

Sommaire

1 Types

2 Constantes

3 Opérateurs

4 Opérateurs sizeof et mémoire

5 Conversion de type

6 Tests

7 Boucles

8 Pointeur

9 Tableaux

10 Fonctions

11 Structures

12 Chaîne de caractères (Strings)

13 Fichiers

1 Types

entiers

char	1 octets	-128 à 127
short	2 octets	-32 768 à 32 767
int	4 octets	-2 147 483 648 à 2 147 483 647



By **l cuvillon** (loikun)
cheatography.com/loikun/

Published 11th August, 2026.
Last updated 18th December, 2023.
Page 1 of 22.

Sponsored by **Readable.com**
Measure your website readability!
<https://readable.com>

1 Types (cont)

unsigned type

pour un type entier non signé.

Exemple: `unsigned char` 0 à 255

virgule flottante

float

2 octets

-3.4e38 à 3.4e38

double

4 octets

- 1.7e308 à 1.7e308

Le type n'est utilisé qu'à 2 occasions :

- déclaration d'une variable, `int a = 2; double var2 = 31.4;`

- entête d'une fonction pour préciser les paramètres et le retour,

`int fonction ( double a, char b) {...}`

C

By **l cuvillon** (loikun)

cheatography.com/loikun/

Published 11th August, 2026.

Last updated 18th December, 2023.

Page 2 of 22.

Sponsored by **Readable.com**

Measure your website readability!

<https://readable.com>

2 Constantes

entières

11 en base 10

0xB en base 16

0b1011 en base 2

réelles

31.4

3.14e1 en notation ingénieur ($3.14 \times 10^1 = 31.4$)

caractères

'A' le code ASCII du caractère A : 65.
`char c = 'A';` ↔ `char c = 65;`

chaines de caractères

"Hello world" un tableau de caractères constant (lecture seule)

"A" un tableau de 2 caractères contenant 'A' et '\0' le caractère de fin de chaîne.



By l cuvillon (loikun)
cheatography.com/loikun/

Published 11th August, 2026.
 Last updated 18th December, 2023.
 Page 3 of 22.

Sponsored by [Readable.com](https://readable.com)
 Measure your website readability!
<https://readable.com>

3 Opérateurs

Arithmétiques

(a=1, b=2)

+	addition	<code>a+2 == 3</code>
-	soustraction	<code>b-a ==1</code>
*	multiplication	<code>a*b == 2</code>
/	division réelle	<code>1.0/b == 0.5</code>
/	division entière ou quotient	<code>1/b == 0</code> ! (<code>1/2 == 0</code>)
%	reste division ou modulo	<code>3%b == 1</code>
++	incrémenter	<code>a++ == 2</code>
--	décrémenter	<code>a-- == 0</code>

Relationnel

==	égalité	<code>a == b</code> <code>faux(0)</code>
!=	différent	<code>a != b</code> <code>vrai(1)</code>
>	supérieur	<code>a > b</code> <code>faux(0)</code>
<	inférieur	<code>a < b</code> <code>vrai(1)</code>
>=	supérieur ou égal	<code>a >= b</code> <code>faux(0)</code>
<=	inférieur ou égal	<code>a <= b</code> <code>faux(0)</code>

Logique

&&	ET	<code>(a==1) && (b==3)</code> <code>faux(0)</code>
	OU	<code>(a==1) (b==3)</code> <code>vrai(1)</code>
!	négation	<code>!(a==b)</code> <code>vrai(1)</code>



3 Opérateurs (cont)

Assignment

=	assigne à gauche la valeur de droite	<code>a = 1</code>
<code>op=</code>	mettre à jour une variable avec l'opération arithmétique <i>op</i>	<code>a+=2; ↔ a=a+2;</code> <code>a%=1; ↔ a=a%1;</code>

4 Opérateurs sizeof et mémoire

<code>sizeof()</code>	taille du type (en octets)	<code>sizeof(int)</code> vaut 4.
<code>&</code>	adresse	<code>&a</code> est l'adresse mémoire où <code>a</code> est stocké.
<code>*</code>	variable pointée, indirection	<code>*p</code> est la variable à l'adresse mémoire contenue dans <code>p</code> . Si <code>p=&a</code> , alors <code>*p</code> est un autre nom pour la variable <code>a</code> .



By **l cuvillon** (loikun)
cheatography.com/loikun/

Published 11th August, 2026.
Last updated 18th December, 2023.
Page 5 of 22.

Sponsored by **Readable.com**
Measure your website readability!
<https://readable.com>

5 Conversion de type

Implicite lors d'une affectation

```
int a = 3.9; ▶ a = 3 (troncature)
double f = 2/3; ▶ double f = 0; ▶ f = 0.0
```

Implicite entre 2 opérandes de type différent

```
double f = 2.0/3; ▶ double f = 2.0/3.0; ▶ f = 0.666
```

Règle: opérande de type de plus petite capacité (entier 3) promu vers le type de l'opérande de plus grande capacité (double)

Explicite

```
( type )                operateur de conversion de type, de cast.
(double)1/2 ▶ 1.0/2 ▶ 0.5
printf("%d\n", (int) 3.94); affiche 3
```



By l cuvillon (loikun)
cheatography.com/loikun/

Published 11th August, 2026.
Last updated 18th December, 2023.
Page 6 of 22.

Sponsored by [Readable.com](https://readable.com)
Measure your website readability!
<https://readable.com>

6 Tests (structure de contrôle)

if

```
if( condition ) { //la condition est évaluée.
    //si vrai, ce bloc de 2 instructions est exécuté
    instruction1;
    instruction2;
}
//si faux, le bloc n'est pas exécuté et l'exécution passe ici.
```

if .. else

```
if( condition ) { //la condition est évaluée.
    //si vrai, ce bloc de 2 instructions est exécuté
    instruction1;
    instruction2;
}
else {
    //si faux, ce bloc est exécuté au lieu du précédent.
    instruction3;
}
```

Exemple:

```
if ( (n%2) == 1 ) {
    printf("nombre impair \n");
}
else {
    printf ("nombre pair\n ");
}
```

if .. else if .. else if .. else

```
if( condition1 ) { //la condition 1 est évaluée.
    //si vrai, ce bloc de 2 instructions est exécuté
    instruction1;
    instruction2;
}
else if( condition2 ){
    //si condition 1 faux mais condition 2 vraie, instruction 3 est exécutée.
    instruction3;
}
else {
    //si condition 1 et 2 fausses, instruction 4 est exécutée.
    instruction4;
}
```

(condition) est un test sur une variable (a), une opération (a!=1) ou le résultat d'une fonction (ftest(param)).
condition est FAUX si la valeur est 0, VRAI pour toute valeur différente de 0.

7 Boucles (Structure de contrôle)

while

```
while( condition ) {
    //exécuter instructions 1 et 2 de ce bloc, tant que condition vrai.
    instruction1;
    instruction2;
    //retour au test de condition
}
```

Exemple : Afficher la suite 0 1 ... 8 9

```
i=0;
while(i<10) {
    printf ("%d \n",i);
    i++;
}
```

do { } while;

```
do {
    instruction1;
    instruction2;
    //exécuter instructions 1 et 2 du bloc,
} while( condition ); //tant que condition vrai.
```

Contrairement à `while`, le bloc est toujours exécuté au moins une fois même si `condition` est faux dès le départ.

❗ `do .. while()` ; est la seule boucle avec un `;` final.

for

```
//Executer initial, puis tester la condition
for ( initial; condition ; iteration) {
    //exécuter instructions 1 et 2 de ce bloc, tant que condition vrai.
    instruction1;
    instruction2;
    //exécuter iteration avant retour au test de condition
}
```

Exemple: Afficher la suite 0 1 ... 8 9

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

for ⇔ while équivalence



7 Boucles (Structure de contrôle) (cont)

```
for ( initial; condition ; iteration) {  
    instructions;  
}
```

⇔

```
initial;  
while( condition ) {  
    instructions;  
    iteration;  
}
```

`break;` sortir immédiatement d'une boucle : la boucle la plus proche si dans des boucles imbriquées.

`continue;` ignorer les instructions de la boucle situées après `continue;`, et retourner au test de la condition. Si dans une boucle `for`, exécuter l'instruction d'iteration avant le test.



By **l cuvillon** (loikun)
cheatography.com/loikun/

Published 11th August, 2026.
Last updated 18th December, 2023.
Page 9 of 22.

Sponsored by **Readable.com**
Measure your website readability!
<https://readable.com>

8 Pointeur

Un pointeur est une adresse mémoire (pointeur == adresse). Un type adresse est `type*`.

Ex: `double*` est le type {adresse d'un double}.

Declaration

`type *p;` p est l'adresse d'une variable de type `type`.

`int *p = NULL;` p est l'adresse d'un entier.
Initialisé à l'adresse invalide `NULL` (0, une bonne pratique).

RAM d'un CPU 64bits (adresses codées sur 8 octets) après l'initialisation avec `NULL` :

Variables: p

```

-----
| 0x0000 (8 octs) |
-----
@mem: 0x040          0x048
  
```

Initialisation (2 options)

`int *p = malloc( sizeof (int) );` Initialise p avec l'adresse d'un bloc de 4 octets réservé avec `malloc()`.

↔ `int *p;`

`p=malloc( sizeof (int) );`

`int *p = &a;` Initialise p avec l'adresse de a (avec `int a=11;`)

↔ `int *p; p=&a;`

Variables: p a

```

-----
| 0xC10 | 11 |
-----
@mem: 0x040          0x048 0xC10 0xC14
  
```

Usage

⚠ Un pointeur ne peut pas être utilisé avant son initialisation avec une adresse valide.



8 Pointeur (cont)

`p` une adresse
Ex: `p=&a ↔ p=0xC10`

`*p` la variable pointée : `a`.
Ex: `*p= 12 ↔ a=12;`

Variables: `p` ----- `*` ----- `->` `a`

```

-----
|          0xC10          |          |          12          |
-----
@mem: 0x040              0x048  0xC10              0xC14
  
```

`p+i` (avec `i` entier) adresse du $i^{\text{ème}}$ `int` en mémoire après `p`.
Donc `p+i` vaut `p+i*sizeof(int)`, quand `p` est de type `int*`.

9 Tableaux

Tableau statique

La taille du tableau est déterminée à la compilation et elle est fixe pendant toute l'exécution. La taille est une *constante*.

<code>int t[3];</code>	un tableau de 3 entiers (valeurs aléatoires)
<code>double tab[4] = {-1.2, 3.2};</code>	un tableau de 4 <code>double</code> . Les 2 premiers éléments sont -1.2 et 3.2, les 2 autres nuls.
<code>double tab[] = {-1.2, 3.2};</code>	un tableau de <code>double</code> , taille 2 imposé par l'initialisation

Tableau dynamique

La taille du tableau est déterminée pendant l'exécution et peut être modifiée. La taille est généralement une *variable*.

<code>double *tab;</code>	un tableau de <code>n</code> <code>double</code> .
<code>tab = (double*) malloc (n*sizeof (int));</code>	Valeurs initiales aléatoires.

Equivalence pointeur et nom d'un tableau

<code>tab ↔ &tab[0]</code>	L'identifiant d'un tableau est l'adresse de son premier élément.
<code>tab+i ↔ &tab[i]</code>	Par nature pour un tableau dynamique, où <code>tab</code> est un pointeur. Par définition pour un tableau statique, où l'identifiant est un alias pour l'adresse du 1 ^{er} élément.



9 Tableaux (cont)

accès aux éléments d'un tableau statique ou dynamique

tab[i] i+1^{ième} élément avec i=0,...taille-1
 ou tab[0] est le premier élément.
 *(tab+i) tab+i est l'adresse du i^{ème} élément après le premier.

Copie entre deux tableaux

```
for (i=0; i< taille ; i++) {
    tab1[i] = tab2[i];
}
```

~~tab1 = tab2;~~ ne copie pas les éléments, c'est une copie entre 2 adresses.

L'opération est incorrecte si tab1 est un tableau dynamique (un pointeur). Aucune copie d'éléments n'est effectué : le même tableau tab2 est accessible via tab1 et tab2.

L'opération est illégale si tab1 est un tableau statique : tab1 est une adresse constante non modifiable.

Comparaison du contenu de deux tableaux

```
equal=1;
for (i=0; i< taille ; i++) {
    if( tab1[i] != tab2[i] ) {
        equal=0;
        break;
    }
}
```

~~tab1 == tab2~~ ne compare pas le contenu, mais l'adresse mémoire des premiers éléments de tab1 et tab2 : donc test toujours FAUX.



10 Fonctions

Passage par valeur

Pour passer des valeurs à une fonction: la fonction utilise le type de retour (`return`) pour son résultat.

```
double add5( double value )           //la variable locale value est initialisée avec 3.14, la valeur de n
{
    double res= value+5;
    return res;
}

int main()
{
    double n=3.14;
    n= add5(n);                       //on passe la valeur n.
    ...                               //n vaut maintenant 8.14
}
```

Passage par adresse

Pour passer des variables à une fonction: la fonction retourne ses résultats en modifiant les variables d'origines via leurs adresses.

```
void add5( double *var )              //le pointeur var est initialisé avec adresse de n
{
    *var = *var + 5;                  //opérateur d'indirection pour accéder à la variable pointée n
}

{
    ...
    double n=3.14;
    add5(&n);                         //on passe l'adresse de n.
    ... }                             //n vaut maintenant 8.14.
```

Note: l'adresse mémoire est le moyen pour la fonction `add5()` d'accéder à des variables qui ne lui appartiennent pas, et donc lui sont inconnues (cf portée des variables).

Passage par adresse pour retourner plusieurs résultats



10 Fonctions (cont)

Car `return` ne permet de retourner qu'un résultat

```
void divisionEntiere ( int nombre, int diviseur, int *quotient, int *reste )
{
    *quotient = nombre /diviseur;
    *reste = nombre %diviseur;
}

{ ...
int n=7, d=3;
int quotient, reminder;           //variable pour les résultats
divisionEntiere(n, d, &quotient, &reminder); //passage de leurs adresses
... }
```

Passage d'un tableau

Passer l'adresse du premier élément (↔ le nom du tableau) et le nombre d'éléments.

Passage par adresse, toute modification des éléments a lieu sur les éléments du tableau d'origine.

```
void tab_init( double *tab, int n )
{
    int i=0;
    for (i=0; i<n ; i++) { tab[i]=0;
    }
}

int main()
{
    double u[3];
    tab_init(u,3);
    return(0);
}
```



11 Structures

Définition et déclaration d'une structure

```
struct person { char name[30];
               int age;
               };
```

Définit le type `struct person` : une structure avec 2 membres ou champs.
Attention: la définition finit avec un `;`.

```
struct person joe, diane;
```

Déclare 2 structures de type `struct person`.

Définition et déclaration avec un `typedef`

```
typedef struct { char name[30];
               int age;
               } person_t;
```

Définit le type `person_t`: une structure avec 2 membres.
`typedef` permet de définir un alias pour un type : utilisé ici pour alléger le type en 2 mots `struct person` en un type en un mot `person_t`.

```
person_t joe, diane;
```

Déclare 2 structures de type `person_t`.

Initialisation à la déclaration

```
struct person joe ={"D upo nt", 13};
struct person joe = {.age= 13, .name= " Dup ont
                    "};
```

Valide à la déclaration.
Après la déclaration, il faut accéder chaque membre séparément pour l'initialiser (cf ci-dessous).

Accès aux membres



By l cuvillon (loikun)
cheatography.com/loikun/

Published 11th August, 2026.
Last updated 18th December, 2023.
Page 16 of 22.

Sponsored by [Readable.com](https://readable.com)
Measure your website readability!
<https://readable.com>

11 Structures (cont)

```
person_t *pdiane;
pdiane= malloc ( sizeof ( person_t ) );
```

Crée 1 structure, allouée via un `malloc()`.

```
joe.age= 27;
strcpy(diane.name, "Verovski");
```

opérateur `.`, pour accéder à partir d'une structure précède.

Note: Avec un pointeur `pdiane`, `( *pdiane ).age = 45;` est valide mais écriture déconseillée.

```
pdiane->age = 27;
strcpy(pdiane->name, "Verovski");
```

opérateur `->`, pour accéder à partir d'un pointeur.

Note: `(&joe)->age=27;` est valide mais écriture déconseillée.

copie de structures

```
joe = diane;
```

Les valeurs des membres de `diane` sont copiés dans les membres de `joe`.

Note : valide entre structures de même type, donc de même taille mémoire, mais invalide pour la copie de tableaux (Voir section tableaux).

Fonction et structre

```
void initPerson ( struct person *p )
{ p->age = 0;
  strcpy(p->name, " ");
}
```

```
{...
  struct person john;
  initPerson(&joe);
  ...
```



11 Structures (cont)

ou avec typedef :

```
void initPerson ( person_t  *p )
{
    p->page = 0;
    strcpy(p->name, " ");
}

{...
    person_t john;
    initPerson(&joe);
    ...
}
```

Le passage par adresse est privilégié pour l'efficacité : pas de copie de toutes les valeurs de la structure.



By **l cuvillon** (loikun)
cheatography.com/loikun/

Published 11th August, 2026.
Last updated 18th December, 2023.
Page 18 of 22.

Sponsored by **Readable.com**
Measure your website readability!
<https://readable.com>

12 Chaînes de caractères (Strings)

Déclaration

```
char name[30];
```

un tableau de `char`, pouvant servir à stocker une chaîne.

```
char name[30]= " marion " ;
```

un tableau, initialisé avec la chaîne de caractères "marion".

```
↔ char name[30] = {'m', 'a', 'r', 'i', 'o',  
, 'n', '\0'};
```

```
char name[] = " marion ";
```

un tableau de 7 char (6+caractère de fin \0, initialisé avec la chaîne de caractères "marion".

~~char *name= "marion";~~

une adresse/pointeur `name` initialisé avec l'adresse d'une chaîne *constante* "marion".

A éviter, la chaîne n'est pas modifiable.

Initialisation après déclaration

```
char name[30];
```

(Voir Tableaux pour plus de détail). Copie d'adresses et non de contenu : `name` est l'adresse du premier élément de `name`.

~~name="marion",~~

```
strcpy(name, " marion ");
```

copie la chaîne " marion " dans name.

```
sprintf(name, " %s", " marion ");
```

copie la chaîne "marion" dans `{{lang-c}}name`.`

By **I cuvillon** (loikun)
cheatography.com/loikun/

Published 11th August, 2026.
Last updated 18th December, 2023.
Page 19 of 22.

Sponsored by **Readable.com**
Measure your website readability!
<https://readable.com>

12 Chaînes de caractères (Strings) (cont)

Lire/extraire une chaîne de caractères

<code>scanf("%s", name);</code>	Lire <i>un mot</i> au clavier. Paramètre: adresse du tableau = son identifiant.
<code>fgets(name, SIZE, stdin);</code>	Lire une ligne au clavier. Le ' <code>\n</code> ' est ajouté à la fin de name. SIZE est la taille max du tableau name
<code>sscanf(str, "%s %d", name, &age);</code>	Comme <code>printf()</code> mais la lecture est faite dans la chaîne <code>printf()</code> et non la mémoire du clavier

Ecrire une chaîne de caractères

<code>strcpy(dest, src);</code>	Copie la chaîne <code>src</code> dans <code>dest</code> . Au programmeur de s'assurer que le tableau <code>dest</code> est assez grand
<code>printf("%s", name);</code>	Affiche une chaîne de caractère à partir de l'adresse du premier élément du tableau, d'où l'identifiant du tableau en paramètre. Le ' <code>\0</code> ' indique à <code>printf</code> où arrêter l'affichage des caractères.
<code>sprintf(name, "%s %d", " marion ", age)</code> ;	comme <code>printf()</code> , mais la chaîne est écrite dans <code>name</code> et non dans le terminal.



12 Chaînes de caractères (Strings) (cont)

Autres manipulation

```
char dest[10] = " tic -", src[5] = "
Concaténation : ajoute à la fin de dest, le contenu de la chaîne src
to c";
strcat(dest, src);
```

```
strcmp(dest, src);
Comparaison : retourne 0 si le contenu de dest est identique au contenu de src
```

```
dest == src
(voir Tableaux pour plus de détail) Compare les adresses mémoire des 2 chaînes (toujours
Faux) et non leur contenu.
```

Une chaîne de caractères (ou string) est un tableau contenant une suite de caractères, dont la fin de la suite est marquée par le caractère spécial '\0'.



By **l cuvillon** (loikun)
cheatography.com/loikun/

Published 11th August, 2026.
 Last updated 18th December, 2023.
 Page 21 of 22.

Sponsored by **Readable.com**
 Measure your website readability!
<https://readable.com>

13 Fichier texte

Lecture d'un fichier

Un fichier texte à lire (la première ligne contient le nombre de données):

3

loic 65.3

marc 43.2

gerard 77.1

Code correspondant:

```
File *f=NULL;
double poids=0;
int n=0;
char name[30];

f=fopen("file.txt","r");
if (f==NULL) {
    printf ("error open\n " ) ; return (-1);
}
fscanf ("%d " ,&n); /lecture du nombre de ligne/
while( fscanf(f, "%s %lf", name, &poids ) != EOF) {
    printf ("%s %f", name, poids);
}
```

A chaque lecture ou écriture, la tête de lecture avance dans le fichier. On ne peut lire que sous la tête de lecture.



By **l cuvillon** (loikun)
cheatography.com/loikun/

Published 11th August, 2026.
Last updated 18th December, 2023.
Page 22 of 22.

Sponsored by **Readable.com**
Measure your website readability!
<https://readable.com>