



# MINDS' ACADEMY

*Elaboré Par Mortadha*

Niveau	1A&3B
Chapitre 1	Initiation
Objectif	<ul style="list-style-type: none"><li>• Connaître la structure d'un programme C.</li><li>• Manipuler les variables et les opérateurs de base en C.</li><li>• Connaître et utiliser les fonctions de lecture et d'écriture en C</li></ul>



# Table des matières

## 1 Introduction

1.1 Quelques repères historiques

1.2 Compilation

## 2 Les composants élémentaires du C

2.1 Les identificateurs

2.2 Les mots-clefs

2.3 Les commentaires

2.4 Premier programme en C

2.4.1 Préprocesseur

2.4.2 La fonction main

2.4.3 Instruction

## 3 Types et variables

3.1 Les types prédéfinis

3.1.1 Le type caractère

3.1.2 Les types entiers

3.1.3 Les types flottants

3.2 Les constantes

## 4 Lire et écrire des données

4.1 Ecriture formatée de données

4.2 Lecture formatée de données

## 5 Les opérateurs

5.1 Les opérateurs arithmétiques

5.2 Les opérateurs relationnels

5.3 Les opérateurs logiques booléens

5.4 Les opérateurs d'affectation composée

5.5 Les opérateurs d'incrémentation et de décrémentation

# Introduction

## 1.1 Quelques repères historiques

---

Le C a été conçu en 1972 par Dennis Richie et Ken Thompson, chercheurs aux Bell Labs, afin de développer un système d'exploitation UNIX sur un DEC PDP-11. En 1978, Brian Kernighan et Dennis Richie publient la définition classique du C dans le livre The C Programming language .

Le C devenant de plus en plus populaire dans les années 80, plusieurs groupes mirent sur le marché des compilateurs comportant des extensions particulières. En 1983, l'ANSI (American National Standards Institute) décida de normaliser le langage ; ce travail s'acheva en 1989 par la définition de la norme ANSI C. Celle-ci fut reprise telle quelle par l'ISO (International Standards Organization) en 1990. C'est ce standard, ANSI C, qui est décrit dans le présent document.

## 1.2 Compilation

---

Le C est un langage compilé . Cela signifie qu'un programme C est décrit par un fichier texte, appelé fichier source. Ce fichier n'étant évidemment pas exécutable par le microprocesseur, il faut le traduire en langage machine. Cette opération est effectuée par un programme appelé compilateur. La compilation se décompose en fait en 4 phases successives :

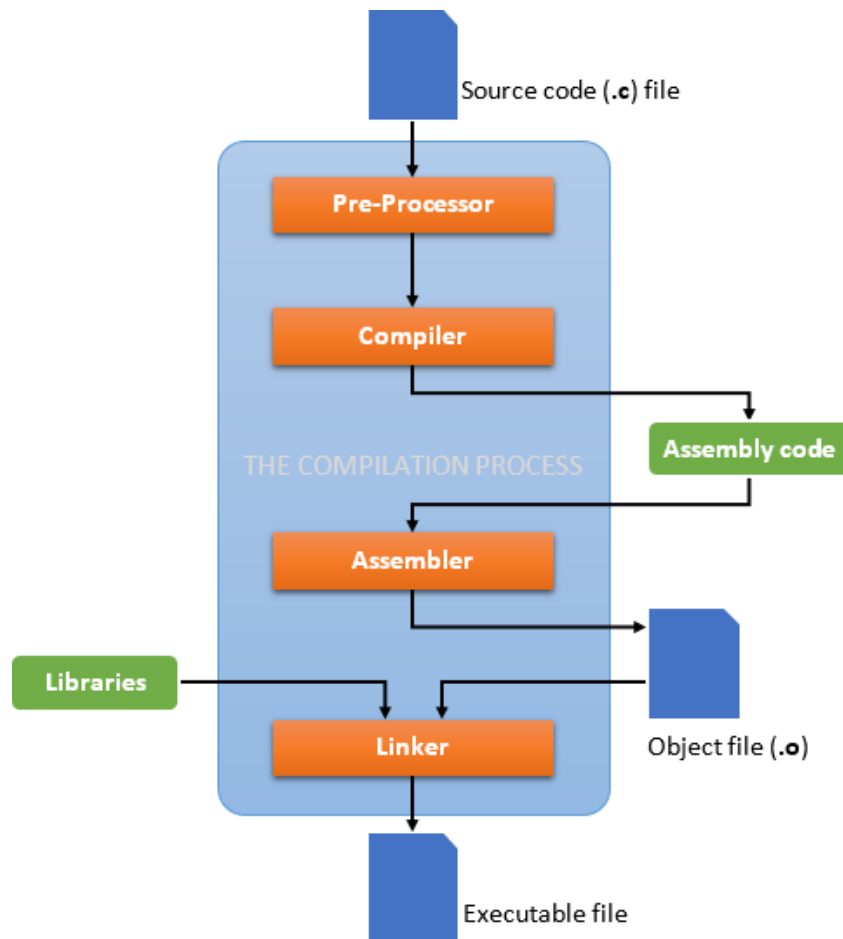
**1. Le traitement par le préprocesseur** : le fichier source est analysé par le préprocesseur qui effectue des transformations purement textuelles .

**2. La compilation** : la compilation proprement dite traduit le fichier généré par le préprocesseur en assembleur.

**3. L'assemblage** : cette opération transforme le code assembleur en un fichier binaire, c'est-à-dire en instructions directement compréhensibles par le processeur. Le fichier produit par l'assemblage est appelé fichier objet.

**4. L'édition de liens** : un programme est souvent séparé en plusieurs fichiers source, pour des raisons de clarté mais aussi parce qu'il fait généralement appel à des bibliothèques de fonctions standard déjà écrites. Une fois chaque code source assemblé, il faut

donc lier entre eux les différents fichiers objets. L'édition de liens produit alors un fichier dit exécutable.



Les différents types de fichiers utilisés lors de la compilation sont distingués par leur suffixe. Les fichiers source sont suffixés par `.c`, les fichiers prétraités par le préprocesseur par `.i`, les fichiers assembleur par `.s`, et les fichiers objet par `.o`. Les fichiers objets correspondant aux librairies pré-compilées ont pour suffixe `.a`

# Les Composants Élémentaires du C

## 1.1 Les identificateurs

Le rôle d'un identificateur est de donner un nom à une entité du programme. Plus précisément, un identificateur peut désigner :

- un nom de variable ou de fonction,
- un type défini par typedef, struct, union ou enum,
- une étiquette.

Un identificateur est une suite de caractères parmi :

- les lettres (minuscules ou majuscules, mais non accentuées),
- les chiffres,
- le ``blanc souligné" (\_).

Le premier caractère d'un identificateur ne peut pas être un chiffre. Par exemple, var1, tab\_23 ou \_deb sont des identificateurs valides ; par contre, 1i et i; ne le sont pas. Il est cependant déconseillé d'utiliser « \_ » comme premier caractère d'un identificateur car il est souvent employé pour définir les variables globales de l'environnement C.

**ATTENTION : Les majuscules et minuscules sont différenciées.**

## 2.2 Les mots-clefs

Un certain nombre de mots, appelés mots-clefs, sont réservés pour le langage lui-même et ne peuvent pas être utilisés comme identificateurs.

L'ANSI C compte 32 mots clefs :

auto	double	int	struct
break	else	long	switch



## 2.3 Les commentaires

Les commentaires sont destinés à faciliter la compréhension du source lors de la relecture. Ils ne sont d'aucune utilité au compilateur, et il est naturel qu'ils n'apparaissent pas dans le source qui lui est destiné. Le préprocesseur retire les caractères compris entre `/*` et `*/`. Il ne gère pas les imbrications de commentaires. La mise en commentaire d'une section source peut alors créer des problèmes.

### Exemple:

```
/* ceci est un commentaire correct */  
/* ceci est /* évidemment */ incorrect */
```

## 2.4 Premier programme en C

```
1 | #include <stdio.h> } Directives de préprocesseur  
2 | int main()  
3 | {  
4 |     printf("Minds Academy!\n"); } Instructions  
5 |     return 0;  
6 | } } Fonctions  
7 |
```

### Resultat:

```
Minds Academy!  
  
Process returned 0 (0x0)   execution time : 0.027 s  
Press any key to continue.
```

## 2.4.1 Préprocesseur

Le préprocesseur effectue un prétraitement du programme source avant qu'il soit compilé. Ce préprocesseur exécute des instructions particulières appelées directives. Ces directives sont identifiées par le caractère # en tête.

Inclusion de fichiers

```
#include <nom-de-fichier> /* répertoire standard */  
#include "nom-de-fichier" /* répertoire courant */
```

**La gestion des fichiers (stdio.h) /\* Entrees-sorties standard**  
**\*/ Les fonctions mathématiques (math.h)**  
**Traitement de chaînes de caractères (string.h)**  
**Utilitaires généraux (stdlib.h)**  
**Date et heure (time.h)**

## 2.4.2 La fonction main

La fonction main est la fonction principale des programmes en C: Elle se trouve obligatoirement dans tous les programmes. L'exécution d'un programme entraîne automatiquement l'appel de la fonction main.

**int main() VS void main():**

Le void main() indique que la fonction main() ne retournera aucune valeur, mais le int main() indique que la main() peut retourner des données de type entier. Lorsque notre programme est simple et qu'il ne va pas se terminer avant d'avoir atteint la dernière ligne du code, ou que le code est sans erreur, alors nous pouvons utiliser le void main(). Mais si nous voulons terminer le programme en utilisant la méthode exit(), nous devons retourner des valeurs entières (zéro ou non). Dans cette situation, le void main() ne fonctionnera pas. Il est donc recommandé d'utiliser int main() sur le void main().

## 2.4.3 Instructions

Voyons maintenant plus précisément le contenu de la ligne 4 qui réalise l'affichage de la phrase "Minds Academy" sur une console. La commande utilisée pour afficher la chaîne de caractères est l'instruction printf(...), la chaîne à afficher étant entre guillemets. Le caractère \n placé à la fin du chaîne indique qu'un saut de ligne doit être réalisé avant d'afficher la phrase.

## 3.1 Les types prédéfinis

Le C est un langage typé. Cela signifie en particulier que toute variable, constante ou fonction est d'un type précis. Le type d'un objet définit la façon dont il est représenté en mémoire. La mémoire de l'ordinateur se décompose en une suite continue d'octets. Chaque octet de la mémoire est caractérisé par son adresse, qui est un entier. Deux octets voisins en mémoire ont des adresses qui diffèrent d'une unité. Quand une variable est définie, il lui est attribué une adresse. Cette variable correspondra à une zone mémoire dont la longueur (le nombre d'octets) est fixée par le type.

La taille mémoire correspondant aux différents types dépend des compilateurs ; toutefois, la norme ANSI spécifie un certain nombre de contraintes. Les types de base en C concernent les caractères, les entiers et les flottants (nombres réels). Ils sont désignés par les mots-clefs suivants :

*char int float double short long unsigned*

### 3.1.1 Le type caractère

Le mot-clef `char` désigne un objet de type caractère. Un `char` peut contenir n'importe quel élément du jeu de caractères de la machine utilisée. La plupart du temps, un objet de type `char` est codé sur un octet ; c'est l'objet le plus élémentaire en C. Le jeu de caractères utilisé correspond généralement au codage ASCII (sur 7 bits). La plupart des machines utilisent désormais le jeu de caractères ISO-8859 (sur 8 bits), dont les 128 premiers caractères correspondent aux caractères ASCII. Les 128 derniers caractères (codés sur 8 bits) sont utilisés pour les caractères propres aux différentes langues.

### 3.1.2 Les types entiers

Le mot-clef désignant le type entier est `int`. Un objet de type `int` est représenté par un mot "naturel" de la machine utilisée. Le type `int` peut être précédé d'un attribut de précision (`short` ou `long`) et/ou d'un attribut de représentation (`unsigned`). Un objet de type `short int` a au moins la taille d'un `char` et au plus la taille d'un `int`. En général, un `short int` est codé sur 16 bits. Un objet de type `long int` a au moins la taille d'un `int`.

### 3.1.3 Les types flottants

Les types float, double et long double servent à représenter des nombres en virgule flottante. Ils correspondent aux différentes précisions possibles.

#### Déclaration de variables

Syntaxe: *type nom;*

On peut déclarer plusieurs variables d'un même type: Exemple:

*int a, b, c;*

On peut initialiser une variable lors de sa déclaration: Exemple:

*float pi = 3.14;*

Type de donnée	Signification	Taille (en octets)	Plage de valeurs acceptée
char	Caractère	1	-128 à 127
unsigned char	Caractère non signé	1	0 à 255
short int	Entier court	2	-32 768 à 32 767
unsigned short int	Entier court non signé	2	0 à 65 535
int	Entier	2 (sur processeur 16 bits) 4 (sur processeur 32 bits)	-32 768 à 32 767 -2 147 483 648 à 2 147 483 647
unsigned int	Entier non signé	2 (sur processeur 16 bits) 4 (sur processeur 32 bits)	0 à 65 535 0 à 4 294 967 295
long int	Entier long	4	-2 147 483 648 à 2 147 483 647
unsigned long int	Entier long non signé	4	0 à 4 294 967 295
float	Flottant (réel)	4	$-3.4 \cdot 10^{-38}$ à $3.4 \cdot 10^{38}$
double	Flottant double	8	$-1.7 \cdot 10^{308}$ à $1.7 \cdot 10^{308}$
long double	Flottant double long	10	$-3.4 \cdot 10^{4932}$ à $3.4 \cdot 10^{4932}$

## 3.2 Les constantes

Une constante est une valeur qui apparaît littéralement dans le code source d'un programme, le type de la constante étant déterminé par la façon dont la constante est écrite. Les constantes peuvent être de 4 types : entier, flottant (nombre réel), caractère. Ces constantes vont être utilisées, par exemple, pour initialiser une variable.

**Syntaxe :** *#define NOM valeur*

**Exemple 1:** *#define pi 3.1416*

**Exemple 2:** *#define N 20*

On pourra ensuite utiliser N qui est initialisé à la valeur 20, on ne peut plus changer sa valeur.

# Lire et écrire des données

La bibliothèque standard `<stdio.h>` contient un ensemble de fonctions qui assurent la communication de la machine avec le monde extérieur.

Les fonctions les plus importantes sont:

## Pour la lecture:

*printf()*: écriture formatée de données

## Pour l'écriture:

*scanf()*: lecture formatée de données

## 4.1 Ecriture formatée de données

*printf()*: cette fonction est utilisée pour transférer du texte, des valeurs de variables ou des résultats d'expression vers le fichier de sortie standard stdout (par défaut l'écran).

### Syntaxe:

```
printf("format", expr_1, expr_2);
```

↑

↑

↑

Format de                      Expressions ou variables dont les valeurs sont à  
représenter                      représentation

"format" est une chaîne de caractère qui peut contenir:

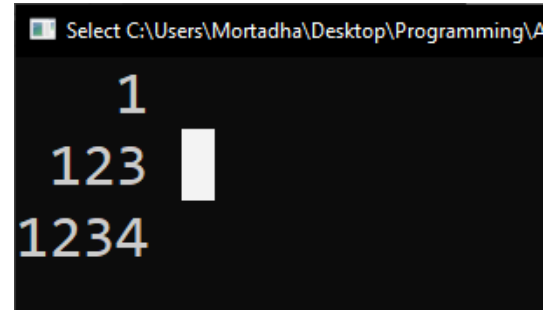
- du texte
- des séquences d'échappement
- des spécificateurs de format (un spécificateur pour chaque expression) Les spécificateurs de format: ils commencent toujours par le symbole %

Symbole	Impression comme	Type
%d ou %i	Entier relatif	int
%u	Entier naturel (non signé)	int
%o	Entier exprimé en octal	int
%x	Entier exprimé en hexadécimal	int
%f	Rationnel en notation décimale	float
%e	Rationnel en notation scientifique	float
%g	Rationnel en notation décimale/scientifique	float
%lf	Rationnel en notation décimale	double
%lg	Rationnel en notation décimale/scientifique	double
%le	Rationnel en notation scientifique	double
%c	Caractère	char
%s	Chaîne de caractère	char*

## Largeur minimale des entiers

Il est possible d'indiquer la largeur minimale de la valeur à afficher. Dans le champ ainsi réservé, les nombres sont justifiés à droite:

```
1 #include <stdio.h>
2 int main()
3 {
4     printf("%4d", 1);
5     printf("\n"); //retour à la ligne
6     printf("%4d", 123);
7     printf("\n"); //retour à la ligne
8     printf("%4d", 1234);
9     printf("\n"); //retour à la ligne
10    return 0;
11 }
12
```



## - Largeur minimale et précision pour les rationnels

La précision par défaut est de 6 décimales.

```
1 #include <stdio.h>
2 int main()
3 {
4     printf("%f", 12.34);
5     return 0;
6 }
7
```

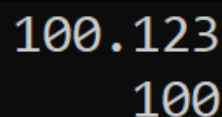


La syntaxe est: "%n.mf"

où: n est la largeur du  
champ

m est le nombre de décimales

```
1 #include <stdio.h>
2 int main()
3 {
4     printf("%10.3f", 100.123);
5     printf("\n");
6     printf("%10.f", 100.123);
7     return 0;
8 }
9
```





## 4.2 Lecture formatée de données

**scanf()**: fonction symétrique de printf().

### Syntaxe:

scanf("format",                    adr\_var\_1,        adr\_var\_2)

↑

↑

↑

Format de  
attribution

Adresses des variables auxquelles les données sont

lecture des données

(adresse d'une variable= nom de la variable précédé de  
&)

- La fonction scanf reçoit ses données à partir du fichier standard stdin (le clavier)
- La chaîne de format détermine comment les données reçues doivent être interprétées
- Les données reçues correctement sont mémorisées aux adresses indiquées par adr\_var\_1, adr\_var\_2

Les spécificateurs de format pour scanf sont:

Symbole	Lecture d'un(e)	Type
%d ou %i	Entier relatif	int
%u	Entier naturel (non signé)	int
%o	Entier exprimé en octal	int
%b	Entier exprimé en hexadécimal	int
%f	Rationnel en notation décimale	float
%e	Rationnel en notation scientifique	float
%g	Rationnel en notation décimale/scientifique	float
%lf	Rationnel en notation décimale	double
%lg	Rationnel en notation décimale/scientifique	double
%le	Rationnel en notation scientifique	double
%c	Caractère	char
%s	Chaîne de caractère	char*

scanf("%d", &nombre);

On entre au clavier 33, nombre = 33

On peut traiter plusieurs variables avec une seule instruction scanf: Lors de l'entrée des données, une suite de signes d'espacement (espace, tab, interligne) est évaluée comme un seul espace (idem si dans la chaîne de format on tape les symboles \n, \t, \r =1 seul espace). Si la chaîne de format contient aussi d'autres caractères que des signes d'espacement, alors ces symboles doivent être introduits exactement dans l'ordre indiqué.

## Exemple

```
int jour, mois, annee;  
scanf("%d/%d/%d",&jour, &mois,  
&annee);
```

Entrées acceptées

24/11/1973

Entrées rejetées

24 11 1973

# Les opérateurs

## 5.1 Les opérateurs arithmétiques

Les opérateurs arithmétiques classiques sont l'opérateur unaire - (changement de signe) ainsi que les opérateurs binaires

- + addition

- soustraction

- \* multiplication

- / division

- % reste de la division (modulo)

Ces opérateurs agissent de la façon attendue sur les entiers comme sur les flottants. Leurs seules spécificités sont les suivantes :

Contrairement à d'autres langages, le C ne dispose que de la notation / pour désigner à la fois la division entière et la division entre flottants. Si les deux opérandes sont de type entier, l'opérateur / produira une division entière (quotient de la division). Par contre, il délivrera une valeur flottante dès que l'un des opérandes est un flottant. Par

exemple,

```
float x;
```

```
x = 3 / 2;
```

affecte à x la valeur 1. Par

contre 

```
x = 3 / 2.;
```

affecte à x la valeur 1.5.

L'opérateur % ne s'applique qu'à des opérandes de type entier. Si l'un des deux opérandes est négatif, le signe du reste dépend de l'implémentation, mais il est en général le même que celui du dividende. Notons enfin qu'il n'y a pas en C d'opérateur effectuant l'élévation à la puissance.

De façon générale, il faut utiliser la fonction `pow(x,y)` de la librairie `math.h` pour calculer  $x^y$ .

## 5.2 Les opérateurs relationnels

> supérieur

>= supérieur ou égal

< strictement inférieur

<= inférieur ou égal

== égal

!= différent

Leur syntaxe est `expression1 op expression2`

Les deux expressions sont évaluées puis comparées. La valeur rendue est de type `int` (il n'y a pas de type booléen en C); elle vaut 1 si la condition est vraie, et 0 sinon.

**ATTENTION** : à ne pas confondre l'opérateur de test d'égalité `=` avec l'opérateur d'affectation `=`

## 5.3 Les opérateurs logiques booléens

`&&` et logique

`||` ou logique

`!` négation logique

Comme pour les opérateurs de comparaison, la valeur retournée par ces opérateurs est un `int` qui vaut 1 si la condition est vraie et 0 sinon.

Dans une expression de type

`expression1 op1 expression2 op2 ...expressionN`

l'évaluation se fait de gauche à droite et s'arrête dès que le résultat final est déterminé.

## 5.4 Les opérateurs d'affectation composée

Les opérateurs d'affectation composée sont

**$+=$**

**$-=$**

**$*=$**

**$/=$**

**$\%=$**

Pour tout opérateur  $op$ , l'expression  
 $expression1\ op = expression2$   
est équivalente à

$expression1 = expression1\ op\ expression2$

Toutefois, avec l'affectation composée,  $expression1$  n'est évaluée qu'une seule fois.

## 5.5 Les opérateurs d'incrémentation et de décrémentation

Les opérateurs d'incrémentation ++ et de décrémentation -- s'utilisent aussi bien en suffixe (i++) qu'en préfixe (++i). Dans les deux cas la variable i sera incrémentée, toutefois dans la notation suffixe la valeur retournée sera l'ancienne valeur de i alors que dans la notation préfixe se sera la nouvelle. Par exemple:

```
int a = 3, b, c;
```

```
b = ++a; /* a et b valent 4 */
```

```
c = b++; /* c vaut 4 et b vaut 5 */
```