

ABC

Types et opérateurs en C



Variables

Déclaration :

```
type identificateur;  
ex: int chiffre;
```

Initialisation :

```
type identificateur = valeur_initiale;  
ex: int chiffre = 1;
```

Affectation :

```
identificateur = valeur;  
ex: chiffre = 2;
```

Utilisation :

```
ex: printf("hello %d\n", chiffre);
```

Constantes

```
const int TAILLE = 8;
```

↳ Allocation mémoire, mais contrôle du type

```
#define TAILLE 8
```

↳ Sans allocation mémoire / Remplacé à la compilation

```
TAILLE = 12; ???
```

↳ Erreur de compilation !

Convention : IDENTIFICATEUR de constante en majuscule

Types : Entiers

Type	Taille (octets/bits)	Limites
char	1/8	-128 à 127
unsigned char	1/8	0 à 255
short	2/16	-32 768 à +32 767
unsigned short	2/16	0 à +65 535
int	4/32	-2 147 483 648 à +2 147 483 647
unsigned int	4/32	0 à +4 294 967 295
long	4/32	-2 147 483 648 à +2 147 483 647
unsigned long	4/32	0 à +4 294 967 295
long long	8/64	-9 223 372 036 854 775 807 à +9 223 372 036 854 775 807
unsigned long long	8/64	0 à +18 446 744 073 709 551 615

Types : Entiers

```
int a; long c; short b; unsigned int x;
```

Affectation de valeur en fonction de la base :

Décimal (10) : 1, 289 , -9999

Octal (8) : **0**1, **0**273, **0**777

Hexadécimal (16) : **0x**7DF, **0x**7DF

Opérateurs arithmétiques

Opérateur unaire	Rôle
$+ a$	Identité
$- a$	Opposé

Opérateur binaire	Rôle
$a + b$	Addition
$a - b$	Soustraction
$a * b$	Multiplication
a / b	Division
$a \% b$	Modulo

Opérateurs relationnels

Opérateur binaire	Rôle
$a > b$	Strictement supérieur à
$a < b$	Strictement inférieur à
$a \geq b$	Supérieur ou égal à
$a \leq b$	Inférieur ou égal à
$a == b$	Egal à
$a != b$	Différent de

```
int a = 5, b = 7;  
// a > b est faux, a <= b est vrai
```

Opérateurs logiques

Opérateur logique	Rôle
<code>a && b</code>	ET logique
<code>a b</code>	OU logique (inclusif)
<code>a ^ b</code>	OU exclusif
<code>!a</code>	NON logique

```
int a = 5, b = 7, c = 4;  
// a > b && b > c      est faux  
// !(a > b) && b > c   est vrai
```


Opérateurs d'affectation composée

Opérateur binaire	Equivalent	Rôle
<code>a += b</code>	<code>a = a + b</code>	Addition
<code>a -= b</code>	<code>a = a - b</code>	Soustraction
<code>a *= b</code>	<code>a = a * b</code>	Multiplication
<code>a /= b</code>	<code>a = a / b</code>	Division
<code>a %= b</code>	<code>a = a % b</code>	Modulo

```
int a = 5, b = 7;  
a += b;  
printf("%d", a); // 12
```

Opérateurs incrémentaux

Opérateur unaire	Rôle
a++	renvoie la valeur puis incrémente de 1
++a	incrémente de 1 puis renvoie la valeur
a--	renvoie la valeur puis décrémente de 1
--a	décrémente de 1 puis renvoie la valeur

```
int a = 0;  
printf("%d", a++); // 0  
printf("%d", a);   // 1  
printf("%d", ++a); // 2
```

Les opérateurs d'adresse

& permet d'obtenir l'adresse mémoire d'une variable

***** permet d'obtenir la valeur contenue à l'adresse

```
int a;  
a = 2;  
printf("%x %d %d", &a, a, *&a);  
// 61fe1c 2 2
```

Opérateur sizeof

`sizeof(variable)` donne la taille en octets de l'espace mémoire occupé par `variable` :

```
int n; long long p;
```

```
sizeof(n); // 4
```

```
sizeof(p); // 8
```

fonctionne aussi avec les types :

```
sizeof(int); // 4
```

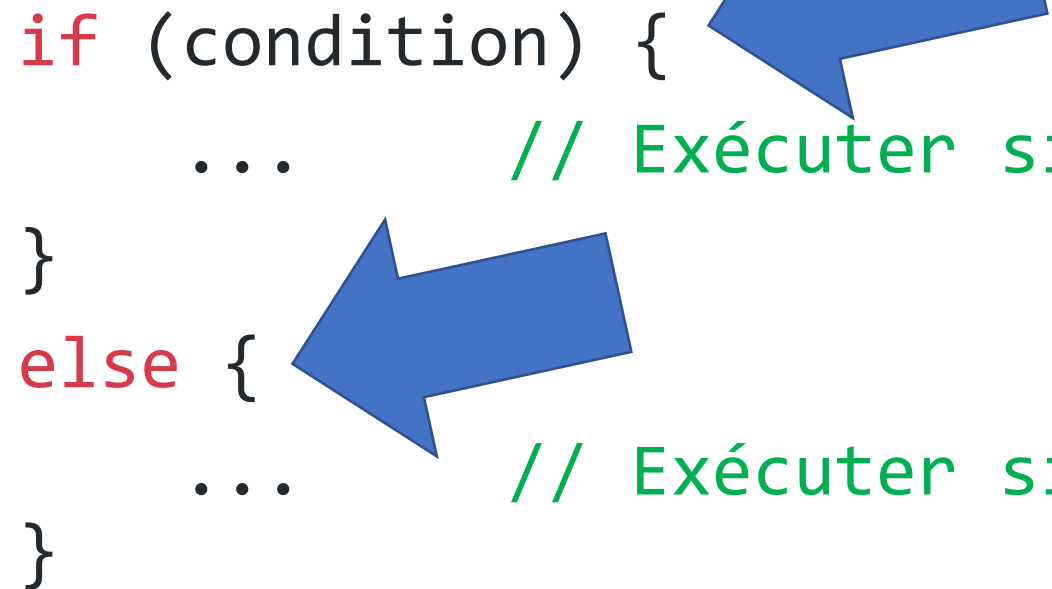
Priorité des opérateurs (vus jusque-là...)

Catégorie	Opérateurs	Associativité
unaire	+ - ++ -- ! & * (cast) sizeof	←
arithmétique	* / %	→
arithmétique	+ -	→
relationnel	< <= > >=	→
relationnel	== !=	→
logique	&&	→
logique		→
conditionnel	? :	→
affectation	= += -= *= /= %=	←
séquentiel	,	→

if ... else ...

Instruction de contrôle

```
if (condition) {  
    ...           // Exécuter si condition vraie  
}  
else {  
    ...           // Exécuter si condition fausse  
}
```



Si plusieurs conditions :

```
if (condition1) {  
    ...  
}  
else if (condition2) {  
    ...  
}  
else {  
    ...  
}
```

Instructions imbriquées :

```
if (condition1) {  
    ...  
    if (condition1.1) {  
        ...  
    }  
    ...  
}  
else {  
    ...  
}
```

I/O - printf

Permet d'afficher une chaîne de caractères dans la sortie standard (la console).

```
printf("chaîne_de_format", arg1, arg2, ..., argn);
```

Les arguments arg1, arg2... sont optionnels.

La chaîne de format est soit :

- une chaîne de caractères utilisée seule
- une chaîne de caractères contenant des formats (%...) qui seront remplacé par les valeurs des arguments arg1, arg2... lors de l'exécution

I/O - printf

Format	Permet d'afficher...
%d ou %i	un entier de 4 octets signé sous forme décimale
%hd	un entier de 2 octets signé sous forme décimale
%hhd	un entier de 1 octets signé sous forme décimale
%x ou %X	un entier en hexadécimal
%o	un entier en octal

I/O - printf

- Entre le % et le caractère de conversion on peut placer :

```
int i = 2;
```

```
printf("XX%dXX", i);      // XX2XX
```

```
printf("XX%3dXX", i);     // XX__2XX
```

```
printf("XX%-3dXX", i);    // XX2__XX
```

```
printf("XX%03dXX", i);    // XX002XX
```

I/O - scanf

```
scanf("chaîne de format", adresse1, adresse2,...);
```

```
#include <stdio.h> // Bibliothèque standard
```

```
int main() {  
    int i;  
    scanf("%d", &i); // 123  
    printf("Entier saisi : %d", i);  
    //Entier saisi : 123  
}
```

Types : Réels

Type	Taille (octets/bits)	Limites
float	4/32	$\pm 10^{-37}$ à $\pm 10^{-38}$
double	8/64	$\pm 10^{-307}$ à $\pm 10^{-308}$
long double	10/80	$\pm 10^{-4932}$ à $\pm 10^{-4932}$

`float a; double c; long double b;`

Décimal : 0.2 ou 927.308

Exposant : 2E-6 ou 0.06E+3

Précision entre 6 et 18 chiffres significatifs

Conversions implicites

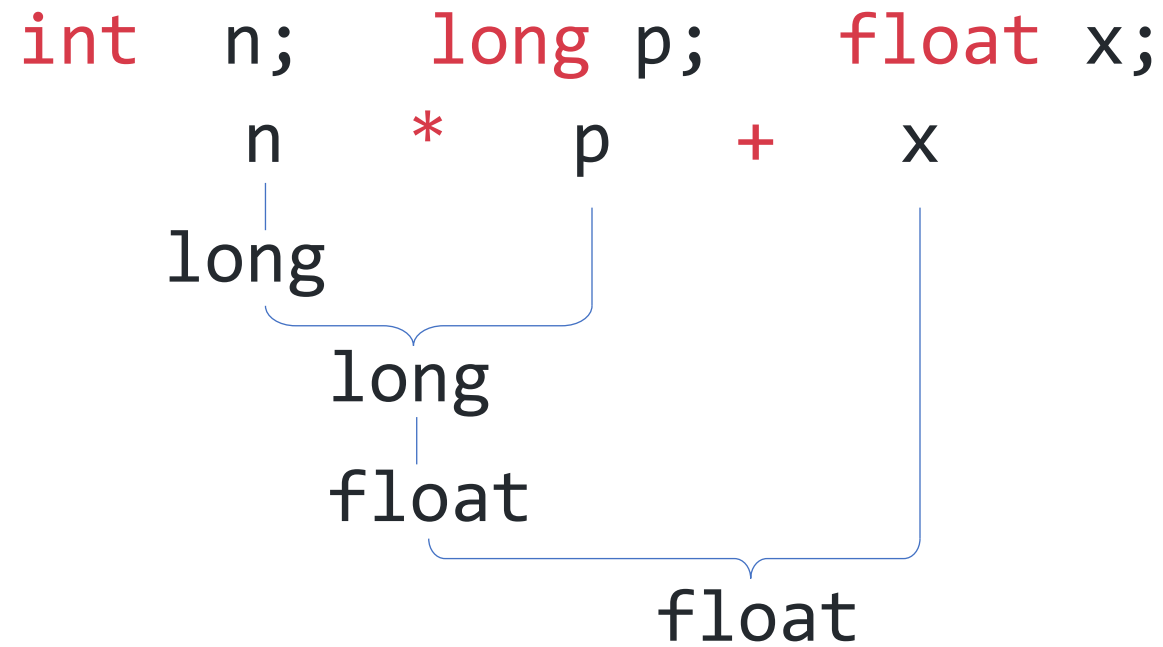
char → short → int → long → float → double

Le compilateur convertit le rang inférieur vers le rang supérieur
(= promotion)

(sens inverse = perte de données)

Conversions implicites

char → short → int → long → float → double



Conversions implicites

Une affectation implique une conversion dans le type de la valeur à gauche (lvalue)

```
int    n;  
float  x = 3.1;  
n = x + 5.8;  
// n = 8 !
```

→ La conversion peut introduire une perte de précision !

Opérateur de conversion

Conversion = « cast »



Forcer le type d'une expression :

```
int n;
```

```
float x;
```

```
n = 12;
```

```
x = n / 5;
```

```
// x = 2.000000
```

```
int n;
```

```
float x;
```

```
n = 12;
```

```
x = (float) (n / 5);
```

```
// x = 2.000000
```

```
int n;
```

```
float x;
```

```
n = 12;
```

```
x = (float) n / 5;
```

```
// x = 2.400000
```


while (tant que)

```
while (condition) {  
    ...  
}
```

```
int a = 1;  
  
while (a <= 10) {  
    printf("%d", a);  
    a = a + 1;  
}
```

```
// 12345678910
```

do ... while (faire ... tant que)

```
do {  
    ...  
}  
while (condition);
```

```
int a = 1;  
  
do {  
    printf("%d", a);  
    a = a + 1;  
}  
while (a <= 10);  
  
// 12345678910
```

for (pour)

```
for (initialisation; condition; incrémentation) {  
    ...  
}
```

Cas simple :

```
for (int i = 1; i <= 10; i++) {  
    printf("%d", i);  
}  
// 12345678910
```

for (pour)

```
int a, b;
```

```
for (                                ) {  
    printf("%2d %2d\n", a, b);  
}
```

« , » est appelé opérateur séquentiel, les expressions sont évaluées de gauche à droite.

1	10
2	9
3	8
4	7
5	6
6	5
7	4
8	3
9	2
10	1

break;

Permet d'interrompre le déroulement de la boucle immédiatement supérieure en passant immédiatement à l'instruction qui suit cette boucle.

```
int saisie;
while (1) {
    scanf("%d", &saisie);
    if (saisie == 100) {
        break;
    }
}
printf("T'es cassé !");
```

continue;

Suspend l'itération en cours (saute les instructions restantes) et reprend l'itération suivante dans la même boucle (break sort de la boucle).

```
int i;
for (i = 1; i <= 100; i++) {
    if (i % 2 == 0) {
        continue;
    }
    printf("Voilà un nombre impair : %d\n", i);
}
```