OPÉRATEURS ET ENTREES SORTIES EN C

Les opérateurs sont la base de tout langage de programmation. Ainsi, la fonctionnalité du langage C est incomplète sans l'utilisation d'opérateurs. Les opérateurs nous permettent d'effectuer différents types d'opérations sur des opérandes. En C, les opérateurs peuvent être classés dans les catégories suivantes:

- **Opérateurs arithmétiques** (+, -, *, /,%, post-incrémentation, pré-incrémentation, post-décrémentation, pré-décrémentation)
- **Opérateurs relationnels** (==,! =,>, < ,> = & <=) Opérateurs logiques (&&, ll et !)
- **Opérateurs binaires** (&, l, ^, ~, >> et <<)
- **Opérateurs d'affectation** (=, +=, -=, *=, etc)
- **Autres opérateurs** (conditionnel, virgule, sizeof, address, redirection)

Opérateurs arithmétiques

Ils sont utilisés pour effectuer des opérations arithmétiques/mathématiques sur les opérandes. Les opérateurs binaires appartenant à cette catégorie sont:

- **Addition** : L'opérateur '**+**' ajoute deux opérandes. Par exemple, **x+y**.
- **Soustraction**: l'opérateur ' **-** ' soustrait deux opérandes. Par exemple, **x-y.**
- **Multiplication**: l'opérateur ' ***** ' multiplie deux opérandes. Par exemple, **x*y**.
- **Division**: l'opérateur '/' divise le premier opérande par le second. Par exemple, **x/y**.
- **Module**: l'opérateur ' **%** ' renvoie le reste lorsque le premier opérande est divisé par le second. Par exemple, **x%y**.

Exemple 1 :

```
1   #include < stdio.h>
2   int main()
3   {
4       int a = 10, b = 4, res;
5
6       res = a + b; // addition
7       printf("a+b : %d\n", res);
8
9       res = a - b; // soustraction
10      printf("a-b : %d\n", res);
11
12      res = a * b; // multiplication
13      printf("a*b : %d\n", res);
14
15      res = a / b; // division
16      printf("a/b : %d\n", res);
17
18      res = a % b; // Module
19      printf("a mod b : %d\n", res);
20
21      return 0;
22  }
```

```
a+b : 14
a-b : 6
a*b : 40
```

```
a/b : 2
a mod b : 2
```

Ceux qui entrent dans la catégorie des opérateurs arithmétiques unaires sont:

Incrémentation : L'opérateur «**++**» permet d'incrémenter la valeur d'un entier. Quand il est est placé devant le nom de la variable (également appelé opérateur de pré-incrémentation), sa valeur est incrémentée instantanément. Par exemple, **++x**. Et quand il est placé après le nom de la variable (également appelé opérateur post-incrémentation), sa valeur est conservée temporairement jusqu'à l'exécution de cette instruction et est mise à jour avant l'exécution de l'instruction suivante. Par exemple, **x++.**

Décrémentation : L'opérateur "**--**" sert à décrémenter la valeur d'un entier. Quand il est est placé avant le nom de la variable (également appelé opérateur de pré-décrémentation), sa valeur est décrémentée instantanément. Par exemple, **--x**. Et quand il est placé après le nom de la variable (également appelé opérateur post-décrémentation), sa valeur est conservée temporairement jusqu'à l'exécution de cette instruction et elle est mise à jour avant l'exécution de l'instruction suivante. Par exemple, **x--**

Exemple 2 :

```
1
2     #include < stdio.h>
3     int main()
4     {
5         int a = 10, b = 4, res;
6
7         // post-incrémentation:
8         res = a++;
9         printf("a : %d et res : %d\n", a, res);
10
11        // post-décrémentation
12        res = a--;
13        printf("a is %d et res : %d\n", a, res);
14
15        // pré-incrémentation
16        res = ++a;
17        printf("a : %d et res : %d\n", a, res);
18
19        // pré-décrémentation
20        res = --a;
21        printf("a : %d et res : %d\n", a, res);
22
23        return 0;
24    }
```

```
a : 11 et res : 10
a is 10 et res : 11
a : 11 et res : 11
a : 10 et res : 10
```

Opérateurs relationnels et logiques

Opérateurs relationnels

Les opérateurs relationnels sont utilisés pour comparer deux valeurs.

L'opérateur '==' vérifie si les deux opérandes sont égales ou non. Si oui, il renvoie la valeur **true**. Sinon, il retourne **false**. Par exemple, **5==5** retournera **true**.

L'opérateur '!=' vérifie si les deux opérandes sont égaux ou non. elle renvoie la valeur **true** si les deux opérandes ne sont pas égaux, Sinon, il retourne **false**. C'est le complément booléen exact de l'opérateur'==’. Par exemple, **5!=5** retournera **false**.

L'opérateur '>' vérifie si le premier opérande est supérieur au deuxième opérande. Si c'est le cas, cela retourne **true**. Sinon, il retourne **false**. Par exemple, **6 > 5** retournera **true**.

L'opérateur '<' vérifie si le premier opérande est inférieur au deuxième opérande. Si c'est le cas, cela retourne **true**. Sinon, il retourne **false**. Par exemple, **6 < 5** renverra **false**.

L'opérateur '> =' vérifie si le premier opérande est supérieur ou égal au deuxième opérande. Si c'est le cas, cela retourne **true**. Sinon, il retourne **false**. Par exemple, **5 > = 5** retournera **true**.

L'opérateur '<=' vérifie si le premier opérande est inférieur ou égal au deuxième opérande. Si c'est le cas, cela retourne **true**. Sinon, il retourne **false**. Par exemple, **5 <= 5** retournera également true.

Exemple 3 :

```
1   #include
2
3   int main()
4   {
5       int a = 10, b = 4;
6
7       // supérieure à
8       if (a > b)
9           printf("a est supérieure à b\n");
10      else
11          printf("a est inférieure ou égale à b\n");
12
13      // supérieure ou égale
14      if (a >= b)
15          printf("a est supérieure ou égale à b\n");
16      else
17          printf("a est inférieure à b\n");
18
19      // inférieure à
20      if (a < b)
21          printf("a est inférieure à b\n");
22      else
```

```
            printf("a est supérieure ou égal b\n");

        // inférieure ou égale
        if (a <= b)
            printf("a est inférieure ou égale à b\n");
        else
            printf("a est supérieure à b\n");

        // égales
        if (a == b)
            printf("les valeurs a et b sont égales \n");
        else
            printf("a et b ne sont pas égales\n");

        // ne sont pas égales
        if (a != b)
            printf("les valeurs a et b ne sont pas égales \n");
        else
            printf("les valeurs a et b sont égales b\n");

        return 0;
    }
```

```
a est supérieure à b
a est supérieure ou égale à b
a est supérieure ou égal b
a est supérieure à b
a et b ne sont pas égales
les valeurs a et b ne sont pas égales
```

Opérateurs logiques

Ils sont utilisés pour combiner deux ou plusieurs conditions / contraintes ou pour compléter l'évaluation de la condition d'origine considérée.
Ils sont décrits ci-dessous:

ET logique: l'opérateur '**&&**' retourne **true** lorsque les deux conditions considérées sont remplies. Sinon, il retourne **false**.
Par exemple, **a && b** renvoie **true** lorsque a et b sont vraies (c'est-à-dire différentes de zéro).

OU logique: l'opérateur «**||**» renvoie la valeur **true** lorsque l'une (ou les deux) des conditions considérées est remplie. Sinon, il retourne **false**.
Par exemple, **a || b** renvoie true si l'un des a ou b est **true** (c'est-à-dire différente de zéro). Bien sûr, il retourne **true** lorsque a et b sont toutes les deux vraies.

NON logique: L'opérateur '**!**' Renvoie **true** si la condition considérée n'est pas remplie. Sinon, il retourne **false**. Par exemple,**!A** retourne **true** si a est false, c'est-à-dire quand **a = 0.**

Exemple 4 :

```
#include

int main()
{
    int a = 10, b = 4, c = 10, d = 20;

```

```
6           // ET logique
7           if (a > b && c == d)
8               printf("a est supérieure à b ET c est égale à d\n");
9           else
10              printf("condition ET non satisfaite\n");
11
12          // OU logique
13          if (a > b || c == d)
14              printf("a est supérieure à b OU c est égale à d\n");
15          else
16              printf("Ni a n'est supérieure à b ni c n'est égale à d ");
17
18          // NON logique
19          if (!a)
20              printf("a est zéro\n");
21          else
22              printf("a est différente de zero");
23
24          return 0;
    }
```

```
condition ET non satisfaite
a est supérieure à b OU c est égale à d
a est différente de zero
```

ENTRÉE ET SORTIE EN LANGAGE C

Entrée signifie fournir au programme certaines données à utiliser dans le programme et **Sortie** signifie afficher des données à l'écran ou les écrire sur une imprimante ou dans un fichier.

Le langage de programmation C offre de nombreuses fonctions intégrées permettant de lire une entrée donnée et d'afficher des données à l'écran lorsqu'il est nécessaire de générer le résultat.

Toutes ces fonctions intégrées sont présentes dans les fichiers d'en-tête C, nous spécifierons également le nom des fichiers d'en-tête dans lesquels une fonction particulière est définie lors de la discussion.

Les fichiers standard

La programmation en C traite tous les périphériques en tant que fichiers. Ainsi, les périphériques tels que l'affichage sont traités de la même manière que les fichiers et les trois fichiers suivants s'ouvrent automatiquement lorsqu'un programme s'exécute pour permettre l'accès au clavier et à l'écran.

Fichier standard	Pointeur de fichier	périphérique
Entrée standard	stdin	Clavier
Sortie standard	stdout	Ecran
Erreur standard	stderr	Ecran

Spécificateurs de format

Type de données	Spécificateur de Format
int	%d
char	%c
float	%f
double	%lf
long double	%Lf

Fonctions scanf() et printf()

Le fichier d'en-tête d'entrée-sortie standard, nommé **stdio.h** contient la définition des fonctions **printf()** et **scanf()**, qui sont utilisées pour afficher la sortie à l'écran et pour lire l'entrée de l'utilisateur respectivement.

printf()

La fonction printf() affiche la valeur transmise en tant que paramètre sur l'écran de la console.

Syntaxe

printf(“%X”, variableDeTypeX);

- où **%X** est le spécificateur de format en C. C'est un moyen d'indiquer au compilateur quel type de données se trouve dans une variable
- **variableDeTypeX** : nom de variable de type X

scanf()

La méthode **scanf()** lit la valeur de la console selon le type spécifié.

Syntaxe

scanf(“%X”, &variableDeTypeX);

- où **%X** est le spécificateur de format en C. C'est un moyen d'indiquer au compilateur quel type de données se trouve dans une variable
- **&** est l'opérateur d'adresse en C, qui indique au compilateur de changer la valeur réelle de cette variable(**variableDeTypeX**), stockée à cette adresse dans la mémoire.

Exemple 1 :

```
1
2       #include < stdio.h>
3
4       int main(void)
5       {
6           int a;
7           float b;
8           char d;
9
10          // lire et ecrire un type entier
11          scanf("%d%",&a);
12          printf("a=%d",a);
13
14          // lire et ecrire un type réel
15          scanf("%f",&b);
16          printf("b=%f",b);
17
18          // lire et ecrire un caractère
19          scanf("%c",&d);
20          printf("d=%c",d);
21
            return 0;
        }
```

```
2
a=2

3.5
b=3.5

f
d=f
```

Remarque : Nous pouvons également limiter le nombre de chiffres ou de caractères pouvant être entrés ou sortis en ajoutant un nombre avec le spécificateur de chaîne de formatage, tel que "**%1d**" ou "**%3s**", le premier correspondant à un seul chiffre et le second. signifie 3 caractères, donc si vous essayez d'entrer 42, alors que scanf() a "**%1d**", il ne prendra que 4 comme entrée. de même pour la sortie.

Valeurs multiples

Voici comment vous pouvez utiliser plusieurs entrées de l'utilisateur et les afficher.

Exemple 2 :

```
1
2       #include < stdio.h>
3
4       int main(void)
5       {
6           int a,b;
7           float c;
8
9           scanf("%d%d%f",&a,&b,&c);
10          printf("a=%d - b=%d - c=%f",a,b,c);
11
12          return 0;
13      }
```

```
2
5
3.5
a=2 - b=5 - c=3.500000
```

Fonctions getchar() et putchar()

La fonction **getchar()** lit un caractère du terminal et le renvoie sous forme d'entier. Cette fonction ne lit qu'un seul caractère à la fois. Vous pouvez utiliser cette méthode dans une boucle si vous souhaitez lire plusieurs caractères.

La fonction **putchar()** affiche le caractère qui lui est passé à l'écran et renvoie le même caractère. Cette fonction affiche également un seul caractère à la fois. Si vous souhaitez afficher plusieurs caractères, utilisez la méthode **putchar()** dans une boucle.

Exemple 3 :

```
1       #include < stdio.h>
2
3       int main(void)
4       {
5           int c;
6           printf("Saisir un caractère : ");
7           c=getchar();
8
9           printf("voici votre caractère : ");
10          putchar(c);
11
12          return 0;
```

```
12      }
```

```
Saisir un caractère : A
voici votre caractère : A
```

Fonctions gets() et puts()

La fonction **gets()** lit une ligne de stdin(entrée standard) dans la mémoire tampon pointée par le pointeur str, jusqu'à la terminaison de la nouvelle ligne ou EOF (fin du fichier) se produise.

La fonction **puts()** écrit la chaine str à la fin de la nouvelle ligne sur stdout.

str : c'est le pointeur sur un tableau de caractères où la chaîne est stockée

Exemple 4 :

```
1
2       #include < stdio.h>
3
4       int main(void)
5       {
6           char nom[20];
7
8           printf("Saisir votre nom : ");
9           gets(nom);
10
11          printf("votre nom est  : ");
12          puts(nom);
13
14          return 0;
15      }
```

```
Saisir votre nom : JEBLI
votre nom est : JEBLI
```

Différence entre scanf() et gets()

La principale différence entre ces deux fonctions est que **scanf()** arrête de lire les caractères lorsqu'il rencontre un espace, mais **gets()** lit également l'espace en tant que caractère.