

---

# PRESENTATION DU LANGAGE C

---

**L**e langage C est un langage algorithmique de la famille Algol, plus particulièrement évolué à partir des langages orientés programmation système. En fait, il descend en droite ligne de CPL (Christopher's Programming Language) par BCPL, puis B qui est un "petit" BCPL écrit par Ken Thomson.

Ses principales qualités sont une portabilité prouvée, une syntaxe concise et régulière.

## Premier Aperçu Des Objets

L'objet déclaré par

```
external char (*fun[])()
```

est un tableau de pointeurs vers des fonctions retournant des caractères et s'utilise comme cela :

```
c = (*fun[tty1])();
```

Les objets peuvent s'adapter à tous les besoins, et une définition d'objet permet de les combiner naturellement.

Exemple :

```
struct {
    char *name;
    void *func();
    KEY key;
} major_model[] = {
    "efface_ligne", eff_ligne, CTRL + 'k';
    ....
}
```

Ceci définit un tableau de structures appelé `major_mode` dont la description est :

- un élément de pointeur vers des caractères;
- un élément de pointeur vers des fonctions;

- un élément de type KEY défini par l'utilisateur;

et initialisé par des triplets "chaîne de caractère", "adresse d'une fonction", "objet de type KEY".

La taille n'est pas définie, c'est le compilateur qui la calcule à partir de l'initialisation.

Enfin, un ensemble riche et puissant d'opérateurs encourage et facilite une programmation efficace et fonctionnelle.

Exemple :

```
while (*++ p) *++ q = *p;
```

ceci copie de manière efficace une chaîne de caractères p dans une chaîne de caractères q.

Ce sont ses qualités qui permettent d'écrire en C un interprète Lisp dont la vitesse est à 25% près la vitesse des meilleurs interprètes écrits en assembleur.

Toutefois, ce serait une grave erreur de prendre C pour un assembleur portable. C'est avant tout un langage de haut niveau manipulant des objets abstraits et complexes. C'est cette qualité qui a permis à l'Université de Yale d'écrire un compilateur Ada portable en 45000 lignes, alors que le projet officiel promet un compilateur Ada en Ada en 80000 à 120000 lignes.

## Les Objets De Base De C

### Les caractères

```
char c ;
```

Ceci déclare un caractère de nom c. C'est une entité stockée sur huit bits, représentant l'index du caractère dans la table des codes.

### Les entiers

```
short s;
```

définit un entier signé de taille  $\leq 16$  bits.

```
int i;
```

définit un entier signé de la taille naturelle pour la machine et d'au moins 16 bits.

```
long l;
```

définit un entier signé d'au moins 32 bits.

Les déclarations suivantes ont une signification identique aux précédentes, mais les objets ainsi définis se manipulent en arithmétique non signée.

- unsigned short us;
- unsigned int ui; ou encore: unsigned ui;
- unsigned long ul;

## Les réels

```
float f;
```

définit un nombre flottant f d'au moins 32 bits;

```
double d;
```

définit un nombre flottant de la taille "deux fois un float".

## Les fonctions

```
func();
```

Une fonction comme cela, sans rien, ni arguments ni initialisations, comme par exemple la fonction ci-dessous :

```
func(){printf("salut vaste monde\n");}
```

Une fonction simple initialisée avec le programme qui écrit sur la sortie standard "salut vaste monde" avec un retour chariot.

```
char func();
```

définit une fonction qui rend un objet de type caractère.

```
void func();
```

définit une fonction qui ne ramène rien, aussi appelée par certains informaticiens une procédure.

Note : La distinction entre procédure et fonction est aussi artificielle que la distinction entre 0 et tous les autres nombres. On peut la justifier en

disant que la nature à horreur du vide et donc rien ne doit pas être une chose définie - mais c'est purement mystique.

## Les Constructeurs d'Objets

### Les structures

```
struct {
    machin truc
    bidule chouette;
} objet;
```

définit un objet, appelé objet, constitué de deux parties : objet.truc et onbjet.chouette, de type machin et bidule.

### Les unions

```
union {
    int i;
    char c ;
} tantôt_l_un_tantôt_l_autre;
```

définit un objet variable : tantôt\_l\_un\_tantôt\_l\_autre.i est un entier et tantôt\_l\_un\_tantôt\_l\_autre.c est un caractère. La place utilisée en mémoire est la même pour les deux alternatives.

### Les énumérations

```
enum {bleu, blanc, rouge} nationalité;
```

définit un objet appelé nationalité pouvant prendre trois valeurs : soit bleu, soit blanc, soit rouge, isomorphe aux entiers 0, 1 et 2.

### Les tableaux

```
int ti[100];
```

définit un objet de cent objets (de type entier) numéroté de 0 à 99.

## Les pointeurs

```
truc *i;
```

définit un pointeur *i* vers des objets du type *truc*.

Un pointeur n'est pas une adresse machine, mais a des propriétés particulières. Par exemple, si *i* est un pointeur vers des entiers, l'instruction

```
i = i + 1;
```

fait pointer *i* vers l'entier suivant, alors que la même instruction sur un pointeur caractère fait pointer vers le caractère suivant. Le code généré n'est donc pas le même.

De nouveau : C sait combiner des choses.

## Les Structures De Contrôle

C fournit les structures de contrôle classiques de tout langage algorithmique. Ci-dessus, une brève énumération des diverses constructions disponible.

### le test

```
if (condition){
  ....
} else {
  ....
}
```

### les itérations

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

ou l'itération avec le test après la boucle :

```
do { ... } condition;
```

et

```
for(partie initialisation;
  test de fin d'iteration;
  action répétitive){
```

```

        .....
    }

```

Voici un exemple d'un petit algorithme de parcours d'une liste:

```

for(current= head;
   (*current).next != NULL;
   current = (*current).next){
    use_list(current);
}

```

## l'aiguillage

```

switch(sélecteur){
    case constante1: { ... }
    case constante2: { ... }
    .....
    default: { ... }
}

```

Contrairement à la plupart des langages, les constantes indiquent des points d'entrée, et si on ne prend pas de précautions particulières, le code exécuté est compris entre la constante d'entrée et la fin de l'instruction switch.

Finalement, les primitives de rupture de séquence. Tout d'abord la primitive :

```
exit(n);
```

qui arrête le programme totalement, et rend au langage de commande (au shell) le status n. Ensuite, la commande :

```
return(v);
```

qui arrête une fonction et retourne à l'appelant la valeur v.

```
break;
```

sort de la construction courante (case, while ou for) et continue à l'instruction suivante.

```
continue;
```

passse à la prochaine itération de l'instruction courante. Finalement, la commande :

```
goto foo; foo: .... ;
```

est – bien sûr – le goto bien connu, avec la restriction qu'il doit être local à la routine qui l'utilise.

## Les Opérateurs

Le langage C est particulièrement riche en opérateurs. Il y a:

les opérateurs arithmétiques ordinaires :

+ - * /	avec leurs significations habituelles
%	l'opérateur modulo

les opérateurs logiques bit à bit :

<<	shift gauche
>>	shift droit
&	et logique
	ou logique
^	ou exclusif

les opérateurs unaires :

-	moins
~	complément à un

les opérateurs logiques

&&	et logique
	ou logique
> >= < <=	avec leur signification habituelle
!=	différent
==	égal

notons que vrai est numériquement égale à 1 et faux est égale à 0.

les opérateurs d'adressage

&	l'opérateur unaire &f rend un pointeur sur l'objet f
*	l'opérateur unaire *f rend l'objet pointé par f

les opérateurs d'affectation

=	l'affectation simple
<b>op</b> =	avec op une des opérations suivante: +, -, /, *, %, >>, <<, &,  ,   , && alors

$a \text{ op} = b$

a la même signification que

$a = a \text{ op } b.$

++            opérateur d'incrément  
--            opérateur de décrémentation

finalement, un opérateur ternaire :

?:            ou  
               $a ? b : c$   
              rend b si a est vrai et c sinon.

