

Branchements conditionnels

Jusqu'ici, toutes les instructions des programmes étaient exécutées, et les unes après les autres.

Les **structures de contrôle** permettent de changer ce comportement.

Il y a 3 structures de contrôle:

- les branchements conditionnels,
- les itérations, et
- les boucles conditionnelles.

Nous allons commencer par les **branchements conditionnels**, qui permet de sauter certaines parties du programme si certaines conditions sont remplies, et qui utilisent le mot-clé `if`.

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

int main()
{
    int n;

    cout << "Entrez votre nombre:" << endl;
    cin >> n;

    if (n < 5) {
        cout << "Votre nombre est plus petit que 5." << endl;
    } else {
        cout << "Votre nombre est plus grand ou egal a 5." << endl;
    }

    cout << "Au revoir" << endl;

    return 0;
}
```

```
if (n < 5) {
    cout << "Votre nombre est plus petit que 5." << endl;
} else {
    cout << "Votre nombre est plus grand ou egal a 5." << endl;
}
```

Mot-clé `if`

```
if (n < 5) {  
    cout << "Votre nombre est plus petit que 5." << endl;  
} else {  
    cout << "Votre nombre est plus grand ou egal a 5." << endl;  
}
```

Condition

```
if (n < 5) {  
    cout << "Votre nombre est plus petit que 5." << endl;  
} else {  
    cout << "Votre nombre est plus grand ou egal a 5." << endl;  
}
```

Une accolade ouvrante

```
if (n < 5) {  
    cout << "Votre nombre est plus petit que 5." << endl;  
} else {  
    cout << "Votre nombre est plus grand ou egal a 5." << endl;  
}
```

Cette instruction sera exécutée si la condition est vraie.

```
if (n < 5) {  
    cout << "Votre nombre est plus petit que 5." << endl;  
} else {  
    cout << "Votre nombre est plus grand ou egal a 5." << endl;  
}
```

Une accolade fermante

```
if (n < 5) {  
    cout << "Votre nombre est plus petit que 5." << endl;  
}  
else {  
    cout << "Votre nombre est plus grand ou egal a 5." << endl;  
}
```

Le mot-clé else

```
if (n < 5) {  
    cout << "Votre nombre est plus petit que 5." << endl;  
} else {  
    cout << "Votre nombre est plus grand ou egal a 5." << endl;  
}
```

Une accolade ouvrante

```
if (n < 5) {  
    cout << "Votre nombre est plus petit que 5." << endl;  
} else {  
    cout << "Votre nombre est plus grand ou egal a 5." << endl;  
}
```

```
if (n < 5) {  
    cout << "Votre nombre est plus petit que 5." << endl;  
} else {  
    cout << "Votre nombre est plus grand ou egal a 5." << endl;  
}
```

Cette instruction sera exécutée si la condition est *fausse*.

```

if (n < 5) {
    cout << "Votre nombre est plus petit que 5." << endl;
} else {
    cout << "Votre nombre est plus grand ou egal a 5." << endl;
}

```



Une accolade fermante

```

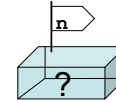
→ int n;

cout << "Entrez votre nombre:" << endl;
cin >> n;

if (n < 5) {
    cout << "Votre nombre est plus petit que 5." << endl;
} else {
    cout << "Votre nombre est plus grand ou egal a 5." << endl;
}

cout << "Au revoir" << endl;

```



Ce qui s'affiche dans la fenêtre Terminal:

```

|

```

```

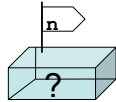
int n;

→ cout << "Entrez votre nombre:" << endl;
cin >> n;

if (n < 5) {
    cout << "Votre nombre est plus petit que 5." << endl;
} else {
    cout << "Votre nombre est plus grand ou egal a 5." << endl;
}

cout << "Au revoir" << endl;

```



Ce qui s'affiche dans la fenêtre Terminal:

Entrez votre nombre:

```

|

```

```

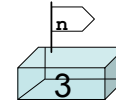
int n;

cout << "Entrez votre nombre:" << endl;
→ cin >> n;

if (n < 5) {
    cout << "Votre nombre est plus petit que 5." << endl;
} else {
    cout << "Votre nombre est plus grand ou egal a 5." << endl;
}

cout << "Au revoir" << endl;

```



Ce qui s'affiche dans la fenêtre Terminal:

Entrez votre nombre:

```

3
|

```

```
int n;
```

```
cout << "Entrez votre nombre:" << endl;  
cin >> n;
```

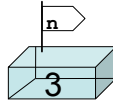
```
    ?  
→ if (n < 5) {  
    cout << "Votre nombre est plus petit que 5." << endl;  
    } else {  
    cout << "Votre nombre est plus grand ou egal a 5." << endl;  
    }  
  
    cout << "Au revoir" << endl;
```

Ce qui s'affiche dans la fenêtre Terminal:

Entrez votre nombre:

3

|



```
int n;
```

```
cout << "Entrez votre nombre:" << endl;  
cin >> n;
```

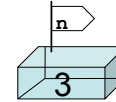
```
    if (n < 5) {  
→    cout << "Votre nombre est plus petit que 5." << endl;  
    } else {  
    cout << "Votre nombre est plus grand ou egal a 5." << endl;  
    }  
  
    cout << "Au revoir" << endl;
```

Ce qui s'affiche dans la fenêtre Terminal:

Entrez votre nombre:

3

|



```
int n;
```

```
cout << "Entrez votre nombre:" << endl;  
cin >> n;
```

```
    if (n < 5) {  
→    cout << "Votre nombre est plus petit que 5." << endl;  
    } else {  
    cout << "Votre nombre est plus grand ou egal a 5." << endl;  
    }  
  
    cout << "Au revoir" << endl;
```

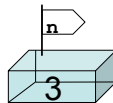
Ce qui s'affiche dans la fenêtre Terminal:

Entrez votre nombre:

3

Votre nombre est plus petit que 5.

|



```
int n;
```

```
cout << "Entrez votre nombre:" << endl;  
cin >> n;
```

```
    if (n < 5) {  
    cout << "Votre nombre est plus petit que 5." << endl;  
    } else {  
    cout << "Votre nombre est plus grand ou egal a 5." << endl;  
    }  
  
→    cout << "Au revoir" << endl;
```

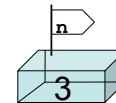
Ce qui s'affiche dans la fenêtre Terminal:

Entrez votre nombre:

3

Votre nombre est plus petit que 5.

|



```
int n;
```

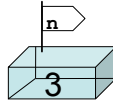
```
cout << "Entrez votre nombre:" << endl;  
cin >> n;
```

```
if (n < 5) {  
    cout << "Votre nombre est plus petit que 5." << endl;  
} else {  
    cout << "Votre nombre est plus grand ou egal a 5." << endl;  
}
```

```
→ cout << "Au revoir" << endl;
```

Ce qui s'affiche dans la fenêtre Terminal:

```
Entrez votre nombre:  
3  
Votre nombre est plus petit que 5.  
Au revoir  
|
```



```
int n;
```

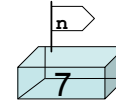
```
cout << "Entrez votre nombre:" << endl;  
cin >> n;
```

```
→ if (n < 5) {  
    cout << "Votre nombre est plus petit que 5." << endl;  
} else {  
    cout << "Votre nombre est plus grand ou egal a 5." << endl;  
}
```

```
cout << "Au revoir" << endl;
```

Ce qui s'affiche dans la fenêtre Terminal:

```
Entrez votre nombre:  
7  
|
```



```
int n;
```

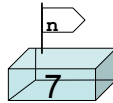
```
cout << "Entrez votre nombre:" << endl;  
cin >> n;
```

```
→ if (n < 5) {  
    cout << "Votre nombre est plus petit que 5." << endl;  
} else {  
    cout << "Votre nombre est plus grand ou egal a 5." << endl;  
}
```

```
cout << "Au revoir" << endl;
```

Ce qui s'affiche dans la fenêtre Terminal:

```
Entrez votre nombre:  
7  
|
```



```
int n;
```

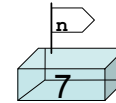
```
cout << "Entrez votre nombre:" << endl;  
cin >> n;
```

```
if (n < 5) {  
    cout << "Votre nombre est plus petit que 5." << endl;  
} else {  
    → cout << "Votre nombre est plus grand ou egal a 5." << endl;  
}
```

```
cout << "Au revoir" << endl;
```

Ce qui s'affiche dans la fenêtre Terminal:

```
Entrez votre nombre:  
7  
|
```

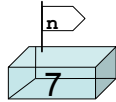


```
int n;

cout << "Entrez votre nombre:" << endl;
cin >> n;

if (n < 5) {
    cout << "Votre nombre est plus petit que 5." << endl;
} else {
    cout << "Votre nombre est plus grand ou egal a 5." << endl;
}

→ cout << "Au revoir" << endl;
```



Ce qui s'affiche dans la fenêtre Terminal:

```
Entrez votre nombre:
7
Au revoir
█
```

```
if (n < 5) {
    cout << "Votre nombre est plus petit que 5." << endl;
} else {
    cout << "Votre nombre est plus grand ou egal a 5." << endl;
}
```

Les accolades délimitent un bloc d'instructions

```
if (n < 5) {
    cout << "Votre nombre est plus petit que 5." << endl;
} else {
    cout << "Votre nombre est plus grand ou egal a 5." << endl;
}
```

Les accolades délimitent un bloc d'instructions

```
if (n < 5) {
    cout << "Votre nombre est plus petit que 5." << endl;
} else {
    cout << "Votre nombre est plus grand ou egal a 5." << endl;
}
```

On peut mettre autant d'instructions qu'on veut dans un bloc.
Supposons qu'on veuille aussi afficher la valeur de `n` quand `n` est plus petit que 5.
Il suffit d'ajouter une instruction dans le premier bloc:

```
if (n < 5) {  
    cout << "Votre nombre est plus petit que 5." << endl;  
    cout << "Votre nombre est " << n << endl;  
} else {  
    cout << "Votre nombre est plus grand ou egal a 5." << endl;  
}
```

Quand un bloc contient une seule instruction, il n'est pas obligatoire d'utiliser des accolades.
On aurait pu écrire:

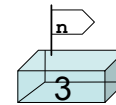
```
if (n < 5)  
    cout << "Votre nombre est plus petit que 5." << endl;  
else  
    cout << "Votre nombre est plus grand ou egal a 5." << endl;
```

Néanmoins, une bonne pratique est de **toujours utiliser des blocs**, même quand il n'y a qu'une seule instruction.
Ca facilite l'ajout d'instructions.

Une instruction `if` peut ne pas avoir de deuxième partie.
Par exemple, si on veut ne rien afficher si `n` est plus grand ou égal à 5, il suffit d'enlever la deuxième partie, à partir du `else`:

```
if (n < 5) {  
    cout << "Votre nombre est plus petit que 5." << endl;  
}
```

```
int n;  
  
cout << "Entrez votre nombre:" << endl;  
cin >> n;
```



```
→ if (n < 5) {  
    cout << "Votre nombre est plus petit que 5." << endl;  
}  
  
cout << "Au revoir" << endl;
```

Ce qui s'affiche dans la fenêtre Terminal:

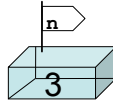
```
Entrez votre nombre:  
3  
|
```



```
int n;
```

```
cout << "Entrez votre nombre:" << endl;  
cin >> n;
```

```
if (n < 5) {  
→ cout << "Votre nombre est plus petit que 5." << endl;  
}  
  
cout << "Au revoir" << endl;
```



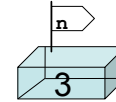
Ce qui s'affiche dans la fenêtre Terminal:

```
Entrez votre nombre:  
3  
Votre nombre est plus petit que 5.  
|
```

```
int n;
```

```
cout << "Entrez votre nombre:" << endl;  
cin >> n;
```

```
if (n < 5) {  
    cout << "Votre nombre est plus petit que 5." << endl;  
}  
  
→ cout << "Au revoir" << endl;
```



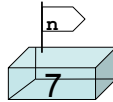
Ce qui s'affiche dans la fenêtre Terminal:

```
Entrez votre nombre:  
3  
Votre nombre est plus petit que 5.  
Au revoir  
|
```

```
int n;
```

```
cout << "Entrez votre nombre:" << endl;  
cin >> n;
```

```
→ if (n < 5) {  
    cout << "Votre nombre est plus petit que 5." << endl;  
}  
  
cout << "Au revoir" << endl;
```



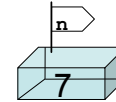
Ce qui s'affiche dans la fenêtre Terminal:

```
Entrez votre nombre:  
7  
|
```

```
int n;
```

```
cout << "Entrez votre nombre:" << endl;  
cin >> n;
```

```
if (n < 5) {  
    cout << "Votre nombre est plus petit que 5." << endl;  
}  
  
→ cout << "Au revoir" << endl;
```



Ce qui s'affiche dans la fenêtre Terminal:

```
Entrez votre nombre:  
7  
Au revoir  
|
```

Les choix imbriqués

L'instruction `if` suit donc le schéma:

```
if (condition1) {  
    ...  
} else {  
    ...  
}
```

Les instructions figurant dans les blocs sont absolument quelconques. *Il peut donc s'agir d'autres instructions `if`.*

Choix imbriqués: exemple

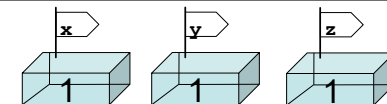
```
if (x == y) {  
    if (y == z) {  
        cout << "Les trois valeurs sont egales." << endl;  
    } else {  
        cout << "Seules les deux premieres valeurs sont egales." << endl;  
    }  
} else {  
    if (x == z) {  
        cout << "Seules la premiere et la troisieme valeurs sont egales." << endl;  
    } else {  
        if (y == z) {  
            cout << "Seules les deux dernieres valeurs sont egales." << endl;  
        } else {  
            cout << "Les trois valeurs sont differentes." << endl;  
        }  
    }  
}
```

Choix imbriqués: exemple

```
if (x == y) {  
    if (y == z) {  
        cout << "Les trois valeurs sont egales." << endl;  
    } else {  
        cout << "Seules les deux premieres valeurs sont egales." << endl;  
    }  
} else {  
    if (x == z) {  
        cout << "Seules la premiere et la troisieme valeurs sont egales." << endl;  
    } else {  
        if (y == z) {  
            cout << "Seules les deux dernieres valeurs sont egales." << endl;  
        } else {  
            cout << "Les trois valeurs sont differentes." << endl;  
        }  
    }  
}
```

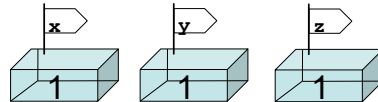
Attention à ne pas abuser de cette solution.
Au-delà de 3 niveaux, le code devient vite illisible!

Supposons:



```
→ if (x == y) {  
    if (y == z) {  
        cout << "Les trois valeurs sont egales." << endl;  
    } else {  
        cout << "Seules les deux premieres valeurs sont egales." << endl;  
    }  
} else {  
    if (x == z) {  
        cout << "Seules la premiere et la troisieme valeurs sont egales." << endl;  
    } else {  
        if (y == z) {  
            cout << "Seules les deux dernieres valeurs sont egales." << endl;  
        } else {  
            cout << "Les trois valeurs sont differentes." << endl;  
        }  
    }  
}
```

Supposons:

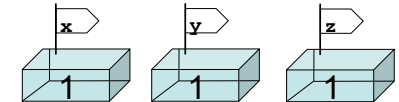


```

if (x == y) {
→ if (y == z) {
    cout << "Les trois valeurs sont egales." << endl;
  } else {
    cout << "Seules les deux premieres valeurs sont egales." << endl;
  }
} else {
  if (x == z) {
    cout << "Seules la premiere et la troisieme valeurs sont egales." << endl;
  } else {
    if (y == z) {
      cout << "Seules les deux dernieres valeurs sont egales." << endl;
    } else {
      cout << "Les trois valeurs sont differentes." << endl;
    }
  }
}

```

Supposons:



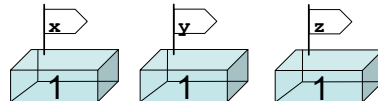
```

if (x == y) {
  if (y == z) {
→ cout << "Les trois valeurs sont egales." << endl;
  } else {
    cout << "Seules les deux premieres valeurs sont egales." << endl;
  }
} else {
  if (x == z) {
    cout << "Seules la premiere et la troisieme valeurs sont egales." << endl;
  } else {
    if (y == z) {
      cout << "Seules les deux dernieres valeurs sont egales." << endl;
    } else {
      cout << "Les trois valeurs sont differentes." << endl;
    }
  }
}

```

Les trois valeurs sont egales. est affiche

Supposons:

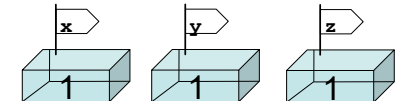


```

if (x == y) {
  if (y == z) {
    cout << "Les trois valeurs sont egales." << endl;
  } else {
    cout << "Seules les deux premieres valeurs sont egales." << endl;
  }
→ } else {
  if (x == z) {
    cout << "Seules la premiere et la troisieme valeurs sont egales." << endl;
  } else {
    if (y == z) {
      cout << "Seules les deux dernieres valeurs sont egales." << endl;
    } else {
      cout << "Les trois valeurs sont differentes." << endl;
    }
  }
}

```

Supposons:

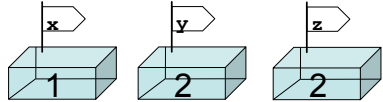


```

if (x == y) {
  if (y == z) {
    cout << "Les trois valeurs sont egales." << endl;
  } else {
    cout << "Seules les deux premieres valeurs sont egales." << endl;
  }
} else {
  if (x == z) {
    cout << "Seules la premiere et la troisieme valeurs sont egales." << endl;
  } else {
    if (y == z) {
      cout << "Seules les deux dernieres valeurs sont egales." << endl;
    } else {
      cout << "Les trois valeurs sont differentes." << endl;
    }
  }
}
→

```

Supposons:

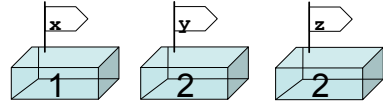


```

→ if (x == y) {
    if (y == z) {
        cout << "Les trois valeurs sont egales." << endl;
    } else {
        cout << "Seules les deux premieres valeurs sont egales." << endl;
    }
} else {
    if (x == z) {
        cout << "Seules la premiere et la troisieme valeurs sont egales." << endl;
    } else {
        if (y == z) {
            cout << "Seules les deux dernieres valeurs sont egales." << endl;
        } else {
            cout << "Les trois valeurs sont differentes." << endl;
        }
    }
}

```

Supposons:

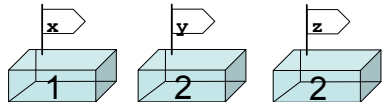


```

if (x == y) {
    if (y == z) {
        cout << "Les trois valeurs sont egales." << endl;
    } else {
        cout << "Seules les deux premieres valeurs sont egales." << endl;
    }
} else {
    if (x == z) {
        cout << "Seules la premiere et la troisieme valeurs sont egales." << endl;
    } else {
        if (y == z) {
            cout << "Seules les deux dernieres valeurs sont egales." << endl;
        } else {
            cout << "Les trois valeurs sont differentes." << endl;
        }
    }
}

```

Supposons:

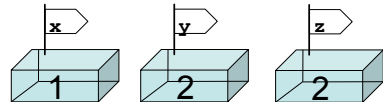


```

if (x == y) {
    if (y == z) {
        cout << "Les trois valeurs sont egales." << endl;
    } else {
        cout << "Seules les deux premieres valeurs sont egales." << endl;
    }
} else {
    if (x == z) {
        cout << "Seules la premiere et la troisieme valeurs sont egales." << endl;
    } else {
        if (y == z) {
            → cout << "Seules les deux dernieres valeurs sont egales." << endl;
        } else {
            cout << "Les trois valeurs sont differentes." << endl;
        }
    }
}

```

Supposons:

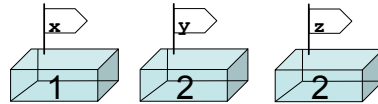


```

if (x == y) {
    if (y == z) {
        cout << "Les trois valeurs sont egales." << endl;
    } else {
        cout << "Seules les deux premieres valeurs sont egales." << endl;
    }
} else {
    if (x == z) {
        cout << "Seules la premiere et la troisieme valeurs sont egales." << endl;
    } else {
        if (y == z) {
            → cout << "Seules les deux dernieres valeurs sont egales." << endl;
        } else {
            cout << "Seules les deux dernieres valeurs sont egales. est affiche"
        }
    }
}

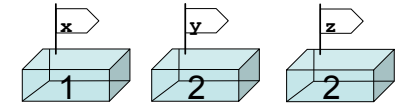
```

Supposons:



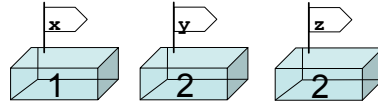
```
if (x == y) {
    if (y == z) {
        cout << "Les trois valeurs sont egales." << endl;
    } else {
        cout << "Seules les deux premieres valeurs sont egales." << endl;
    }
} else {
    if (x == z) {
        cout << "Seules la premiere et la troisieme valeurs sont egales." << endl;
    } else {
        if (y == z) {
            cout << "Seules les deux dernieres valeurs sont egales." << endl;
        } else {
            cout << "Les trois valeurs sont differentes." << endl;
        }
    }
}
```

Supposons:



```
if (x == y) {
    if (y == z) {
        cout << "Les trois valeurs sont egales." << endl;
    } else {
        cout << "Seules les deux premieres valeurs sont egales." << endl;
    }
} else {
    if (x == z) {
        cout << "Seules la premiere et la troisieme valeurs sont egales." << endl;
    } else {
        if (y == z) {
            cout << "Seules les deux dernieres valeurs sont egales." << endl;
        } else {
            cout << "Les trois valeurs sont differentes." << endl;
        }
    }
}
```

Supposons:



```
if (x == y) {
    if (y == z) {
        cout << "Les trois valeurs sont egales." << endl;
    } else {
        cout << "Seules les deux premieres valeurs sont egales." << endl;
    }
} else {
    if (x == z) {
        cout << "Seules la premiere et la troisieme valeurs sont egales." << endl;
    } else {
        if (y == z) {
            cout << "Seules les deux dernieres valeurs sont egales." << endl;
        } else {
            cout << "Les trois valeurs sont differentes." << endl;
        }
    }
}
```

Les conditions

L'instruction `if` fait apparaître une **condition** entre parenthèses

```

Condition
  |
if (n < 5) {
    cout << "Votre nombre est plus petit que 5." << endl;
} else {
    cout << "Votre nombre est plus grand ou egal a 5." << endl;
}

```

Attention, la condition est toujours entourée de parenthèses.

Valeurs des conditions

Une condition est un cas particulier d'expression, **qui ne peut prendre que deux valeurs**.

En C++, ces valeurs se notent **true** et **false**.

- Une condition vaut **true** quand elle est vraie, et
- une condition vaut **false** quand elle est fausse.

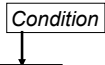
Par exemple, la condition `n < 5` vaut **true** si `n` vaut 0, et **false** si `n` vaut 10.

Pour l'instant, nous n'avons rencontré que des conditions simples, comme `n < 5` ou `x == y`.

Nous allons voir maintenant comment s'écrivent les conditions d'une façon générale.

Les conditions

L'instruction `if` fait apparaître une **condition** entre parenthèses



```
if (n < 5) {  
    cout << "Votre nombre est plus petit que 5." << endl;  
} else {  
    cout << "Votre nombre est plus grand ou egal a 5." << endl;  
}
```

Attention, la condition est toujours entourée de parenthèses.

Pour l'instant, nous n'avons rencontré qu'une condition simple, `n < 5`

Nous allons voir maintenant comment s'écrivent les conditions d'une façon générale.

Les conditions simples Les opérateurs de comparaison

Une **condition simple** compare deux expressions.

Elle utilise un **opérateur de comparaison**, comme `<` ou `>`

Opérateurs de comparaison du langage C++:

Opérateur de comparaison	Signification
<code><</code>	inférieur à
<code>></code>	supérieur à
<code>==</code>	égal à
<code><=</code>	inférieur ou égal à
<code>>=</code>	supérieur ou égal à
<code>!=</code>	différent de

Les conditions Les opérateurs

Une condition
Elle utilise un
Opérateurs de

Attention:

L'opérateur pour tester si deux valeurs sont égales s'écrit avec deux signes égal ==

Un seul signe = représente l'affectation

Par exemple, si on veut tester si la variable n est égale à 5, il faut écrire:

```
if (n == 5)
```

et non pas:

```
if (n = 5)
```

==

égal à

<=

inférieur ou égal à

>=

supérieur ou égal à

!=

différent de

Attention:

Il n'y a pas d'espaces entre les deux caractères

Opérateur de comparaison

Signification

inférieur à

supérieur à

égal à

inférieur ou égal à

supérieur ou égal à

différent de

```
int a(1);
int b(2);

if (a == b) {
    cout << "Cas 1" << endl;
} else {
    cout << "Cas 2" << endl;
}

if (2 * a == b) {
    cout << "b est egal au double de a." << endl;
}
```

affiche

Cas 2

b est egal au double de a.

```
int a(1);
int b(2);

if (a != b) {
    cout << "Cas 2" << endl;
} else {
    cout << "Cas 1" << endl;
}

if (2 * a != b) {
    cout << "b est different du double de a." << endl;
}
```

affiche

Cas 2

b est different du double de a.

```

int a(1);
int b(2);

if (a <= b) {
    cout << "Cas 3" << endl;
} else {
    cout << "Cas 4" << endl;
}

if (2 * a <= b) {
    cout << "b est superieur ou egal au double de a." << endl;
}

affiche

Cas 3
b est superieur ou egal au double de a.

```

Les opérateurs logiques

On peut relier des conditions simples par des opérateurs logiques.

L'opérateur logique and (ET):

par exemple, la condition

$(a < b) \text{ and } (c < d)$

est vraie **uniquement** si les deux conditions $(a < b)$ et $(c < d)$ sont toutes les deux vraies.

L'opérateur and peut aussi s'écrire `&&`: On aurait pu écrire

$(a < b) \text{ \&\& } (c < d)$

Exemple avec l'opérateur logique and

```

cout << "Entrez un nombre entre 1 et 10:" << endl;
cin >> n;

if ((n >= 1) and (n <= 10)) {
    cout << "correct" << endl;
} else {
    cout << "incorrect" << endl;
}

```

Exemple avec l'opérateur logique and

```

cout << "Entrez un nombre entre 1 et 10:" << endl;
cin >> n;

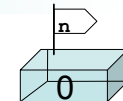
```

faux

```

if ((n >= 1) and (n <= 10)) {
    cout << "correct" << endl;
} else {
    cout << "incorrect" << endl;
}

```



Supposons que la valeur
entrée pour `n` soit 0

incorrect est affiché

Exemple avec l'opérateur logique and

```
cout << "Entrez un nombre entre 1 et 10:" << endl;
cin >> n;
```

faux

vrai faux

```
if ((n >= 1) and (n <= 10)) {
    cout << "correct" << endl;
} else {
    cout << "incorrect" << endl;
}
```

n

12

Supposons que la valeur
entrée pour n soit 12

incorrect est affiché

Exemple avec l'opérateur logique and

```
cout << "Entrez un nombre entre 1 et 10:" << endl;
cin >> n;
```

vrai

vrai vrai

```
if ((n >= 1) and (n <= 10)) {
    cout << "correct" << endl;
} else {
    cout << "incorrect" << endl;
}
```

n

5

Supposons que la valeur
entrée pour n soit 5

correct est affiché

Les opérateurs logiques

L'opérateur logique **or** (OU):

par exemple, la condition

$(a < b) \text{ or } (c < d)$

est vraie **si au moins une** des deux conditions $(a < b)$ ou $(c < d)$ est vraie.

L'opérateur **or** peut aussi s'écrire **||**: On aurait pu écrire

$(a < b) \text{ || } (c < d)$

Exemple avec l'opérateur logique or

```
cout << "Entrez deux valeurs:" << endl;
cin >> m >> n;
```

```
if ((m >= 0) or (n >= 0)) {
    cout << "au moins une valeur est positive" << endl;
} else {
    cout << "les deux valeurs sont negatives" << endl;
}
```

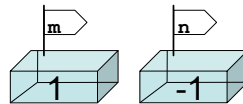
Exemple avec l'opérateur logique `or`

```
cout << "Entrez deux valeurs:" << endl;
cin >> m >> n;
```

```

      vrai
    -----
    vrai
  -----
if ((m >= 0) or (n >= 0)) {
    cout << "au moins une valeur est positive" << endl;
} else {
    cout << "les deux valeurs sont negatives" << endl;
}

```



Supposons que la valeur entrée pour `m` soit +1, et la valeur entrée pour `n` soit -1

au moins une valeur est positive est affiché

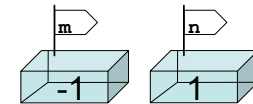
Exemple avec l'opérateur logique `or`

```
cout << "Entrez deux valeurs:" << endl;
cin >> m >> n;
```

```

      faux      vrai
    -----
    faux
  -----
if ((m >= 0) or (n >= 0)) {
    cout << "au moins une valeur est positive" << endl;
} else {
    cout << "les deux valeurs sont negatives" << endl;
}

```



Supposons que la valeur entrée pour `m` soit -1, et la valeur entrée pour `n` soit +1

au moins une valeur est positive est affiché

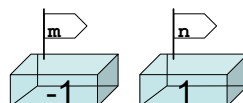
Exemple avec l'opérateur logique `or`

```
cout << "Entrez deux valeurs:" << endl;
cin >> m >> n;
```

```

      faux      faux
    -----
    faux
  -----
if ((m >= 0) or (n >= 0)) {
    cout << "au moins une valeur est positive" << endl;
} else {
    cout << "les deux valeurs sont negatives" << endl;
}

```



Supposons que la valeur entrée pour `m` soit -1, et la valeur entrée pour `n` soit -1

les deux valeurs sont negatives est affiché

Les opérateurs logiques

L'opérateur logique `not` (NON):

par exemple, la condition

`not(a < b)`

est vraie si $(a < b)$ est fausse, et fausse si $(a < b)$ est vraie.

L'opérateur `not` peut aussi s'écrire `!`: On aurait pu écrire

`!(a < b)`

Nous verrons des exemples d'utilisation de cet opérateur plus loin dans la suite du cours.

Erreurs classiques

Le test d'égalité s'écrit ==, et pas =

```
if (a = 1) // !!!
```

est accepté par le compilateur mais

- ne teste pas si a vaut 1, et
- affecte la valeur 1 à a.

Utilisé avec -Wall, g++ affiche le *warning* suivant:

warning: suggest parentheses around assignment used as truth value

ou si vous avez une installation en Français:

attention : parenthèses suggérées autour de l'affectation utilisée comme valeur de vérité

Erreurs classiques

```
if (a == 1); // !!!  
cout << "a vaut 1" << endl;
```

a vaut 1 est toujours affiché quelle que soit la valeur de a!

Le point-virgule est considéré comme une instruction, qui ne fait rien.

Le code précédent est compris par le compilateur comme:

```
if (a == 1)  
;  
cout << "a vaut 1" << endl;
```

le cout est donc situé après le if.

Aucun *warning* n'est affiché.

Si on utilise des accolades même quand il n'y a qu'une instruction dans le bloc, et qu'on écrit le test de la façon suivante:

```
if (a == 1) {  
    cout << "a vaut 1" << endl;  
}
```

l'erreur précédente a beaucoup moins de chance d'arriver.

Erreurs classiques

Ne pas oublier les accolades, l'indentation ne suffit pas:

```
if (n < p)  
    cout << "n est plus petit que p" << endl;  
    max = p;  
else  
    cout << "n est plus grand ou egal a p" << endl;
```

génère à la compilation l'erreur:

syntax error before "else"

Voici une meilleure présentation du code précédent:

```
if (n < p)  
    cout << "n est plus petit que p" << endl;  
max = p;  
else  
    cout << "n est plus grand ou egal a p" << endl;
```

L'instruction max = p; est déjà en dehors du if!

```
cout << "Entrez le premier nombre:" << endl;
cin >> n;
cout << "Entrez le deuxieme nombre:" << endl;
cin >> p;
```

```
if ((n < p) and (2 * n >= p)) {
    cout << "1";
}
```

```
if ((n < p) or (2 * n >= p)) {
    cout << "2";
}
```

```
if (n < p) {
    if (2 * n >= p) {
        cout << "3";
    } else {
        cout << "4";
    }
}
```

```
cout << endl;
```

Qu'affiche ce programme quand l'utilisateur entre 1 et 2 ?

Rappel:

- Pour le ET (and):
les deux conditions doivent être vraies;
- Pour le OU (or):
au moins l'une des conditions doit être vraie.

A: 2

B: 24

C: 123

D: 1234

?

```
cout << "Entrez le premier nombre:" << endl;
cin >> n;
cout << "Entrez le deuxieme nombre:" << endl;
cin >> p;
```

```
if ((n < p) and (2 * n >= p)) {
    cout << "1";
}
```

```
if ((n < p) or (2 * n >= p)) {
    cout << "2";
}
```

```
if (n < p) {
    if (2 * n >= p) {
        cout << "3";
    } else {
        cout << "4";
    }
}
```

```
cout << endl;
```

Qu'affiche ce programme quand l'utilisateur entre 1 et 3 ?

Rappel:

- Pour le ET (and):
les deux conditions doivent être vraies;
- Pour le OU (or):
au moins l'une des conditions doit être vraie.

A: 2

B: 24

C: 123

D: 1234

?

```
cout << "Entrez le premier nombre:" << endl;
cin >> n;
cout << "Entrez le deuxieme nombre:" << endl;
cin >> p;
```

```
if ((n < p) and (2 * n >= p)) {
    cout << "1";
}
```

```
if ((n < p) or (2 * n >= p)) {
    cout << "2";
}
```

```
if (n < p) {
    if (2 * n >= p) {
        cout << "3";
    } else {
        cout << "4";
    }
}
```

```
cout << endl;
```

Qu'affiche ce programme quand l'utilisateur entre 2 et 1 ?

Rappel:

- Pour le ET (and):
les deux conditions doivent être vraies;
- Pour le OU (or):
au moins l'une des conditions doit être vraie.

A: 2

B: 24

C: 123

D: 1234

?

Le type booléen (bool)

Le type bool

Le type `bool` (pour *boolean*, ou booléen) est le type des **conditions**.

Il permet de déclarer des **variables contenant la valeur d'une condition**.

Une variable de type booléen est souvent appelée simplement un **booléen**.

Comme les conditions, un booléen ne peut prendre que **deux valeurs possibles**: `true` ou `false`

Exemple:

```
int a(1);  
int b(2);  
bool test1(a == b);  
bool test2(a < b);
```

Le type bool

Le type `bool` (pour *boolean*, ou booléen) est le type des **conditions**.

Il permet de déclarer des **variables contenant la valeur d'une condition**.

Une variable de type booléen est souvent appelée simplement un **booléen**.

Comme les conditions, un booléen ne peut prendre que **deux valeurs possibles**: `true` ou `false`

Exemple:

```
int a(1);  
int b(2);  
→ bool test1(a == b);  
bool test2(a < b);
```

Le type bool

Le type `bool` (pour *boolean*, ou booléen) est le type des **conditions**.

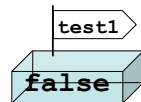
Il permet de déclarer des **variables contenant la valeur d'une condition**.

Une variable de type booléen est souvent appelée simplement un **booléen**.

Comme les conditions, un booléen ne peut prendre que **deux valeurs possibles**: `true` ou `false`

Exemple:

```
int a(1);  
int b(2);  
→ bool test1(a == b);  
bool test2(a < b);
```



Le type bool

Le type `bool` (pour *boolean*, ou booléen) est le type des **conditions**.

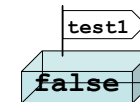
Il permet de déclarer des **variables contenant la valeur d'une condition**.

Une variable de type booléen est souvent appelée simplement un **booléen**.

Comme les conditions, un booléen ne peut prendre que **deux valeurs possibles**: `true` ou `false`

Exemple:

```
int a(1);  
int b(2);  
bool test1(a == b);  
→ bool test2(a < b);
```



Le type bool

Le type `bool` (pour *boolean*, ou booléen) est le type des **conditions**.

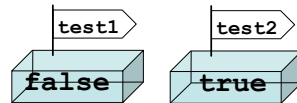
Il permet de déclarer des **variables contenant la valeur d'une condition**.

Une variable de type booléen est souvent appelée simplement un **booléen**.

Comme les conditions, un booléen ne peut prendre que **deux valeurs possibles**: `true` ou `false`

Exemple:

```
int a(1);
int b(2);
bool test1(a == b);
→ bool test2(a < b);
```



On peut initialiser des booléens à l'aide des constantes `false` et `true`.

On peut utiliser des booléens comme des conditions. Par exemple:

- on peut utiliser des opérateurs logiques (`and`, `or` et `not`) entre booléens;
- on peut utiliser un booléen comme condition dans un `if`.

```
int a(1);
int b(2);

bool c(true);
bool d(a == b);
bool e(d or (a < b));

if (e) {
    cout << "e vaut true" << endl;
}
```

On peut initialiser des booléens à l'aide des constantes `false` et `true`.

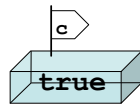
On peut utiliser des booléens comme des conditions. Par exemple:

- on peut utiliser des opérateurs logiques (`and`, `or` et `not`) entre booléens;
- on peut utiliser un booléen comme condition dans un `if`.

```
int a(1);
int b(2);

→ bool c(true);
bool d(a == b);
bool e(d or (a < b));

if (e) {
    cout << "e vaut true" << endl;
}
```



On peut initialiser des booléens à l'aide des constantes `false` et `true`.

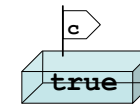
On peut utiliser des booléens comme des conditions. Par exemple:

- on peut utiliser des opérateurs logiques (`and`, `or` et `not`) entre booléens;
- on peut utiliser un booléen comme condition dans un `if`.

```
int a(1);
int b(2);

bool c(true);
→ bool d(a == b);
bool e(d or (a < b));

if (e) {
    cout << "e vaut true" << endl;
}
```



On peut initialiser des booléens à l'aide des constantes `false` et `true`.

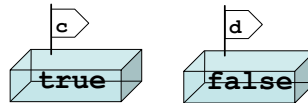
On peut utiliser des booléens comme des conditions. Par exemple:

- on peut utiliser des opérateurs logiques (`and`, `or` et `not`) entre booléens;
- on peut utiliser un booléen comme condition dans un `if`.

```
int a(1);
int b(2);

bool c(true);
→ bool d(a == b);
bool e(d or (a < b));

if (e) {
    cout << "e vaut true" << endl;
}
```



On peut initialiser des booléens à l'aide des constantes `false` et `true`.

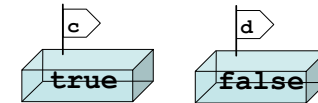
On peut utiliser des booléens comme des conditions. Par exemple:

- on peut utiliser des opérateurs logiques (`and`, `or` et `not`) entre booléens;
- on peut utiliser un booléen comme condition dans un `if`.

```
int a(1);
int b(2);

bool c(true);
bool d(a == b);
→ bool e(d or (a < b));

if (e) {
    cout << "e vaut true" << endl;
}
```



On peut initialiser des booléens à l'aide des constantes `false` et `true`.

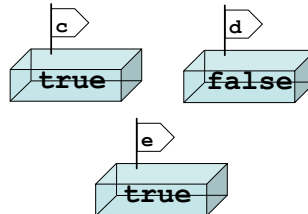
On peut utiliser des booléens comme des conditions. Par exemple:

- on peut utiliser des opérateurs logiques (`and`, `or` et `not`) entre booléens;
- on peut utiliser un booléen comme condition dans un `if`.

```
int a(1);
int b(2);

bool c(true);
bool d(a == b);
→ bool e(d or (a < b));

if (e) {
    cout << "e vaut true" << endl;
}
```



On peut initialiser des booléens à l'aide des constantes `false` et `true`.

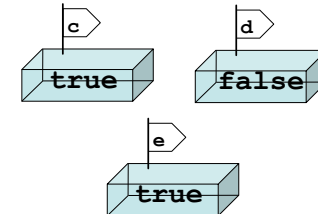
On peut utiliser des booléens comme des conditions. Par exemple:

- on peut utiliser des opérateurs logiques (`and`, `or` et `not`) entre booléens;
- on peut utiliser un booléen comme condition dans un `if`.

```
int a(1);
int b(2);

bool c(true);
bool d(a == b);
bool e(d or (a < b));

→ if (e) {
    cout << "e vaut true" << endl;
}
```



On peut initialiser des booléens à l'aide des constantes `false` et `true`.

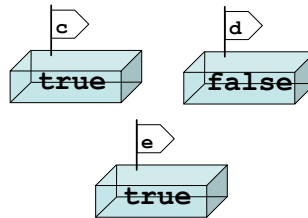
On peut utiliser des booléens comme des conditions. Par exemple:

- on peut utiliser des opérateurs logiques (`and`, `or` et `not`) entre booléens;
- on peut utiliser un booléen comme condition dans un `if`.

```
int a(1);
int b(2);

bool c(true);
bool d(a == b);
bool e(d or (a < b));

if (e) {
→ cout << "e vaut true" << endl;
}
```



Les booléens sont utiles pour de nombreux problèmes, nous rencontrerons des exemples concrets dans la suite du cours.