# Types, Variables, Constantes, opérateurs, fonctions printf()/scanf() et les fonctions

## 1. Les types de base

En mathématiques, l'ordre de grandeur des nombres est illimité et les rationnels peuvent être exprimés sans perte de précision. Or un ordinateur ne peut traiter aisément que des nombres entiers de taille limitée. Il utilise le système binaire pour calculer et sauvegarder ces nombres. Ce n'est que par des astuces de calcul et de représentation que l'ordinateur obtient des valeurs correctement approchées des entiers très grands.

Un programmeur en C ne doit pas connaitre les détails des méthodes de codage et de calcul mais il doit :

- Choisir le type le plus approprié à un problème donné.
- Choisir également un type approprié pour la représentation sur l'écran.
- Prévoir le type résultant d'une opération entre différents types.
- Prévoir et optimiser la précision des résultats intermédiaires au cours d'un calcul complexe (avec la possibilité de forcer le changement de types pour des types plus adaptés : c'est ce que l'on appelle le cast).

Les types de base permettent de définir des objets comme entiers, flottant ou caractère. Ces objets peuvent être des variables, des fonctions ou encore des constantes (cf. exemple plus haut la fonction `main()` est de type `int`).

Ci-dessous le tableau des caractéristiques des types en C :

| Type | Description | Taille | Valeur |
|---|---|---|---|
| void | Type générique | | |
| char | caractère | 1 octet | comme signed ou unsigned char |
| unsigned char | caractère non signé | 1 octet | 0 **à** 255 |
| signed char | caractère signé | 1 octet | -128 **à** 127 |
| short | entier court signé | 2 octets | -32 768 **à** 32 767 |
| unsigned short | entier non signé | 2 octets | 0 **à** 65 535 |
| int | entier signé | 2 ou 4 octets | (en fonction du compilateur) |
| unsigned | entier non signé | 2 ou 4 octets | (en fonction du compilateur) |
| long | entier signé long | 4 octets | -2 147 483 648 **à** 2 147 483 647 |

| | | | |
|---|---|---|---|
| unsigned long | entier non signé long | 4 octets | 0 **à** 4 294 967 296 |
| float | flottant | 4 octets | Mantisse : +- 6 chiffres significatifs |
| double | flottant | 8 octets | Mantisse : +- 12 chiffres significatifs |
| long double | flottant | 16 octets | |

- **Les caractères** : le mot clé désignant les caractères est `char`. Si la variable `b` de type `char` a pour valeur `'e'`, `b+1` renverra le caractère suivant dans le code ASCII à savoir `'f'`.

## Table ASCII -I

| Dec | Hex | Char | Dec | Hex | Char | Dec | Hex | Char | Dec | Hex | Char |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 00 | Null | 32 | 20 | Space | 64 | 40 | @ | 96 | 60 | ` |
| 1 | 01 | Start of heading | 33 | 21 | ! | 65 | 41 | A | 97 | 61 | a |
| 2 | 02 | Start of text | 34 | 22 | " | 66 | 42 | B | 98 | 62 | b |
| 3 | 03 | End of text | 35 | 23 | # | 67 | 43 | C | 99 | 63 | c |
| 4 | 04 | End of transmit | 36 | 24 | $ | 68 | 44 | D | 100 | 64 | d |
| 5 | 05 | Enquiry | 37 | 25 | % | 69 | 45 | E | 101 | 65 | e |
| 6 | 06 | Acknowledge | 38 | 26 | & | 70 | 46 | F | 102 | 66 | f |
| 7 | 07 | Audible bell | 39 | 27 | ' | 71 | 47 | G | 103 | 67 | g |
| 8 | 08 | Backspace | 40 | 28 | ( | 72 | 48 | H | 104 | 68 | h |
| 9 | 09 | Horizontal tab | 41 | 29 | ) | 73 | 49 | I | 105 | 69 | i |
| 10 | 0A | Line feed | 42 | 2A | * | 74 | 4A | J | 106 | 6A | j |
| 11 | 0B | Vertical tab | 43 | 2B | + | 75 | 4B | K | 107 | 6B | k |
| 12 | 0C | Form feed | 44 | 2C | , | 76 | 4C | L | 108 | 6C | l |
| 13 | 0D | Carriage return | 45 | 2D | - | 77 | 4D | M | 109 | 6D | m |
| 14 | 0E | Shift out | 46 | 2E | . | 78 | 4E | N | 110 | 6E | n |
| 15 | 0F | Shift in | 47 | 2F | / | 79 | 4F | O | 111 | 6F | o |
| 16 | 10 | Data link escape | 48 | 30 | 0 | 80 | 50 | P | 112 | 70 | p |
| 17 | 11 | Device control 1 | 49 | 31 | 1 | 81 | 51 | Q | 113 | 71 | q |
| 18 | 12 | Device control 2 | 50 | 32 | 2 | 82 | 52 | R | 114 | 72 | r |
| 19 | 13 | Device control 3 | 51 | 33 | 3 | 83 | 53 | S | 115 | 73 | s |
| 20 | 14 | Device control 4 | 52 | 34 | 4 | 84 | 54 | T | 116 | 74 | t |
| 21 | 15 | Neg. acknowledge | 53 | 35 | 5 | 85 | 55 | U | 117 | 75 | u |
| 22 | 16 | Synchronous idle | 54 | 36 | 6 | 86 | 56 | V | 118 | 76 | v |
| 23 | 17 | End trans. block | 55 | 37 | 7 | 87 | 57 | W | 119 | 77 | w |
| 24 | 18 | Cancel | 56 | 38 | 8 | 88 | 58 | X | 120 | 78 | x |
| 25 | 19 | End of medium | 57 | 39 | 9 | 89 | 59 | Y | 121 | 79 | y |
| 26 | 1A | Substitution | 58 | 3A | : | 90 | 5A | Z | 122 | 7A | z |
| 27 | 1B | Escape | 59 | 3B | ; | 91 | 5B | [ | 123 | 7B | { |
| 28 | 1C | File separator | 60 | 3C | < | 92 | 5C | \ | 124 | 7C | \| |
| 29 | 1D | Group separator | 61 | 3D | = | 93 | 5D | ] | 125 | 7D | } |
| 30 | 1E | Record separator | 62 | 3E | > | 94 | 5E | ^ | 126 | 7E | ~ |
| 31 | 1F | Unit separator | 63 | 3F | ? | 95 | 5F | _ | 127 | 7F | □ |

## TABLE ASCII -II

| Dec | Hex | Char | Dec | Hex | Char | Dec | Hex | Char | Dec | Hex | Char |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 128 | 80 | Ç | 160 | A0 | á | 192 | C0 | └ | 224 | E0 | α |
| 129 | 81 | ü | 161 | A1 | í | 193 | C1 | ┴ | 225 | E1 | ß |
| 130 | 82 | é | 162 | A2 | ó | 194 | C2 | ┬ | 226 | E2 | Γ |
| 131 | 83 | â | 163 | A3 | ú | 195 | C3 | ├ | 227 | E3 | π |
| 132 | 84 | ä | 164 | A4 | ñ | 196 | C4 | ─ | 228 | E4 | ∑ |
| 133 | 85 | à | 165 | A5 | Ñ | 197 | C5 | ┼ | 229 | E5 | σ |
| 134 | 86 | å | 166 | A6 | ª | 198 | C6 | ╞ | 230 | E6 | µ |
| 135 | 87 | ç | 167 | A7 | º | 199 | C7 | ╟ | 231 | E7 | τ |
| 136 | 88 | ê | 168 | A8 | ¿ | 200 | C8 | ╚ | 232 | E8 | Φ |
| 137 | 89 | ë | 169 | A9 | ⌐ | 201 | C9 | ╔ | 233 | E9 | Θ |
| 138 | 8A | è | 170 | AA | ¬ | 202 | CA | ╩ | 234 | EA | Ω |
| 139 | 8B | ï | 171 | AB | ½ | 203 | CB | ╦ | 235 | EB | δ |
| 140 | 8C | î | 172 | AC | ¼ | 204 | CC | ╠ | 236 | EC | ∞ |
| 141 | 8D | ì | 173 | AD | ¡ | 205 | CD | ═ | 237 | ED | ø |
| 142 | 8E | Ä | 174 | AE | « | 206 | CE | ╬ | 238 | EE | ε |
| 143 | 8F | Å | 175 | AF | » | 207 | CF | ╧ | 239 | EF | ∩ |
| 144 | 90 | É | 176 | B0 | ░ | 208 | D0 | ╨ | 240 | F0 | ≡ |
| 145 | 91 | æ | 177 | B1 | ▒ | 209 | D1 | ╤ | 241 | F1 | ± |
| 146 | 92 | Æ | 178 | B2 | ▓ | 210 | D2 | ╥ | 242 | F2 | ≥ |
| 147 | 93 | ô | 179 | B3 | │ | 211 | D3 | ╙ | 243 | F3 | ≤ |
| 148 | 94 | ö | 180 | B4 | ┤ | 212 | D4 | ╘ | 244 | F4 | ⌠ |
| 149 | 95 | ò | 181 | B5 | ╡ | 213 | D5 | ╒ | 245 | F5 | ⌡ |
| 150 | 96 | û | 182 | B6 | ╢ | 214 | D6 | ╓ | 246 | F6 | ÷ |
| 151 | 97 | ù | 183 | B7 | ╖ | 215 | D7 | ╫ | 247 | F7 | ≈ |
| 152 | 98 | ÿ | 184 | B8 | ╕ | 216 | D8 | ╪ | 248 | F8 | ° |
| 153 | 99 | Ö | 185 | B9 | ╣ | 217 | D9 | ┘ | 249 | F9 | ∙ |
| 154 | 9A | Ü | 186 | BA | ║ | 218 | DA | ┌ | 250 | FA | · |
| 155 | 9B | ¢ | 187 | BB | ╗ | 219 | DB | █ | 251 | FB | √ |
| 156 | 9C | £ | 188 | BC | ╝ | 220 | DC | ▄ | 252 | FC | ⁿ |
| 157 | 9D | ¥ | 189 | BD | ╜ | 221 | DD | ▌ | 253 | FD | ² |
| 158 | 9E | ₧ | 190 | BE | ╛ | 222 | DE | ▐ | 254 | FE | ■ |
| 159 | 9F | ƒ | 191 | BF | ┐ | 223 | DF | ▀ | 255 | FF |  |

/!\ Attention le type **string** (ou chaîne de caractère) n'existe pas en C comme type de base. Pour déclarer un string il faut utiliser le type `char` de cette façon : `char * mot ;`

Exemple :

```
char c = "Hello" ;
printf("%c", c) ;
```

va retourner 'H'

```
char *s = "Hello" ; printf("%s", c) ;
```

va retourner 'Hello'

La chaine de caractère représente une zone mémoire de 5+1 caractères. Car le compilateur à besoin de rajouter en fin de chaîne le caractère '\0' qui indique la fin de la chaîne.

- **Les entiers** : le mot clé désignant les entiers est `int`. Il existe des trois niveaux de précision d'entiers :

  Les entiers de ce type sont signés, le mot clé `unsigned` précédent `int`, `short int` ou `long int` permet de définir l'objet comme entier non signé.

  Les valeurs limites des types entiers sont indiquées dans la <limits.h>.

- **Les flottants** : le mot clé désignant les flottants est `float`. Il existe trois niveaux de précision de flottants : `float`, `double`, `long double`. Les valeurs limites des types flottants sont dans <float.h>.

  Le `double` a été introduit par la norme ANSI pour remplacer l'ancienne notation en C `long float`.
  La notation décimale doit comporter un point. La partie décimale ou entière peuvent être omises mais pas toutes les deux en même temps :

  12.3 -0.56 -.2 4. .11

  `float x = 3 ;` // est traduit par le compilateur par `3.0`

  Il est possible d'utiliser l'exponentielle avec les caractères e ou E :
  float y = 4.25e3 ; // sera traduit par 4250.0 – équivalent à 4.25E3
  float y = 4.25e-3 ; // cette forme d'écriture est également correcte

  Quelques particularités des flottants en C :

  - float a, b ; l'approximation de a + l'approximation de a b n'est pas forcément égale à l'approximation de (a+b).
  - 4/5 vaut 0.000000 alors que (float)4/(float)5 vaudra 0.800000

Il n'existe pas de type **booleen** en C, tous les types numériques peuvent être utilisés pour exprimer des opérations logiques : le 0 exprimera le `Faux` et les autres valeurs exprimeront les `Vrai`.

# 2. Variables et Constantes

Les variables sont au cœur de la programmation impérative. Un programme ne peut fonctionner sans variables. Une *variable* associe un nom à une valeur. La variable n'est pas constante, sa valeur peut changer dans le programme.

## 2.1 Déclaration et initialisation de variables

La déclaration peut se faire comme suit :

`<Type> <nomvar> ;` déclaration d'une variable

`<Type> <nomvar1>, <nomvar2>, …, <nomvarN>;` déclaration d'une suite de variables

Dans ce cas l'initialisation des valeurs de variables peut se faire soit immédiatement à suite de la déclaration, soit dans le programme.

L'initialisation peut sinon avec la déclaration :

```
<Type> <nomvar> = <valeur>;
<Type> <nomvar1> = <valeur>, <nomvar2>, <nomvar3> = <valeur> ;
```

Le nom de la variable est définit par le programmeur. Il est recommandé de choisir des noms de variables courts, et en lien avec la donnée manipulée. Par exemple si vous implémentez un compteur vous pouvez déclarer une variable cpt ou compteur.

Ci-dessous le tableau des règles de syntaxe d'un nom de variable :

| Identificateurs corrects | Identificateurs incorrects | Pourquoi ? |
|---|---|---|
| nom1 | 1nom | Ne peut commencer par un chiffre |
| nom_2 | nom.2 | Les points ne sont pas autorisés |
| _nom_3 | -nom-3 | Les tirets ne sont pas autorisés |
| nom_de_variable | Nom de variable | Les espaces ne sont pas autorisés |
| deuxieme_choix | deuxième_choix | Les caractères accentués ne sont pas autorisés |

| | | |
|---|---|---|
| mot_francais | mot_français | Les cédilles ne sont pas autorisées |

La variable peut être déclarée avec les types que nous avons définis en début de chapitre.

### 2.2 Portée d'une variable

- **locale** : la variable n'est visible que dans le fichier ou le bloc (espace entre deux accolades {} de même niveau) où elle est définie.
- **globale** : la variable est visible dans tous le fichier source, mais aussi dans les autres fichiers sources du projet.

/!\ *Qualité : l'usage des variables globales doit être limité, elles peuvent créer des problèmes* :

  - dits d'*effets de bord* : elles peuvent être modifiées par erreur par une autre fonction;
  - à l'édition de lien : définition multiple d'un symbole.

### 2.3 Déclaration et initialisation de constantes

Les constantes sont proches des variables, à un point près : elles ne peuvent pas être modifiées dans le programme.

Pour définir une constante, sa déclaration doit se faire en même temps que son initialisation. La déclaration doit être précédée du mot clé `const` :

```
const <type> <varName> = <valeur>;
```
Déclaration et initialisation d'une seule constante

```
const <type> <varName1> = <valeur>, <varName1> = <valeur>;
```
Déclaration et initialisation de plusieurs constantes de même type Une constante peut être de tout type de base.

Il existe par ailleurs des constantes prédéfinie en C, telles que :
- `EXIT_SUCCESS` : vaut 1. Et est utilisée par le return du main() généralement.
- `INT_MIN,INT_MAX` : indiquent les valeurs limites du type entier. Elles sont définies dans <limits.h>.

## 3. Les opérateurs

Les opérateurs sont des signes servant à calculer avec les variables. Les six opérateurs les plus importants sont :

| Signe | Nom | Exemple |
|---|---|---|
| + | addition | int a = 4 + 10 //a vaut 14 |
| - | soustraction | int a = 9 - 6 //a vaut 3 |
| * | multiplication | int a = 8 * 9 //a vaut 72 |
| / | division | int a = 9 / 3 //a vaut 3 |
| % | modulo (reste de la divison entiere) | int a = 15 % 9 //a vaut 6 |
| = | affectation | var1 = 3 |

Il y a un ordre de priorités de traitement des opérateurs, il est donc important de forcer cet ordre en délimitant par des parenthèses. Exemple :
`1 + 3 * 5` donnera 16
`(1+3)*5` donnera 20

## 3.1 Opérateurs d'accumulation

Ce type d'opérateurs modifie la valeur de la variable. Exemple : `int a = 5 ;`
`a *= 2 ;` équivaut à `a = a * 2`
`printf(" %d", a) ;` va retourner 10

Ci-dessous la liste des opérateurs d'accumulation :

| Opérateur | Description | Exemple (int i vaut 10) |
|---|---|---|
| += | addition | i += 23 /* i vaut 33 */ |
| -= | soustraction | i -= 5 /*i vaut 5 */ |
| *= | multiplication | i *= 10 /* i vaut 100 */ |
| /= | division | i /= 2 /* i vaut 5 */ |
| %= | modulo | i %= 6 /* i vaut 4 */ |

On appelle *incrémentation*, le fait d'ajouter 1 à un entier. Voici les trois formes d'écriture d'une incrémentation :
`x = x +1 ; x += 1 ; x++ ;`

## 3.2 Opérateurs binaires

Ils servent à manipuler les bits de valeurs entières. Si ces opérateurs sont utilisés sur des valeurs entières, ils opéreront au niveau des bits au lieu de la valeur numérique.

| Opérateur | Rôle | Exemple |
|---|---|---|
| & | et binaire bit à bit | 1010 1101 & 1111 1110 --> 1010 1100 |
| \| | ou binaire bit à bit | 1010 1100 \| 0000 0001 --> 1010 1101 |
| ^ | ou exclusif bit à bit | 1010 1101 ^ 0000 0001 --> 1010 1100 |
| ~ | Complément à un | ~1010 1101 --> 0101 0010 |
| << | Décalage de n bits vers la gauche | 1010 1101 >> 1 --> 0101 0110 |
| >> | Décalage de n bits vers la droite | 1010 1101 << 1 --> 0101 1010 |

# 4. Les fonctions printf() et scanf()

La fonction `printf()` permet d'afficher une chaîne de caractères à l'écran pendant l'exécution du programme.
La fonction `scanf()` permet de lire des valeurs saisies durant l'exécution du programme.

## 4.1 printf()

Syntaxe : `printf(format[, liste_d_expressions])`

| | |
|---|---|
| `Format` | C'est une chaîne de caractères. |
| `Liste_d_expressions` | suite d'expressions séparées par des virgules. Chaque expression doit être du même type que le code de format correspondant.<br>Peut être absente, dans le cas ou le format ne contient aucun code de format. |

La valeur de retour de `printf` est le nombre de caractères qu'elle a écrit. Ce nombre ne peut être obtenu qu'après l'exécution de la fonction.

Les différents codes de formats :

| format | Valeurs de n | Affichage | Remarque |
|---|---|---|---|
| `printf("%d", n)` | 20<br>3<br>2358<br>-5200 | 20<br>3<br>2358<br>-5200 | |

| | | | |
|---|---|---|---|
| `printf("%3d", n)` | 20<br>3<br>2358<br>-5200 | 020<br>003<br>2358<br>-5200 | Entier sur 3 caractères min |
| `printf("%f", n)` | 1.2345<br>12.3456789<br>0.000012345<br>1e-10 | 1.234500<br>12.345679<br>0.000012<br>0. 000000 | 6 chiffres après le point<br>Arrondi à 6 chiffres<br>" |
| `printf("%10f", n)` | 1.2345<br>12.3456789<br>0.000012345<br>1e-10 | 001.234500<br>012.345679<br>000.000012<br>000. 000000 | 6 chiffres après le point et sur 10 caractères minimum |
| `printf("%.3f", n)` | 1.2345<br>12.3456789<br>0.000012345 | 1.235<br>12.346<br>0.000 | Arrondi à 3 chiffres après le point |
| `printf("%10.3f", n)` | 1.2345 | 000001.235 | 3 chiffres après le point et sur 10 caractères minimum |
| `printf("%e", n)` | 1.2345 123.45<br>123.456789E8 | 1.234500e+00<br>1.234500 e+02<br>1.234568 e+10 | Notation exponentielle<br><br>Arrondi à 6 chiffres et exponentielle |
| `printf("%s", n)` | `"Bonjour"` | `Bonjour` | Chaine de caractères |
| `printf("%c", n)` | `"Bonjour"` | `B` | Un caractère |

## 4.2 scanf()

Syntaxe : `scanf(format[, liste_d_expressions])`

| | |
|---|---|
| `Format` | C'est une chaîne de caractères. |
| `Liste_d_expressions` | suite d'expressions séparées par des virgules. Chaque expression doit être du même type que le code de format correspondant.<br>Peut être absente, dans le cas ou le format ne contient aucun code de format. |

La valeur de retour de `scanf` est le nombre de caractères qu'elle a écrit. Ce nombre ne peut être obtenu qu'après l'exécution de la fonction.
Lorsque `scanf` lit des informations au clavier, il attend la saisie d'une touche dite de validation pour finir la lecture. Tant que cette touche n'a pas été détectée, il est possible de modifier les valeurs saisies. Ceci est possible, grâce à l'utilisation de mémoire tampon. La validation introduit dans la mémoire tampon, un caractère de fin de ligne. Dans certains cas comme le %c il pourra apparaitre comme un caractère à part entière.

Les données saisies, étant stockées en mémoire tampon, `scanf` associe le contenu d'emplacements mémoire à des variables, préalablement déclarées. Il utilise la notion de pointeur que nous verrons aux prochains Chapitres.
La valeur de retour de `scanf` permet de savoir si la lecture s'est bien déroulée. Elle renvoie le nombre de valeurs lues et affectées à la liste d'expressions.

Les différents codes de formats :

| format | Saisie | Affectation |
|---|---|---|
| `printf("%d%c", &n, &c)` | 20a<enter><br>20 a<enter> | n= 20 , c=a<br>n= 20 , c=a |
| `printf("%d%d", &n,&m)` | 20 45e10<enter> | n= 20 , m=45 |
| `printf("%d%d %c", &n, &m, &c)` | 12 25 b<enter><br>12b<enter><br>b<enter> | n= 12 , m = 25, c=b<br>n= 12 , m = inchangé, c=inchangé n= inchangé , m = inchangé, c=inchangé |

Liste non exhaustive des codes de formats pour `scanf` :

| Code de format | Type |
|---|---|
| `%c` | `char` |
| `%d ou %i` | `int` |
| `%f, %e,%E, %g, %G` | `float` |
| `%lf, %le,%lE, %lg, %lG` | `double` |
| `%Le,%LE, %Lg, %LG` | `long double` |

# 5. Les fonctions

Les fonctions sont des sous programmes qui permettent d'effectuer un ensemble d'instructions seulement lorsque la fonction est appelée dans le corps du programme principal.

Une fonction peut également être appelée par une autre fonction. Et même plusieurs fois dans le programme.