Le C —
Conditions et boucles

https://tinyurl.com/yjexspdu

Nathanael Demacon

Opérateurs
Relationnels

Expressions classiques

- `a < b` (inférieur à)
- `a > b` (supérieur à)
- `a <= b` (inférieur ou égal à)
- `a >= b` (supérieur ou égal à)
- `a == b` (égal à)
- `a != b` (n'est pas égal à)

Attention : == et = sont deux opérateurs différents, == permet de comparer deux valeurs alors que = permet d'assigner une valeur à une variable.

Une condition "vraie" donnera l'entier `1` et une condition "fausse" donnera l'entier `0`. Par défaut il n'y a pas `true` et `false` en C.

```c
int max(int a, int b)
{
    if (a > b)
    {
        return a;
    }

    return b;
}
```

Opérateurs
Relationnels

a < b

Inférieur à

a	b	a < b
1	2	1
2	1	0
3	3	0

a > b

Supérieur à

a	b	a > b
1	2	0
2	1	1
3	3	0

a == b

Égal à

a	b	a == b
1	2	0
2	1	0
3	3	1

a <= b

Inférieur ou égal à

a	b	a <= b
1	2	1
2	1	0
3	3	1

a >= b

Supérieur ou égal à

a	b	a >= b
1	2	0
2	1	1
3	3	1

a != b

N’est pas égal à

a	b	a != b
1	2	1
2	1	1
3	3	0

Opérateurs
Logiques

Expressions classiques

- `a && b (ET logique)`
- `a || b (OU logique)`
- `!a (négation logique)`

Tout comme == et =, il ne faut pas confondre les opérateurs && avec & et || avec |. Les opérateurs logiques renverront toujours 1 ou 0 contrairement aux opérateurs arithmétiques.

Opérateurs
Logiques

a && b
ET logique

a	b	a && b
1	2	1
2	1	1
0	3	0
0	0	0
-1	5	1

a || b
OU logique

a	b	a || b
1	2	1
2	1	1
0	3	1
0	0	0
-1	5	1

!a
Négation logique

a	!a
1	0
2	0
0	1
-1	0

Opérateurs
Logiques

```c
#include <stdio.h>

int a() {
   printf("a\n");
   return 0;
}

int b() {
   printf("b\n");
   return 1;
}

int main() {
   if (a() && b()) {
       printf("true\n");
   } else {
       printf("false\n");
   }

   return 0;
}
```

Avec l'opérateur &&, les expressions sont exécutées dans l'ordre. Si une de ces expressions est "fausse", les expressions suivantes ne seront pas exécutées. On appelle ça un court-circuit.

Même chose avec l'opérateur || si une des expressions est "vraie".

Sortie

```
a
false
```

Conditions
If, else et else if

```
int main()
{
    if (condition1)
    {
        // if la condition 1 est vraie
    }
    else if (condition2)
    {
        // si la condition 2 est vraie
    }
    else
    {
        // si toutes les conditions sont fausses
    }

    return 0;
}
```

Les opérations `if` permettent d'exécuter des instructions si une condition particulière est “vraie”

Les opérations `else` et `else if` ne sont pas obligatoires.

Il est possible de mettre une infinité de `else if` à la chaîne.

Conditions

If, else et else if

```c
int lesser_or_greater(int a, int b)
{
    if (a > b)
    {
        return 1;
    }
    else if (a < b)
    {
        return -1;
    }
    else
    {
        return 0;
    }
}
```

Conditions
Switch

```c
int main()
{
    switch (expression)
    {
    case value1:
        // si expression == value1
        break;
    case value2:
        // si expression == value2
        break;
    case value3:
    case value4:
        // si expression == value3
        // ou expression == value4
        break;
    default:
        // si aucune des conditions n'est vraie
        break;
    }

    return 0;
}
```

Les opérations switch permettent de simplifier plusieurs tests d'égalité successifs.

Il est important d'utiliser `break` pour que le programme ne passe pas au bloc de code suivant.

`default` est optionnel.

Conditions
Ternaire

```c
int main()
{
    int result;

    if (condition)
    {
        result = true_value;
    }
    else
    {
        result = false_value;
    }

    return 0;
}
```

```c
int main()
{
    int result = condition ? true_value : false_value;

    return 0;
}
```

Bien qu'ils soient pratiques et compacts, les ternaires réduisent la lisibilité du code par rapport aux conditions classiques.

Boucles

Une boucle permet d’exécuter plusieurs fois un même bloc d’instructions.

Boucles
While

```c
int main()
{
    while (condition)
    {
        // tant que la condition est vraie
    }

    return 0;
}
```

Les boucles `while` exécutent des instructions tant que la condition est “vraie”. Si la condition n’est jamais vraie, la boucle ne s'exécutera jamais.

Boucles
While

```c
#include <stdio.h>

int main()
{
    int i = 0;

    while (i < 10)
    {
        printf("%d\n", i);
        i++;
    }

    return 0;
}
```

Sortie

```
0
1
2
3
4
5
6
7
8
9
```

Boucles
While

```c
#include <stdio.h>

int main()
{
    int i = 0;

    while (i >= 0)
    {
        printf("%d\n", i);
        i++;
    }

    return 0;
}
```

Attention aux boucles infinies ! Il est possible de rester bloqué dans une boucle si on ne fait pas attention. Le code dans la boucle s'exécutera donc sans interruption.

Sortie

```
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
```

Boucles

For

```c
int main()
{
    for (initialisation; condition; incrémentation)
    {
        // tant que la condition est vraie
    }

    return 0;
}
```

Les boucles `for` fonctionnent exactement comme des boucles `while` mais permettent d'initialiser une ou plusieurs variables ainsi que de les modifier à chaque tour de boucle.

Boucles

For

```c
#include <stdio.h>

int main()
{
    for (int i = 0; i < 10; i++)
    {
        printf("%d\n", i);
    }

    return 0;
}
```


```
0
1
2
3
4
5
6
7
8
9
```

Boucles
Do...While

```c
#include <stdio.h>

int main()
{
    int i = 0;

    do
    {
        printf("%d\n", i);
        i++;
    } while (i < 10);

    return 0;
}
```

Fonctionne comme une boucle `while` normale mais est exécutée au moins une fois. La condition est vérifiée après la première exécution.

Sortie

```
0
1
2
3
4
5
6
7
8
9
```

Boucles
Break

```c
#include <stdio.h>

int main()
{
    int i = 0;

    while (i < 10)
    {
        if (i == 5)
        {
            break;
        }

        printf("%d\n", i);
    }

    return 0;
}
```

Le mot clé “break” permet de sortir de sa boucle parente.

Sortie

```
0
1
2
3
4
```

Boucles
Continue

```c
#include <stdio.h>

int main()
{
    for (int i = 0; i < 10; i++)
    {
        if (i % 2 == 0)
        {
            continue;
        }

        printf("%d\n", i);
    }

    return 0;
}
```

Termine l'itération actuelle et passe à la suivante.

Sortie

```
1
3
5
7
9
```