

Branchements conditionnels

Jusqu'ici, tous les programmes que nous avons vus, contenaient des instructions qui s'exécutent les unes après les autres. Dans cette section, on va voir comment changer cette manière d'exécution. En effet, on va voir comment un ordinateur sera capable de choisir, i.e., exécuter des instructions ou les ignorer si certaines conditions sont remplies.

Pour mieux éclaircir les choses, écrivons un programme qui permet de déterminer si un nombre entier entré au clavier est positif ou négatif.

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

int main(){
    int n;
    cout << "Entrer un nombre entier: ";
    cin >> n;

    if (n < 0){
        cout << "Nombre negatif." << endl;
    }
    else {
        cout << "Nombre positif." << endl;
    }

    cout << "Fin du programme!";
}
```

Dans le programme ci-dessus, on a introduit une nouvelle instruction :

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

L'instruction 'if' fait apparaître une condition entre parenthèses. Si la condition est vérifiée (vraie) alors les instructions qui se trouvent dans le premier bloc (le bloc if) seront exécutées. Sinon, c'est à dire la condition n'est pas vérifiée (fausse) alors on exécute les instructions qui se trouvent dans le deuxième bloc (bloc else).

```
if (condition) {
    Condition vraie: on exécute les instructions de ce bloc.
}
else {
    Condition fausse: on exécute les instructions de ce bloc.
}
```

C'est-à-dire, soit le bloc 'if' soit le bloc 'else' qui sera exécuté, et non pas les deux ensemble. Exécutons le programme précédent pas à pas. Supposons que l'utilisateur a choisi la valeur 3 pour la variable n.

```
if (n < 0){
    cout << "Nombre negatif." << endl;
}
else {
    cout << "Nombre positif." << endl;
}
```

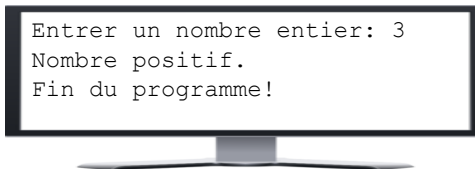
Si $n = 3$ alors la condition $(n < 0)$ n'est pas vérifiée (fausse) parce que 3 n'est pas inférieur à 0. Ce qui veut dire qu'on va sauter (ignorer) le premier bloc (le bloc if) et exécuter seulement les instructions du deuxième bloc (else), à savoir :

```
cout << "Nombre positif." << endl;
```

Ensuite, l'ordinateur va continuer l'exécution du reste du code, dans notre exemple, l'ordinateur exécute l'instruction :

```
cout << "Fin du programme!";
```

	Mémoire
	...
n	3
	...



Supposons maintenant que l'utilisateur a choisi la valeur -5 pour la variable n.

```
if (n < 0){
    cout << "Nombre negatif." << endl;
}
else {
    cout << "Nombre positif." << endl;
}
```

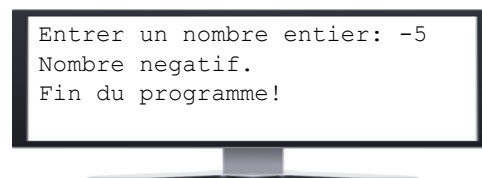
Si $n = -5$ alors la condition $(n < 0)$ est vérifiée (vraie) parce que -5 est inférieur à 0. Cela veut dire qu'on va exécuter les instructions du premier bloc (le bloc if) et sauter (ignorer) le deuxième bloc (else) :

```
cout << "Nombre Negatif." << endl;
```

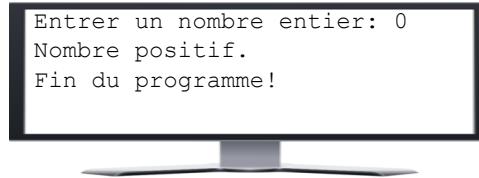
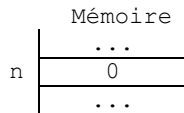
Après l'exécution de l'instruction ci-dessus, l'ordinateur va continuer l'exécution du reste du code, i.e., il exécutera l'instruction :

```
cout << "Fin du programme!";
```

	Mémoire
	...
n	-5
	...



Un dernier exemple, supposons que l'utilisateur a choisi la valeur 0 pour la variable n.



Si $n = 0$ alors la condition $(n < 0)$ n'est pas vérifiée (fausse) parce que 0 n'est pas inférieur à 0. Ce qui veut dire qu'on va sauter (ignorer) le premier bloc (le bloc `if`) et exécuter seulement les instructions du deuxième bloc (`else`), à savoir :

```
cout << "Nombre positif." << endl;
```

Ensuite, l'ordinateur exécute l'instruction :

```
cout << "Fin du programme!";
```

Remarques

- L'écriture suivante est erronée:

```
if n < 0 {  
    ...  
}  
else {  
    ...  
}
```

La condition doit être toujours entourée de parenthèses.

```
if (n < 0) {  
    ...  
}  
else {  
    ...  
}
```

- On peut mettre autant d'instructions qu'on veut dans un bloc.

Exemple

```
if (n < 0) {  
    cout << "Votre nombre est " << n << endl;  
    cout << "Bloc if: nombre inferieur a zero" << endl;  
    cout << "Votre nombre est negatif." << endl;  
}  
else {  
    cout << "Votre nombre est " << n << endl;  
    cout << "Bloc else: nombre superieur ou egal a zero" << endl;  
    cout << "Votre nombre est positif." << endl;  
}
```

- Les accolades '{' et '}' délimitent les blocs d'instructions. Mais, quand un bloc contient une seule instruction, il n'est pas obligatoire d'utiliser des accolades.

Exemple

```
if (n < 0)
    cout << "Nombre negatif." << endl;
else
    cout << "Nombre positif." << endl;
```

Néanmoins, une bonne pratique est de toujours utiliser les accolades, même s'il n'y a qu'une seule instruction. Donc, il vaut mieux écrire les instructions précédentes de cette manière:

```
if (n < 0){
    cout << "Nombre negatif." << endl;
}
else {
    cout << "Nombre positif." << endl;
}
```

- L'instruction 'if' peut ne pas avoir de deuxième partie.

Exemple

Supposons qu'on veut juste vérifier si un nombre est négatif. En d'autres termes, rien à faire si le nombre est positif.

```
#include<iostream>
using namespace std;
int main(){
    int n;
    cout << "Entrer un nombre entier: ";
    cin >> n;

    if (n < 0){
        cout << "Nombre negatif.";
    }
}
```

Exercice : Ecrire un programme qui détermine le maximum entre deux nombres entiers choisis par l'utilisateur et affiche ce maximum à l'écran.

```
#include<iostream>
using namespace std;
int main(){
    int a, b;

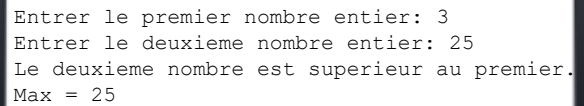
    cout << "Entrer le premier nombre entier: ";
    cin >> a;
    cout << "Entrer le deuxieme nombre entier: ";
    cin >> b;

    if (a > b){
        cout << "Le premier nombre est superieur au deuxieme." << endl;
        cout << "Max = " << a << endl;
    }
    else {
        cout << "Le deuxieme nombre est superieur au premier." << endl;
        cout << "Max = " << b << endl;
    }
}
```

Supposons que l'utilisateur a choisi :

a = 3
b = 25

Mémoire	
	...
a	3
b	25
	...



```
Entrer le premier nombre entier: 3
Entrer le deuxieme nombre entier: 25
Le deuxieme nombre est superieur au premier.
Max = 25
```

Si a = 3 et b = 25 alors la condition (3 > 25) est fausse. Ce qui veut dire que le deuxième bloc (else) sera exécuté:

```
else {
    cout << "Le deuxieme nombre est superieur au premier." << endl;
    cout << "Max = " << b << endl;
}
```

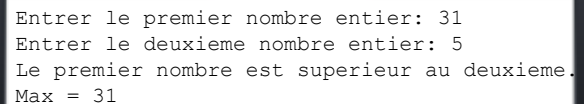
La partie else représente la négation de la condition (a > b).

```
if (a > b) {
    ici: a > b
} else {
    ici: a ≤ b c'est-à-dire b ≥ a
}
```

Supposons maintenant que l'utilisateur a choisi :

a = 31
b = 5

Mémoire	
	...
a	31
b	5
	...



```
Entrer le premier nombre entier: 31
Entrer le deuxieme nombre entier: 5
Le premier nombre est superieur au deuxieme.
Max = 31
```

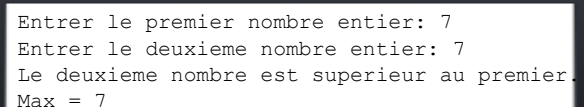
a = 31 et b = 5 alors la condition (31 > 5) est vraie. Cela veut dire que le premier bloc (if) sera exécuté :

```
if (a > b) {
    cout << "Le premier nombre est superieur au deuxieme." << endl;
    cout << "Max = " << a << endl;
}
```

Un dernier exemple, supposons que l'utilisateur a choisi:

a = 7
b = 7

Mémoire	
	...
a	7
b	7
	...



```
Entrer le premier nombre entier: 7
Entrer le deuxieme nombre entier: 7
Le deuxieme nombre est superieur au premier.
Max = 7
```

Si $a = 7$ et $b = 7$ alors la condition $(7 > 7)$ est fausse. Ce qui veut dire que le deuxième bloc (`else`) sera exécuté.

Remarque 1

```
if (a > b){
    cout << "Le premier nombre est superieur au deuxieme." << endl;
    cout << "Max = " << a << endl;
}
else {
    cout << "Le deuxieme nombre est superieur au premier." << endl;
    cout << "Max = " << b << endl;
}
```

Quelle est la différence entre la solution précédente et cette deuxième solution :

```
if (a > b){
    cout << "Le premier nombre est superieur au deuxieme." << endl;
    cout << "Max = " << a << endl;
}

if (b > a){
    cout << "Le deuxieme nombre est superieur au premier." << endl;
    cout << "Max = " << b << endl;
}
```

Premièrement, cette deuxième solution ne traite pas le cas où les deux nombres sont égaux. Mais, on peut la corriger en utilisant l'opérateur supérieur ou égal (\geq) comme suit :

```
if (a >= b){
    cout << "Le premier nombre est superieur au deuxieme." << endl;
    cout << "Max = " << a << endl;
}

if (b > a){
    cout << "Le deuxieme nombre est superieur au premier." << endl;
    cout << "Max = " << b << endl;
}
```

En fait, pour comparer deux programmes (algorithmes), on utilise comme critères *l'espace mémoire* utilisé par chaque programme et le *temps d'exécution*. Bien évidemment, un meilleur programme utilise un petit espace mémoire et s'exécute rapidement (un petit temps d'exécution). Revenons à nos deux programmes.

- Concernant l'espace mémoire, les deux programmes utilisent deux variables de type entier (a et b). C'est-à-dire, ils utilisent le même espace mémoire.
- Passons au temps d'exécution, dans le premier programme, on a une seule instruction de comparaison (`if(a>b)`) alors que dans le deuxième, on a deux instructions de comparaison (`if(a>=b)` et `if(b>a)`). Alors, on en conclut que le premier programme est meilleur.

Un deuxième point à prendre en considération. Dans le premier programme, on a utilisé les instructions `if-else`:

```

if (a > b){
    cout << "Le premier nombre est superieur au deuxieme." << endl;
    cout << "Max = " << a << endl;
}
else {
    cout << "Le deuxieme nombre est superieur au premier." << endl;
    cout << "Max = " << b << endl;
}

```

Donc, l'un des deux blocs sera exécuté, soit le bloc `if` soit le bloc `else`. Si `a` n'est pas supérieur à `b` alors `a` est inférieur ou égal `b`. On n'a pas besoin de refaire le test. La partie `else` s'en charge automatiquement. Mais, dans le deuxième programme :

```

if (a >= b){
    cout << "Le premier nombre est superieur au deuxieme." << endl;
    cout << "Max = " << a << endl;
}

if (b > a){
    cout << "Le deuxieme nombre est superieur au premier." << endl;
    cout << "Max = " << b << endl;
}

```

même si la première condition (`a >= b`) est vérifiée, par exemple, `a = 3` et `b = 1`

```

if (a >= b){
    cout << "Le premier nombre est superieur au deuxieme." << endl;
    cout << "Max = " << a << endl;
}

```

on doit quand même passer au deuxième test (`b > a`) et faire la comparaison pour rien.

```

if (b > a){
    cout << "Le deuxieme nombre est superieur au premier." << endl;
    cout << "Max = " << b << endl;
}

```

Remarque 2 : On peut utiliser plusieurs instructions `'if'`.

```

if (condition_1){
    ...
}
else if (condition_2){
    ...
}
else {
    ...
}

```

Par exemple, modifions le programme précédent (maximum entre deux nombres) pour déterminer aussi si les deux nombres entiers entrés au clavier sont égaux. Voici le programme à modifier:

```

if (a > b){
    cout << "Le premier nombre est superieur au deuxieme." << endl;
    cout << "Max = " << a << endl;
}
else {
    cout << "Le deuxieme nombre est superieur au premier." << endl;
    cout << "Max = " << b << endl;
}

```

Comme on a déjà vu, la partie `else` veut dire que $a \leq b$ (ou $b \geq a$). Donc, on a deux autres cas à traiter : $b > a$ et $b == a$. Voici la solution :

```
#include<iostream>
using namespace std;
int main(){
    int a, b;

    cout << "Entrer le premier nombre entier: ";
    cin >> a;
    cout << "Entrer le deuxieme nombre entier: ";
    cin >> b;

    if (a > b){
        cout << "Le premier nombre est superieur." << endl;
        cout << "Max = " << a << endl;
    }
    else if (b > a){
        cout << "Le deuxieme nombre est superieur." << endl;
        cout << "Max = " << b << endl;
    }
    else {
        cout << "Les deux nombres sont egaux." << endl;
    }
}
```

Choix imbriqués

L'instruction `'if'` suit le schéma suivant:

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

Les instructions qui figurent dans les blocs `'if'` ou `'else'` sont absolument quelconques. Donc, il peut s'agir d'autres instructions `'if-else'`.

```
if (condition_1) {
    if(condition_2){
        ...
    }
    else {
        ...
    }
}
else {
    ...
}

if (condition_1) {
    ...
}
else {
    if(condition_2){
        ...
    }
    else {
        ...
    }
}

if (condition_1) {
    if(condition_2){
        ...
    }
    else {
        ...
    }
}
else {
    if(condition_3){
        ...
    }
    else {
        ...
    }
}
```


Exercice : Ecrire un programme C++ qui détermine le maximum entre *trois nombres réels* x , y et z .

On a trois variables x , y et z . Commençons par comparer x avec y .
Si ($x > y$) alors
 ici on est sûr que $x > y$, alors il nous reste à comparer x avec z
 Si ($x > z$) alors : x est le maximum
 Sinon : z est le maximum
Sinon
 Cette partie veut dire que $x \leq y$, ce qui veut dire que x n'est pas max, alors il nous reste à comparer y avec z
 Si ($y > z$) alors : y est le maximum
 Sinon : z est le maximum

```
#include<iostream>
using namespace std;
int main(){
    double x, y, z;
    cout << "Entrer le premier nombre: "; cin >> x;
    cout << "Entrer le deuxieme nombre: "; cin >> y;
    cout << "Entrer le troisieme nombre: "; cin >> z;

    if (x > y){
        if (x > z){
            cout << "Max = " << x << endl;
        }
        else {
            cout << "Max = " << z << endl;
        }
    }
    else {
        if (y > z){
            cout << "Max = " << y << endl;
        }
        else {
            cout << "Max = " << z << endl;
        }
    }
}
```

Deuxième solution

```
#include<iostream>
using namespace std;
int main(){
    double x, y, z;
    cout << "Entrer le premier nombre: "; cin >> x;
    cout << "Entrer le deuxieme nombre: "; cin >> y;
    cout << "Entrer le troisieme nombre: "; cin >> z;

    double max;

    if (x > y){ max = x; }
    else { max = y; }

    if (max < z){ max = z; }

    cout << "Max = " << max << endl;
}
```

Troisième solution

```
#include<iostream>
using namespace std;
int main(){
    double x, y, z;
    cout << "Entrer le premier nombre: ";    cin >> x;
    cout << "Entrer le deuxieme nombre: ";  cin >> y;
    cout << "Entrer le troisieme nombre: "; cin >> z;

    double max = x;
    if (max < y){ max = y; }
    if (max < z){ max = z; }

    cout << "Max = " << max << endl;
}
```