Le C — Variables et expressions

Nathanael Demacon

Qu’est-ce que le C ?

Le C est un langage de programmation impératif créé en 1972 par Dennis Ritchie, un des créateurs d’UNIX.

```c
#include <stdio.h>

int main()
{
    printf("Hello, World!\n");
    return 0;
}
```

gcc a.c

```
0000 0000 0000 4855 e589 8d48 293d 0000
e800 0004 0000 c031 c35d 25ff 4070 0000
8d4c 711d 0040 4100 ff53 6125 0000 9000
0068 0000 e900 ffe6 ffff 6548 6c6c 2c6f
5720 726f 646c 0021 0001 0000 001c 0000
0000 0000 001c 0000 0000 0000 001c 0000
0002 0000 3f76 0000 0034 0000 0034 0000
3f8b 0000 0000 0000 0034 0000 0003 0000
000c 0001 0010 0001 0000 0000 0000 0100
0000 0000 0000 0000 0000 0000 0000 0000
```

Pour exécuter un programme C, on utilise un compilateur, plus communément `gcc` ou `clang`. Le compilateur transforme le code C en code machine.

Types

Le langage C est un langage de programmation fortement typé. Un type définit la nature des valeurs que peut prendre une donnée, ainsi que les opérateurs qui peuvent lui être appliqués.

```c
int main()
{
   int i = 5;    // a est un entier
   char c = 'a'; // b est un caractère

   return 0;
}
```

Types de base

Voici la liste des types de base en C :

Compilateur 64 bits

- `char` (1 octet)
- `short` (2 octets)
- `int` (4 octets)
- `long` (8 octets)

- `float` (4 octets)
- `double` (8 octets)
- `long double` (16 octets)

Compilateur 32 bits

- `char` (1 octet)
- `short` (2 octets)
- `int` (4 octets)
- `long` (4 octets)

- `float` (4 octets)
- `double` (8 octets)
- `long double` (16 octets)

1 octet = 8 bits

Types de base
sizeof

On peut connaître le nombre d'octets d'un type grâce à `sizeof`. Cet opérateur fonctionne aussi bien avec les types qu'avec les expressions.

```c
int main()
{
    int size_of_int = sizeof(int);  // 4
    int size_of_char = sizeof('a'); // 1

    return 0;
}
```

Types de base
Conversion explicite

Il est possible de convertir explicitement un type en un autre. Il est néanmoins important de faire attention à la compatibilité du type d'origine avec le type cible.

```c
int main()
{
    float a = 3.14;
    int b = (int)a; // 3

    return 0;
}
```

Il n'est cependant pas nécessaire de convertir explicitement tous les types en C. Dans l'exemple ci-dessous, la conversion est faite implicitement.

```c
int main()
{
    float a = 3.14;
    int b = a; // 3

    return 0;
}
```

Variables
Déclaration

Une variable permet de stocker une donnée et de la nommer. En C le type d'une variable doit être spécifié explicitement.

```c
int main()
{
   int a;
   a = 1;
}
```

On peut aussi déclarer plusieurs variables sur la même ligne. Cette méthode de déclaration n'est pas conseillée car elle nuit à la lisibilité du code.

```c
int main()
{
   int a, b, c;
   a = 1;
   b = 2;
   c = 3;
}
```

Variables
Déclaration

Il est plus simple et plus lisible d'affecter directement une valeur à une variable au moment de sa déclaration plutôt qu'après.

```c
int main()
{
    int a = 1;
}
```

Si aucune valeur n'est assignée par défaut, la variable aura une valeur aléatoire. Lors de l'utilisation d'une variable sans valeur explicite, on arrive dans un état de comportement inconnu.

```c
int main()
{
    int a;
    int b = a + 1; // valeur inconnue
}
```

Variables
Scope

En C, les variables ont un principe de scope, ça signifie qu'elles ne peuvent être utilisées que dans un espace limité.

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

    if (1)
    {
        a; // ok
        int b = 2;
    }

    b; // erreur de compilation

    return 0;
}
```

Nombres entiers
Signé VS Non-signé

Ci-dessous une représentation binaire d'entiers de 8 bits (char).

Pour les entiers signés (par défaut), le premier bit sert à définir si l'entier est négatif ou positif. On a donc 1 bit en moins pour stocker notre entier.

Entiers non signés

	128	64	32	16	8	4	2	1
5	0	0	0	0	0	1	0	1
26	0	0	0	1	1	0	1	0
250	1	1	1	1	1	0	1	0

On peut remarquer qu'en non-signé, nous avons accès à une puissance de 2 supérieure comparé aux entiers signés. Cet avantage nous permet d'avoir de plus grands entiers.

Entiers signés

	-	64	32	16	8	4	2	1
5	0	0	0	0	0	1	0	1
26	0	0	0	1	1	0	1	0
-6	1	1	1	1	1	0	1	0

En négatif, la représentation binaire est inversée.
En faisant `0 - 1` on passe donc de `0000 0000` à `1111 1111`.

Nombres entiers
Signé VS Non-signé

Par défaut, tous les types de base sont signés. Pour déclarer un entier non signé, il faut préfixer le mot clé `unsigned` au type.

```
int main()
{
   unsigned int a = 5;

   return 0;
}
```

On peut aussi déclarer un entier littéral non-signé directement grâce au suffix `U`.

```
int main()
{
   1U;  // unsigned int
   2UL; // unsigned long

   return 0;
}
```

Nombres entiers
Limites

Chaque type d'entier étant limité en taille, il y a une limite de stockage pour chacun d'entre eux. La limite étant encore plus importante pour les entiers signés.

	Bits	Minimum signé en 64 bits	Maximum signé en 64 bits	Maximum non-signé en 64 bits
char	8	-128	127	255
short	16	-32,768	32,767	65,535
int	32	-2,147,483,648	2,147,483,647	4,294,967,295
long	64	-9,223,372,036,854,775,808	9,223,372,036,854,775,807	18,446,744,073,709,551,615

Nombres entiers
Limites

Attention : il est possible de dépasser la limite d'espace d'un entier.

Entiers signés

	Décimal	Binaire
a	127	0111 1111
a + 1	-128	1000 0000

Entiers non signés

	Décimal	Binaire
a	255	1111 1111
a + 1	0	0000 0000

Pour palier à ce problème, utiliser le bon type de données en fonction de l'utilisation visée est primordial.

Nombres entiers
Méthodes de déclaration

On peut déclarer un entier de plusieurs manières. Ces diverses méthodes peuvent prouver leur utilité pour améliorer la clarté du code dans certains cas.

```c
int main()
{
    45;         // Décimal
    0b101101;   // Binaire
    0x2D;       // Hexadécimal
    055;        // Octal
}
```

Nombres entiers
Booléens

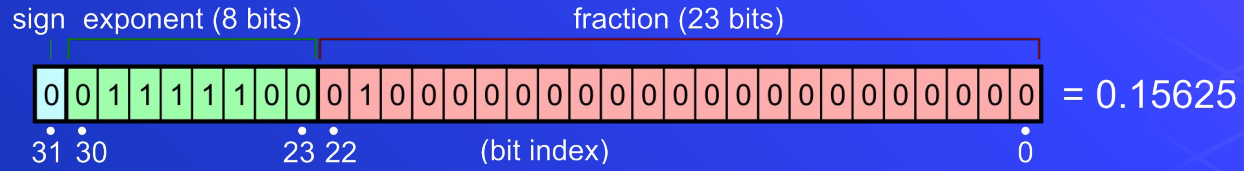
Par défaut, true et false n'existent pas en C. On les remplace respectivement par 1 et 0.

```c
int main()
{
    if (1)
    {
        // executé
    }

    if (0)
    {
        // non-executé
    }

    return 0;
}
```

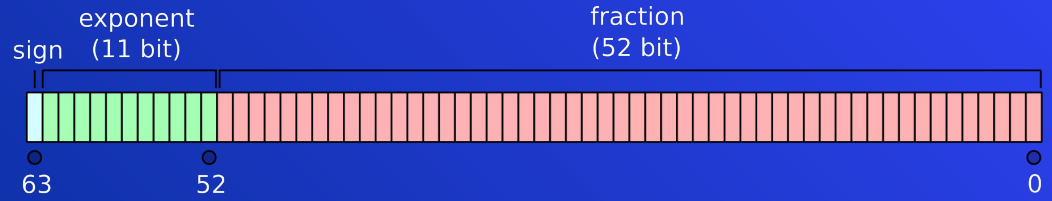
Nombres flottants

Format

Il existe uniquement deux types de flottants en C. Généralement `float` est sur 32 bits et `double` sur 64 bits. La norme qui définit le format des nombres flottant est la IEEE 754.

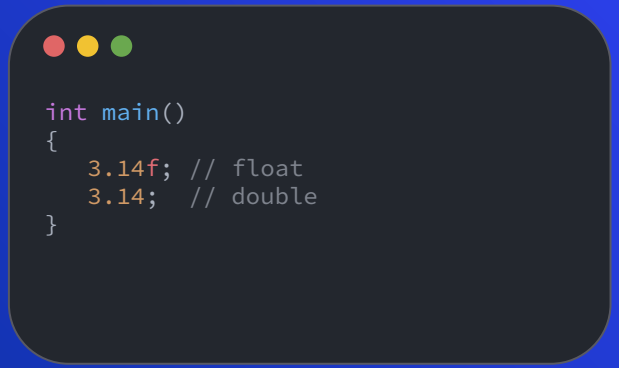


Comparé à un float, un double peut contenir un exposant de 11 bits et une fraction de 52 bits au lieu de respectivement 8 bits et 23 bits pour le float.



Nombres flottants
Méthodes de déclaration

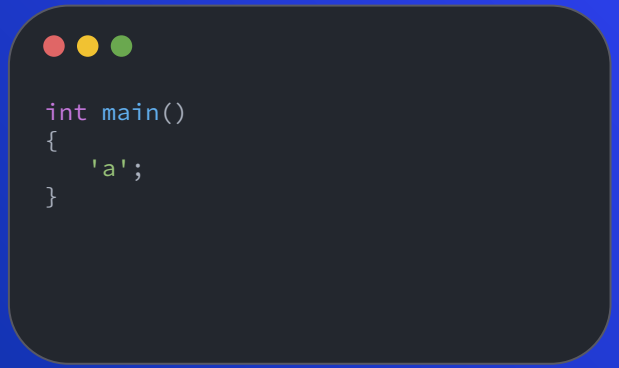
Il existe uniquement deux types de flottants en C. Généralement `float` est représenté en 32 bits et `double` en 64 bits.



Les doubles permettent une précision accrue mais les opérations effectuées dessus seront plus lourdes qu'avec des floats.

Caractères

Un caractère représente une valeur ASCII (comprise entre 0 et 255). Il se déclare entre deux apostrophes.



```c
int main()
{
    'a';
}
```

Attention : ne pas confondre l'entier 0 et le caractère '0'. Le caractère 'a' par exemple correspond à l'entier 97 dans la table ASCII.

Certains caractères peuvent être échappés via \. Voici quelques caractères spéciaux utiles.

	Caractère
Saut de ligne	\n
Tabulation	\t
Null	\0

Opérateurs
Arithmétique

Expressions classiques

- `a + b` (addition)
- `a - b` (soustraction)
- `a * b` (multiplication)
- `a / b` (division)
- `a % b` (modulo)

Le modulo permet de récupérer le reste d'une division entre deux entiers. Par exemple `5 % 2` renverra 1.

Manipulation de bits

- `~a` (complément à un)
- `a & b` (ET)
- `a | b` (OU inclusif)
- `a ^ b` (XOR / OU exclusif)
- `a << n` (décalage de bits vers la gauche)
- `a >> n` (décalage de bits vers la droite)

Opérateurs
Arithmétique

~a
Complément à un

	Décimal	Binaire
a	113	0111 0001
~a	-114	1000 1110

a & b
ET

	Décimal	Binaire
a	113	0111 0001
b	108	0110 1100
a & b	96	0110 0000

a | b
OU inclusif

	Décimal	Binaire
a	113	0111 0001
b	108	0110 1100
a | b	125	0111 1101

a ^ b
XOR / OU exclusif

	Décimal	Binaire
a	113	0111 0001
b	108	0110 1100
a ^ b	29	0001 1101

a << n
Décalage de bits vers la gauche

	Décimal	Binaire
a	113	0111 0001
a << 2	-68	1100 0100
a << 8	0	0000 0000

a >> n
Décalage de bits vers la droite

	Décimal	Binaire
a	113	0111 0001
a >> 2	28	0001 1100
a >> 8	0	0000 0000

Opérateurs
Affectation

Les opérateurs d'affectation permettent de simplifier le code en évitant les répétitions.

- `a = b`
- `a += b`
- `a -= b`
- `a *= b`
- `a /= b`
- `a %= b`
- `a &= b`
- `a |= b`
- `a ^= b`
- `a <<= b`
- `a >>= b`

```c
int main()
{
    // ❌ Sans opérateur d'affectation
    int b = 1;
    b = b + 2;

    // ✅ Avec opérateur d'affectation
    int a = 1;
    a += 2;

    return 0;
}
```

Fonctions
printf

La fonction `printf` permet d'afficher du texte dans la console. Pour utiliser cette fonction, nous devons inclure la bibliothèque standard `stdio.h`.

```c
#include <stdio.h>

int main()
{
   printf("Hello, World!\n");
   return 0;
}
```

On remarque le caractère spécial “`\n`” permettant de faire un saut de ligne.
Ce caractère permet aussi de forcer l'affichage du texte dans le terminal sur certains systèmes d'exploitation. À noter qu'il n'est pas obligatoire dans l'utilisation de `printf`.

Fonctions
printf

Il est possible d'afficher des expressions avec printf. Pour ça, il faut utiliser un format particulier pour chaque type d'expression. Par exemple:

```c
#include <stdio.h>

int main()
{
   int i = 5;
   float f = 3.14;

   printf("i = %d\n", i);
   printf("%f / %d = %f\n", f, i, f / i);

   return 0;
}

// Sortie console :
// i = 5
// 3.140000 / 5 = 0.628000
```

La fonction printf peut prendre une infinité d'éléments. Il faut bien faire attention à ce que la quantité de formats utilisés soit égale au nombre d'arguments passés.

Fonctions
printf

On peut aussi customiser le format de sortie des entiers et des flottants.

```c
#include <stdio.h>

int main()
{
   int i = 5;

   printf("i = %2d\n", i);
   printf("i = %02d\n", i);

   return 0;
}


// Sortie console :
// i =  5
// i = 05
```

```c
#include <stdio.h>

int main()
{
   float a = 3.14;
   printf("a = %.2f\n", a);

   float b = 10e-12;
   printf("b = %.0e\n", b);

   return 0;
}

// Sortie console :
// a = 3.14
// b = 1e-11
```

Fonctions

printf

Type	Format
int / short	%d
unsigned int / unsigned short	%u
long	%ld
unsigned long	%lu
float ou double	%f
char	%c
char*	%s
pointeur	%p
caractère %	%%

Sortie	Format
int avec longueur minimale de n	%nd
int avec longueur minimale de n et rempli de zéros	%0nd
float avec n chiffres derrière la virgule (n'arrondit pas la valeur)	%.2f
float en format exposant	%.0e
hexadécimal	%x
hexadécimal avec minimum n caractères	%0nx

Pour une liste plus fournie, vous pouvez vous référer à la documentation suivante :
https://cplusplus.com/reference/cstdio/printf