 0) {
        cout << "x et y sont tous les deux positifs." << endl;
    }

    if (x > 0 || y > 0) {
        cout << "Au moins l'un des nombres x et y est positif." << endl;
    }

    if (!(x > 0)) {
        cout << "x n'est pas positif." << endl;
    }

    return 0;
}
```

Dans cet exemple, nous utilisons l’opérateur logique `&&` pour vérifier si les variables `x` et `y` sont toutes les deux positives. Si c’est le cas, nous affichons un message indiquant que “x et y sont tous les deux positifs.”.

Ensuite, nous utilisons l’opérateur logique `||` pour vérifier si au moins l’une des variables `x` ou `y` est positive. Si c’est le cas, nous affichons un message indiquant que “Au moins l’un des nombres x et y est positif.”.

Enfin, nous utilisons l’opérateur logique `!` pour vérifier si la variable `x` n’est pas positive. Si c’est le cas, nous affichons un message indiquant que “x n’est pas positif.”.

Exemple complexe de conditions combinées

Voici un exemple complexe qui utilise plusieurs conditions combinées avec les opérateurs logiques `||` (ou) et `!` (non) en C++ :

```cpp
#include <iostream>
using namespace std;

int main() {
    int age;
    bool etudiant, retraite;

    cout << "Entrez votre âge: ";
    cin >> age;

    cout << "Êtes-vous étudiant? (1 pour oui, 0 pour non): ";
    cin >> etudiant;

    cout << "Êtes-vous à la retraite? (1 pour oui, 0 pour non): ";
    cin >> retraite;

    if ((age < 18 || age >= 65) || etudiant || retraite) {
        cout << "Vous êtes éligible à un tarif réduit." << endl;
    } else {
        cout << "Vous n'êtes pas éligible à un tarif réduit." << endl;
    }

    return 0;
}
```

Dans cet exemple, nous demandons à l'utilisateur d'entrer son âge, puis de répondre à deux questions pour savoir s'il est étudiant ou à la retraite. Nous utilisons ensuite une condition complexe combinant plusieurs conditions avec les opérateurs logiques `||` et `!` pour vérifier si l'utilisateur est éligible à un tarif réduit.

La condition `(age < 18 || age >= 65) || etudiant || retraite` vérifie si l'âge de l'utilisateur est inférieur à 18 ans ou supérieur ou égal à 65 ans, ou si l'utilisateur est étudiant ou à la retraite. Si l'une de ces conditions est vraie, alors l'utilisateur est éligible à un tarif réduit et nous affichons un message en conséquence. Sinon, nous affichons un message indiquant que l'utilisateur n'est pas éligible à un tarif réduit.

Résumé

Les conditions en programmation permettent de tester la valeur des variables et d'exécuter des instructions spécifiques en fonction du résultat de ces tests.

La structure conditionnelle la plus couramment utilisée est la structure `if` ... `else if` ... `else`, qui permet de tester plusieurs conditions en cascade.

En utilisant des opérateurs logiques tels que `&&` (et), `||` (ou) et `!` (non), il est possible de combiner plusieurs conditions pour créer des tests complexes.