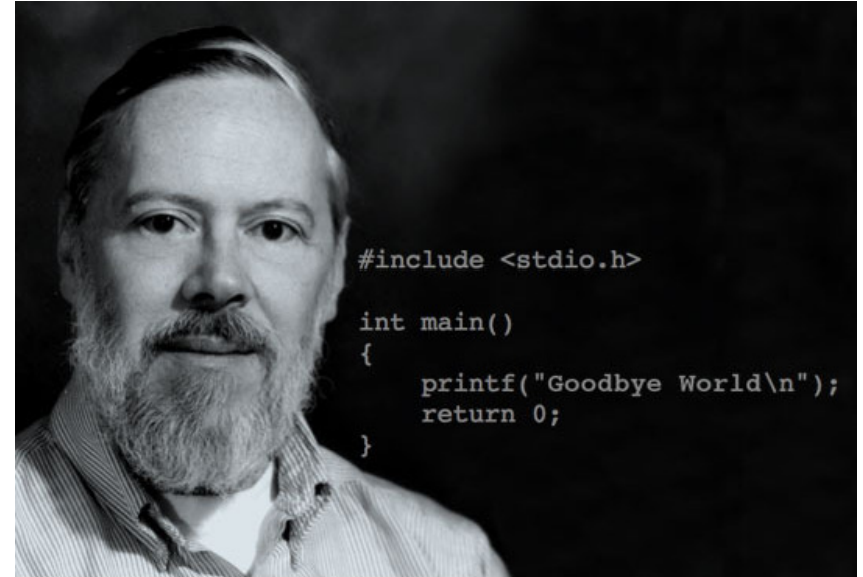


Introduction au Langage C

Langage C conçu 1972 par
Dennis Ritchie(1941-2011)

Objectif : écrire un système d'exploitation (UNIX)
Inspiré du langage B (K.Thompson)
Normalisé ANSI 1989, ISO1999 et ISO2011



Présentation du Langage C

Premier programme :

```
main()
{
    printf("bonjour " );
}
```

main : tête précise que ce qui suit est « programme principal ».
prog principal délimité par { }
" " guillemets ou *double quote* pour les chaînes de caractères.
; obligatoire après chaque instruction.

```
main()
{
printf("bonjour \n") ;
printf("monsieur ") ; }
```

\n caractère fin de ligne

Variables et leurs types :

```
main()
{ int n ;
  n=10 ;
printf(" valeur %d ",n) ;
}
```

Déclaration obligatoire

int n ; déclaration du type entier
; affectation

printf : fonction avec arguments

% : signifie que ce qui suit n'est pas un texte à afficher mais un code format
d : entier signé, u : entier non signé, c : caractère, s : chaîne de caractère
int n=10 ; déclaration et affectation.

Affichage de plusieurs valeurs

```
main()
{ int a,b,c ;
a=10 ; b=20 ; c=a+b ;
printf(" la somme de %d et %d est : %d ",a,b,c) ; }
```

la somme de 10 et 20 est : 30

nous aurions pu éviter de déclarer c :
printf(" la somme de %d et %d est : %d ",a,b,a+b) ;

Type caractère et format %c

```
main()
{ char x ;
x='e' ;
printf(" lettre = %c ",x) ;
}
```

'e' : caractère

"e" : chaine de caractères

Afficher un seul caractère : putchar

```
#include<stdio.h>
main()
{ char x ;
x='e' ;
printf(" lettre = ") ;
putchar(x) ;
}
```

Lire un seul caractère : getchar

```
#include<stdio.h>
main()
{ char c ;
printf(" donnez un
caractère : ") ;
c=getchar() ;
printf(" merci pour %c
",c) ;
}
```

[putchar/getchar/printf/scanf](#) des fonctions qui appartiennent à une biblio, leurs déclarations se trouvent dans le fichier entête (header) `stdio.h`

[#include<stdio.h>](#) demande d'incorporer à votre prog source, avant sa compilation le contenu du fichier entête `stdio.h` (qui contient tout ce qui nécessaire aux entrées/sorties).

Lire des informations de types quelconques : scanf

```
main()
{ int n,p ;
printf(" donnez deux nombres : ") ;
scanf("%d %d",&n,&p) ;
printf(" leur somme est %d",n+p) ; }
```

& : opérateur signifie adresse de

Remarque : si vous omettez & qui précède les noms de variables, le compilateur ne détectera aucune erreur, les informations seront rangées dans des emplacements aléatoires.

Instruction for pour les boucles

```
main()
{ int i,n ;
printf(" bonjour \n ") ;
printf(" je vais vous calculer 3 carrés \n ") ;
for (i=1 ;i<=3 ;i++)
{ printf(" donnez un nombre entier : ") ;
scanf(" %d",&n) ;
printf(" son carré est : %d\n",n*n) ;
}
printf(" au revoir ") ;
}
```

```
bonjour
je vais vous calculer 3 carrés
donnez un nombre entier : 3
son carré est : 9
donnez un nombre entier : 8
son carré est : 64
donnez un nombre entier : 5
son carré est : 25
au revoir
```

{ } : bloc , i=1 : initialisation, i<=3 : condition
i++ : (i=i+1) incrémentation après exécution du bloc

La directive #define pour définir une constante

```
#define NC 3
```

remplacer NC par 3 partout où il apparaît dans le prog

```
NC=NC+1 erreur
```

Instruction if

```
main()
```

```
{ int a,b,q,r ;
```

```
printf(" donnez deux entiers : ") ;
```

```
scanf(" %d %d ",&a,&b) ;
```

```
if (b !=0)
```

```
{ q=a/b ; r=a%b ;
```

```
printf(" division de %d par %d \n ",a,b) ;
```

```
printf(" %d=%d*%d+%d ",a,b,q,r) ; }
```

```
else printf(" diviseur nul ") ; }
```

donnez deux entiers : 37 8

division de 37 par 8

$37 = 8 * 4 + 5$

Règles générales d'écriture d'un prog en C :

Les identificateurs

Les identificateurs servent à désigner les différents objets manipulés :

- Commence nécessairement par une lettre (ou underscore_)
- Respecte la casse
- Au plus 31 caractères

ex : val, _Total2, Prix_unit

Mots clés :

auto	break	case	char	const	continue	default	do
double	else	enum	extern	float	for	goto	if
int	long	register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void	volatile	while

Les commentaires : texte explicatif destiné aux lecteurs:

`/* programme de calcul */`

`/* commentaire s'étendant sur
plusieurs lignes du programme source */`

`int nb ; // déclaration d'entier (commentaire de fin de ligne)`

Types de variable :

- **char** → caractères
- **int** → entiers
- **short** [int] → entiers courts
- **long** [int] → entiers longs
- **float** → nombres décimaux
- **double** → nombres décimaux de précision supérieure
- **long double** → nombres décimaux encore plus précis
- **unsigned int** → entier non signé

Types de base en Langage C

Types Entiers :

Type	description	taille	Valeurs limites
long unsigned long	entier long entier long non signé	4 octets	$-2^{31} \leq n \leq 2^{31} - 1$ $0 \leq n \leq 2^{32}$
short unsigned short	entier court signé entier court non signé	2 octets	$-2^{15} \leq n \leq 2^{15} - 1$ $0 \leq n \leq 2^{16}$
char unsigned char	caractère signé caractère non signé	1 octet	$-2^7 \leq n \leq 2^7 - 1$ $0 \leq n \leq 2^8$
int unsigned int	entier signé entier positif	dépend de la machine 2 ou 4 Octets	

```
#include<stdio.h>
main()
{
    short int n;
    n = 10 + 0xFF;
    printf(" Première Valeur : %d \n ",n);
    n = 10 + 0xFFFF;
    printf(" Seconde Valeur : %d \n ",n);
}
```

Première Valeur : 265 Seconde Valeur : 9

Les types caractères : le type `char` est un cas particulier du type entier(8bits),
exemple : `b` → 98

```
main() {  
    char u,v;  
    puts("bonjour");  
    u=65;v='A';  
    printf("premier affichage de u :%d \n ",u);  
    printf("deuxième affichage de u :%c \n ",u);  
    printf("premier affichage de v :%d \n ",v);  
    printf("deuxième affichage de v :%c \n ",v);  
}
```

```
bonjour  
premier affichage de u :65  
deuxième affichage de u :A  
premier affichage de v :65  
deuxième affichage de v :A
```

Les types réels : composés d'un signe, une mantisse (partie entière) et d'un exposant :

définition	précision	mantisse	domaine min	domaine max	nbr d'octet
float	simple	6	$3.4 * 10^{-38}$	$3.4 * 10^{38}$	4
double	double	15	$1.7 * 10^{-308}$	$1.7 * 10^{308}$	8
long double	suppl	19	$3.4 * 10^{-4932}$	$1.1 * 10^{4932}$	10

Entrées-Sorties conversationnelles :

La fonction **printf** a comme 1er argument une chaîne de caractères qui spécifie:

Des caractères à afficher tels quels;

Des **code de format** repérés par %.

Un code de format peut contenir des informations complémentaires agissant sur le : **cadrage**, le **gabarit** ou la **précision**.

Les principaux codes de conversion :

- **c** : char: caractère affiché "en clair" (convient aussi à short ou à int)
- **d** : int (convient aussi à char)
- **u** : unsigned int (convient aussi à unsigned char ou à unsigned short)
- **f** : double ou float écrit en notation "décimale" avec six chiffres après le point
- **e** : double ou float écrit en notation "exponentielle" (mantisse entre 1 et 9) avec six chiffres après le point décimal, sous la forme **x.xxxxxxe+yyy** .
- **s** : chaîne de caractères .
- **ld** : long
- **lu** : unsigned long

gabarit d'affichage: Les entiers sont affichés par défaut sans espaces avant ou après. Les flottants avec six chiffres après le point. Pour agir sur l'affichage, un nombre est placé après % et précise le nombre de caractère **minimum à utiliser.**

printf("%3d" , n);	n = 20	^20
	n = 3	^^3
	n = -5200	-5200

Les possibilités de la fonction **scanf** : Les principaux codes de conversion :

- **c** : char
- **d** : int
- **u** : unsigned int
- **hd** : short int
- **hu** : unsigned short
- **ld** : long
- **lu** : unsigned long
- **f** ou **e** : float écrit en notation "décimale" ou "exponentielle"
- **Lf** ou **le** : double écrit en notation "décimale" ou "exponentielle"
- **s** : chaîne de caractères dont on fournit l'adresse.

Opérateurs & Expressions :

- classiques (arithmétique, relationnels, logiques).
- moins classiques (manipulation de bits...)

priorité :

+ - : unaire ;

*** / ; binaire ;**

+ - : binaire ;

parenthèses

Exercice : type et résultats des expressions suivantes :

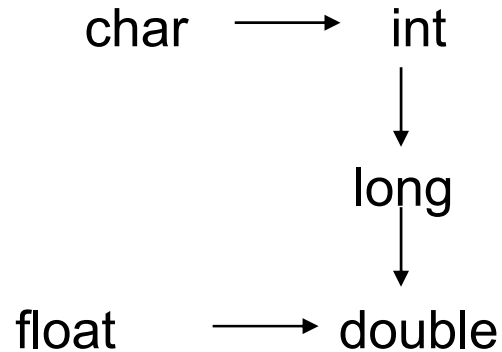
char c = '\x05' ;

int n = 5 ; long p = 100 ;

float x = 1.25 ; double z = 5.5 ;

$n + c + p$	$5 + 5 = 10$ int $10 + 100 = 110$ long
$2 * x + c$	$2 * 1.25 = 2.5$ float $5(\text{char}) \rightarrow 5(\text{int}) + 2.5 \rightarrow 7.5(\text{float})$
$z + n / 2$	$n / 2 = 5 / 2 = 2$ int $5.5 + 2(\text{int} \rightarrow \text{double}) = 7.5$ double

1-Règle générale de conversion (absorption) :



Horiz : systématique
Vert : ajustement de type

Prise en compte de l'attribut signe :

<pre>main() { int n ; signed char c ; c='\xfe'; n=c+1; printf ("%d", n); }</pre>	<pre>main() { int n ; unsigned char c ; c='\xfe'; n=c+1; printf ("%d", n); }</pre>
-128.....0.....127	0.....255
-1	255

Opérateurs Relationnels:

$<$, \leq , \geq , $>$: priorité 1

$=$, \neq : priorité 2

$a < b = c < d$ devient $(a < b) == (c < d)$

Résultat de la comparaison est non pas une valeur booléenne(vrai/faux)

mais un entier valant : 0 si résultat faux, 1 si résultat vrai

Résultat pourra intervenir dans des calculs arithmétiques $n = p + (a > b)$;

Opérateurs relationnels sont moins prioritaires que les opérateurs arithmétiques :

$$x + y < a + 2 \Leftrightarrow (x + y) < (a + 2)$$

Opérateurs Logiques :

C dispose des opérateurs : $\&\&$: et $\|\|$: ou $!$: négation

$(a < b) \&\& (c < d)$; $(a < b) \|\| (c < d)$; $!(a < b)$

Les opérateurs produisent un type numérique (int); C ne dispose pas de type logique.

si n et p sont des nombres quelconques :

les expressions : $n \|\| p$; $n \&\& p$; $!n$: sont acceptées.

$if (!n) \Leftrightarrow if (n == 0)$

$a < b \&\& c < d \Leftrightarrow ((a < b) \&\& (c < d))$

Opérateurs d'affectation :

$i = 5$; affectation simple

$c = b + 3$; évaluation et affectation

$c + 5 = x$; Impossible (fausse)

$i = j = 5$; Associativité (droite à gauche $j=5$ puis $i=5$)

Opérateurs d'incrément et de décrémentation

$i = i + 1$;

$n = n - 1$;

$++i$; pré incrément

$i++$; post incrément

si $i = 5$

$n = ++i - 5$;	$i=6$ $n=1$
$n = i++ - 5$;	$i=6$ $n=0$

Opérateurs de manipulations de bits : (type entier)

Type	Opérateur	Signification
binaire	&	Et
binaire		Ou
binaire	^	Ou exclusif
binaire	<<	Décalage à gauche
binaire	>>	Décalage à droite
unaire	~	Complément à 1

n	0000010101101110	056E	1390
p	0000001110110011	03B3	947
n & p	0000000100100010	0122	209
n p	0000011111111111	07FF	2047
n ^ p	0000011011011101	06DD	1757
~ n	1111101010010001	FA91	-1391
n << 2	0001010110111000		
n >> 3	0000000010101101		

Exemple :

```
main( )  
{ int n ;  
  printf(" donnez un entier") ;  
  scanf ("%d",&n) ;  
  if (n & 0x0001 == 1) printf("          ") ;  
  else printf("          ") ; }
```

Opérateurs d'affectation élargie:

$i = i + k ;$	$i \text{ + = } k ;$
$a = a * b ;$	$a \text{ * = } b ;$
$n = n \ll 3 ;$	$n \text{ \ll = } 3 ;$
Liste des Opérateurs : += ; -= ; *= ; /= ; %= ; = ; ; ^= ; &= ; <<= ; >>=	

Opérateur conditionnel ? :

Considérons l'expression :

```
if ( a > b )      max = a ;  
    else        max = b ;
```

sera remplacé par :

$\text{max} = \text{a} > \text{b} \text{ ? } \text{a} : \text{b} ;$

priorité	
$z = (x = y) ? a : b ;$ affecte y à x , et si cette valeur est non nulle, affecter a à z, sinon affecter b à z.	$z = x = y ? a : b ;$ va être évaluée $z = x = (y ? a : b) ;$ si $y \neq 0$ affecter a à x et z, sinon affecter b à x et z.

Opérateur de conversion "cast" :

Pour deux entiers n et p avec $n = 10 ; p = 3 ;$

(double) (n/p) vaut 3

(double) n/p vaut 3.33

Opérateur Séquentiel :

`i++ , j=i+k ;` est équivalente à : `i++ ; j=i+k ;`

Opérateur sizeof : fournit la taille d'une variable, si `z` est de type double ;

`sizeof(z)` vaudra 8.

`sizeof(int)` vaut 2.

`sizeof(long int)` vaut 4.

Exemple : `int n = 5 , p = 9 ; int q ; float x ;`

<code>q = n < p ;</code>	1
<code>q = n == p ;</code>	0
<code>q = p % n + p > n ;</code>	$5 = 4 + 1$
<code>x = p / n ;</code>	1
<code>x = (float) p / n ;</code>	1.8
<code>x = (p + 0.5) / n ;</code>	1.9
<code>x = (int)(p + 0.5) / n ;</code>	1
<code>q = n * (p > n ? n : p)</code>	25
<code>q = n * (p < n ? n : p)</code>	45

12- Priorité des Opérateurs :

Catégorie	Opérateurs	Associativité
Référence	() []	→
Unaire	+ - ++ -- ! ~ * & cast sizeof	←
Arithmétique	* / %	→
Arithmétique	+ -	→
Décalage	<< >>	→
Relationnels	< <= > >=	→
Relationnels	== !=	→
manip-bit	&	→
manip-bit	^	→
manip-bit	 	→
Logique	&&	→
Logique	 	→
Conditionnel	? :	→
Affectation	= ; += ; -= ; *= ; /= ; %= ; &= ; = ; ^= ; <<= ; >>=	←
Séquentiel	,	→

Instructions de contrôle :

Instruction IF :

if (expression) instruction1 ; else instruction2

if (a<=b) printf(" inférieur ");

else printf(" supérieur ");

Instruction switch :

switch (expression)

{

case const1 :

[suite instructions1]

case const2 :

[suite instructions2]

case const_n :

[suite instructions_n]

default : suite instructions

}

```
#include <stdio.h>
```

```
main() {
```

```
int n;
```

```
printf(" donnez un entier ");
```

```
scanf("%d",&n);
```

```
switch (n)
```

```
{
```

```
case 0 : printf("nul \n") ; break ;
```

```
case 1 : printf("un \n") ; break ;
```

```
default : printf(" grand \n") ;
```

```
}
```

```
printf(" au revoir "); }
```

Instruction do ...while

```
do instructions;
while (expression) ;
main() {
    int n;
do {
printf(" donnez un entier >0  ");
scanf("%d",&n);
printf(" vous avez fourni : %d\\ n  ", n);
} while (n<=0) ;
printf(" réponse correcte  ");
}
```

Instruction while :

```
while (expression) instructions;

main() {
    int n , som = 0 ;
while (som<100)
{printf("donnez un nombre :");
scanf("%d",&n);
som+=n ;}
printf("Somme obtenue : %d ",som);
}
```

Instruction for :

```
fact=1;
for(i=1;i<=n;i++)
fact=fact*i;
printf(" Factoriel %d ",fact);
```

```
for([exp1] ;[exp2] ;[exp3])
instructions
```



```
exp1;
while(exp2)
{ instructions; exp3; }
```

Instructions de branchement inconditionnel:

-1 Instruction break : interrompt déroulement d'une boucle

<pre>main() { int i ; for(i=1;i<=10;i++) { printf("début tour %d \n",i); printf("bonjour \n"); if (i== 3) break; printf("fin tour %d\n",i); } printf("après boucle"); }</pre>	<pre>début tour 1 bonjour fin tour 1 début tour 2 bonjour fin tour2 début tour 3 bonjour après boucle</pre>
---	---

-2 Instruction continue: permet de passer au tour suivant de la boucle

<pre>main() { int i ; for(i=1;i<=4 ;i++) { printf("tour %d \n",i); if (i < 3) continue; printf(" bonjour \n"); } }</pre>	<pre>tour 1 tour 2 tour 3 bonjour tour 4 bonjour</pre>
---	--

<pre> main() { int n ; do { printf("donnez un nb>0 : "); scanf("%d",&n); if (n < 0) { printf(" svp >0 : \n "); continue; } printf(" son carré est : %d \n ",n*n);} while (n) ; }</pre>	<p>donnez un nb >0 : 2 son carré est : 4 donnez un nb > 0 : -5 svp > 0 donnez un nb >0 : 0 son carré est : 0</p>
--	---

-3 Instruction goto : permet le branchement en un emplacement quelconque

<pre> main() { int i ; for(i=1;i<=10;i++) { printf(" tour %d \n",i); printf("bonjour \n"); if (i= = 3) goto sortie ; printf("fin tour %d\n",i); } sortie : printf("après boucle"); }</pre>	<p>tour 1 bonjour fin tour 1 tour 2 bonjour fin tour2 tour 3 bonjour après boucle</p>
--	---

VII- Les Fonctions :

Type de variables :

1. variable globale : accessible depuis n'importe quel endroit du source et même d'autres fichiers.
2. globale cachée(static) : accessible depuis n'importe quel endroit du source, mais pas d'autres fichiers.
3. locale à une fonction (main).
4. locale à un bloc.

Exemple1 : fonction sans argument et sans valeur :

```
main()
{
    optimist();
}

optimist()
{
    printf(" il fait beau ");
}
```

Exemple2 : fonction avec argument et sans valeur :

```
main()
{
    int a = 10, b = 20 ;
    ecris(a) ;
    ecris(b);
    ecris (a+b);
}

ecris(int n)
{
    printf(" valeur :%d \n ",n);
}
```


Exemple3 : fonction fournissant un résultat :

```
main()
{ double a,b,c,d,x,y ;
double som(double, double) ; /*prototype*/
a = 1. ; b = 2. ; c = 3. ; d = 4. ;
x = som (a,b) + 5 ;
printf(" x = %e \n ",x);
y = 3 * som(c,d);
printf(" y = %e \n ",y);
}
```

```
double som(double u, double v)
{
    double s ;
    s = u + v ;
    return s;
}
x = 8.000000e+000
y = 2.100000e+001
```

- notion de prototype : déclaration entête d'une fonction.

double som(double, double) ;

- Le type **void** : sert à spécifier qu'une fonction ne retourne pas de résultat, et améliore la lisibilité.

ex : void optimist () ;

void ecris(int) ;

même si elle retourne une valeur, C ne fait pas de control.

Arguments transmis par valeurs :

```

echange(int a, int b)
{ int c ;
printf(" début échange : %d%d \n",a,b);
c = a ;
a = b ;
b = c ;
printf(" fin échange : %d%d \n ",a,b);
}

```

```

main()
{ int n = 10 , p = 20 ;
printf(" avant appel : %d%d \n ",n,p);
echange(n,p) ;
printf(" après appel : %d%d \n ",n,p);
}
avant appel : 10 20
début échange : 10 20
fin échange : 20 10
après appel : 10 20

```

Arguments transmis par adresses :

```

main()
{ void p(int x, int *y) ;
int A = 0 ,B = 0;
p(A,&B) ;
printf("%d%d \n ",A,B); }
void p(int x, int *y)
{ x= x + 1 ;
  *y = *y + 1 ;
printf("%d%d \n ",x,*y); }

```

Affichage de l'exécution :

1 1
0 1

A	B	Appel →	x	*y
0	0	retour	0	0
0	1	←	1	0
			1	1

VIII Tableaux et Pointeurs :

1-Tableau à une dimension :

```
main() { int i, som, nbm;  
        float moy;  
        int t[20];  
        for (i=0;i<20;i++)  
        { printf("donnez la note n° %d :", i+1);  
          scanf("%d",&t[i]); }  
        for (i=0;som=0;i<20;i++)  
        som+=t[i];  
        moy=som/20;  
        printf("moyenne : %f ",moy); }
```

//opérations possibles

```
t[2] = 5 ;  
t[3]++;  
--t[i] ;  
t[q*p-2+j%i] = var ;
```

-Initialisation de tableau à une dimension :

```
int tab[5] = {10,20,5,0,3} ;
```

```
int tab[5] = {10,20} ; /* valeurs manquantes = 0 */
```

```
int tab[5] = {,,5,,3} ; // dépend du compilateur
```

```
int tab[ ] = {10,20,5,0,3} ;
```

1-Tableau à plusieurs dimensions :

```
int t [5][3] ;           [0]
son                     t[0][1]
arrangement             [2]
mémoire :               [0]
                        t[1][1]
                        [2]
                        ...
                        [0]
                        t[4][1]
                        [2]
```

Dépassement d'indice :

$t[0][5] \rightarrow t[1][2]$

Initialisation de tableau :

```
int tab[3][4]={ {1,2,3,4},{5,6,7,8},
                {9,10,11,12}};
```

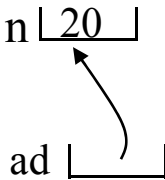
```
int tab[3][4]={1,2,3,4,5,6,7,8,9,10,11,12} ;
```

Notion de pointeurs (opérateurs * et &) :

`int *ad ; /* réserve comme étant pointeur sur un entier */`

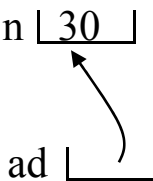
`int n =20 ;` n | 20 |

`ad = &n;`



The diagram shows a memory box for variable 'n' containing the value 20. Below it, a memory box for variable 'ad' is shown with an arrow pointing from its center to the center of the 'n' box, indicating that 'ad' stores the address of 'n'.

`*ad = 30 ;`



The diagram shows a memory box for variable 'n' containing the value 30. Below it, a memory box for variable 'ad' is shown with an arrow pointing from its center to the center of the 'n' box, indicating that 'ad' stores the address of 'n'.

Exemple : `int *ad1,*ad2 ; int n = 10 , p = 20 ;`

`ad1 = &n ;`

`ad2 = &p ;`

`*ad1 = *ad2 + 2 ; \Leftrightarrow n = p + 2 ;`

`*ad1 += 3 ; \Leftrightarrow n = n + 3 ;`

`(*ad1)++ ; \Leftrightarrow n++ ;`

si ad est un pointeur : ad ou *ad ont un sens (lvalue) , par contre &ad n'a pas de sens (&2 adresse fixe).

Utilisation des pointeurs en argument de fonction :

```
main( )
{ int a = 10, b = 20 ;
printf(" avant appel %d %d \n ",a,b);
echange(&a,&b) ;
printf(" après appel %d %d \n ",a,b);
}
```

```
echange(int *ad1, int *ad2)
{ int x ;
  x = *ad1 ;
  *ad1 = *ad2 ;
  *ad2 = x ; }
avant appel  10  20
après appel  20  10
```

pointeurs dans les tableaux :

```
int t[10] ;
alors les notations suivantes sont équivalentes :
t   ⇔  &t[0]
t+1 ⇔  &t[1]
t+i ⇔  &t[i]
t[i] ⇔  *(t+i)

pour int t[3][4];
t[0] ⇔  t[0][0];
t[1] ⇔  t[1][0];
```

fonction qui calcule la somme des éléments
d'un tableau de taille quelconque

```
int som(int t[ ], int nb) ;
{ int i, s = 0;
  for (i = 0 ; i < nb ; i++)
    s += t[i];
  return s ; }
```

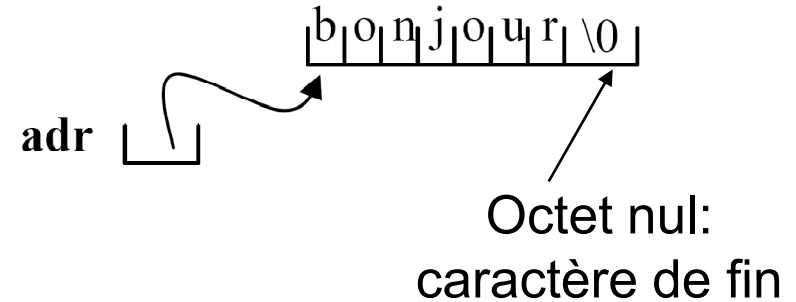
appel de cette fonction se fait :

```
int t1[30], t2[15] ;
s1 = som (t1,30) ;
s2 = som (t2,15) ;
```

XI. LES CHAINES DE CARACTERES(C ne dispose de string) :

1. Constantes chaîne de caractères :

```
main ()  
{   char *adr ;  
  adr = "bonjour" ;  
    do  printf ("%c ",*adr) ;  
    while (*adr++) ;  }
```



2. Initialisation tableaux caractères :

```
char  ch[20] ;  
ch= "bonjour" ; n'est pas correct  
Par contre : char ch[20] ={'b','o','h','j','o','4','5','\0'}
```

3. Tableaux de pointeurs sur chaînes :

```
main () { int i ;  
char *jour[7]= { "lundi","mardi",...."dimanche"} ;  
printf ("donnez un entier entre 1 et 7") ;  
scanf ("%d",&i) ;  
printf ("le jour numéro %d de la semaine est %s",i,jour [i-1]) ; }  
donnez un entier entre 1 et 7 :3  
le jour numero 3 de la semaine est mercredi
```

4. Fonctions gets, puts et le code format %s :

```
main( ) {  
    char nom[20], prénom [20], ville[25] ;  
    printf ("quelle est votre ville : ") ;  
    gets(ville) ;  
    printf ("donnez votre nom et prénom : ") ;  
    scanf ("%s%s", nom ,prénom) ;  
    printf ("bonjour cher %s%s qui habite à ", prénom, nom) ;  
    puts(ville) ;  
}
```

Quelle est votre ville: paris

Donnez votre nom et prénom: dupont yves

Bonjour cher yves dupont qui habite paris

- **Fonctions** **strlen**(chaine) retourne la longueur(lenght) d'une chaîne.

Remarques :

- ❑ printf,scanf:permettent de lire et afficher plusieurs infos de type quelconques.
- ❑ gets, puts : ne traitent qu'une chaîne à la fois.
- ❑ %s **délimiteurs** classiques:espace, tabulation et fin de ligne, càd, scanf interdit lecture d'une chaîne contenant un espace, avec gets : seul fin ligne délimiteur.
- ❑ Les identificateurs de scanf et gets ne doivent être précédés de l'opérateur & (puisque'ils représentent déjà des adresses).
- ❑ Puts réalise un changement de ligne à la fin de l'affichage ce qui n'est pas le cas de printf (%s).

5. Fonctions de concaténation de chaînes : (prototypes dans string.h)

strcat(but, source) : recopie la seconde à la suite de la première chaîne.

```
main( ) {  
    char ch1[50]= "bonjour" ;  
    char * ch2= "monsieur" ;  
    printf ("avant : %s\n",ch1) ;  
    strcat (ch1,ch2) ;  
    printf ("après :%s",ch1) ; }
```

avant : bonjour
après : bonjourmonsieur

strncat(but,source,lgmax) contrôle sur nombre de caractères qui seront concaténés.

```
char ch1[50]= "bonjour" ;  
char *ch2="monsieur" ;  
printf ("avant : %s\n",ch1) ;  
    strncat (ch1,ch2,6) ;  
printf ("après : %s",ch1) ;
```

avant : bonjour
après : bonjourmonsie

6. Fonctions de comparaisons : (prototype dans string. h)

strcmp (chaine1, chaine2) : Compare chaine1 et chaine2 et retourne une valeur entière :

- Positive si $\text{chaine1} > \text{chaine2}$
- Nulle si $\text{chaine1} = \text{chaine2}$ (même suite de caractères)
- Négative si $\text{chaine1} < \text{chaine2}$

Exemple :

strcmp ("bonjour","monsieur") negative

strcmp ("paris2", "paris10") positive

strncmp (chaine1, chaine2, lgmax) : limite la comparaison au nombre de caractère indiqué par l'entier lgmax

Exemple :

strncmp("bonjour","bon",4) positive

strncmp("bonjour","bon",2) nulle

7. Fonctions de copie de chaîne : (prototype dans string.h)

strcpy(destin,source) : Recopie source dans destin, il est nécessaire que la taille soit suffisante pour accueillir la chaîne à recopier.

strncpy (destin, source, lmax) : Limite la recopie au nombre de caractère précisé :

si $\text{lg}(\text{source}) < \text{lmax}$ alors caractère `'\0'` sera copié sinon le caractère `'\0'` ne sera pas copié.

Exemple :

```
main() {  
    char ch1[20]="xxxxxxxxxxxxxxxxxxxxxx";  
    char ch2[20];  
    printf("donnez un mot :");  
    gets(ch2);  
    strncpy(ch1,ch2,6);  
    printf(" %s",ch1);  
}
```

```
donnez un mot : bon  
bon  
donnez un mot :bonjour  
bonjouxxxxxxxxxxxxxx
```

8. fonction de rechercher dans une chaîne (string.h):

strchr (chaîne, caractère) : recherche la première position du caractère dans la chaîne retourne l'adresse en cas succès ou pointeur nul sinon.

strrchr (chaîne, caractère) : même chose mais en explorant la chaîne à partir de la fin.

strstr (chaîne,sous-chaîne) : recherche première occurrence de sous chaîne.

strpbrk (ch1,ch2) : recherche dans ch1 la première occurrence d'un char quelconque de ch2.

9. fonctions de conversion : (prototype dans stdlib.h) :

Chaîne \Rightarrow valeur numérique : il existe trois fonctions permettent de convertir une chaîne en une valeur numérique (int, long,double)

atoi (chaîne): fournit résultat de type **int**.

atol (chaîne): fournit résultat de type **long**.

atof (chaîne): fournit résultat de type **double**.

Valeur numérique \Rightarrow chaîne :

itoa (entier, chaîne, base) pour entier **int**

ltoa (entier, chaîne, base) pour entier **long**

ltoa (entier, chaîne,base) pour entier **unsigned long**

Exemple :

```
#include<stdio.h>
#include<string.h>
#define c 'e'
#define sch "re"
#define voy "aeiou"
main( ) {
char mot[40];

printf("donnez un mot : ");
gets(mot);

printf(" premiere occurrence de %c en %s \n",c,strchr(mot,c));
printf(" derniere occurrence de %c en %s \n",c,strrchr(mot,c));
printf(" premiere occurrence de %s en %s
\n",sch,strstr(mot,sch));
printf(" premiere occurrence de l'une des lettres %s en
%s\n",voy,strpbrk(mot,voy)); }
```

```
donnez un mot : correspondances
premiere occurrence de e en espondances
derniere occurrence de e en es
premiere occurrence de re en respondances
premiere occurrence de l'une des lettres aeiou en orrespondances
```

```
donnez un mot : bonjour
premiere occurrence de e en (null)
derniere occurrence de e en (null)
premiere occurrence de re en (null)
premiere occurrence de l'une des lettres aeiou en onjour
```

Exemple 1:

```
#include <stdlib.h>
main() { char ch[40] ;
int n;
do { printf ("donnez une chaine: ") ;
gets (ch);
printf ("val int : %d\n",n=atoi(ch));
printf("val double :%e\n",atof(ch)); }
while (n); }
```

```
donnez une chaine :123
val int 123
val double 1.230000e+002
donnez une chaine : -123.45
val int -123
val double -1.234500e+002
donnez une chaine : bof
val int 0
val double 0.000000e+000
```

Exemple 2 :

```
#include <stdlib.h>
main() {
char ch[50];
int n,b;
do {
printf("donnez un nbre et une base: ");
scanf("%d%d",&n,&b) ;
printf("%s\n" , itoa(n,ch,b)); }
while (n); }
```

```
donnez un nbre et base : -123 10
-123
donnez un nbre et base : 200 16
C8
donnez un nbre et base : 35 36
z
donnez un nbre et base : 0 0
```

XII. LES STRUCTURES :

1. structures (déclarations) :

```
struct enreg  
{ int numero ;  
    int qte ;  
    float prix ;  
};
```

2. déclaration de variables :

```
struct enreg art1,art2 ;  
ou struct enreg {  
    int numero ;  
    int qte ;  
    float prix ;  
} art1,art2 ;
```

3. utilisation des champs d'une structure :

```
art1.numero=15 ;  
printf ("%e",art1.prix) ;  
scanf("%e",&art2.prix) ;  
art1.numero++ ;          /* . plus prioritaire que ++ et autre */  
art1=art2 ;
```

struct enreg art1= { 15,285,253.75} initialisation

Initialisation statique sont par défaut à zéro, automatique ne sont pas initialisées donc aléatoire.

4. déclaration de types synonymes: **typedef**

Déclaration: **typedef** **int** entier ;
 int n,p ; \Leftrightarrow entier n,p;
 typedef int *ptr ;
 int *p1,*p2 ; \Leftrightarrow ptr p1,p2;
 typedef int vecteur [3]:
 int v[3],w[3]; \Leftrightarrow vecteur v,w;

Application aux structures :

```
struct enreg {  
    int numéro ;  
    int qte ;  
    float prix ; }  
  
typedef struct enreg s_enreg ;  
s_enreg art1,art2 ; ou bien  
  
typedef struct {  
    int numero;  
    int qte ;  
    float prix ; } s_enreg ; s_enreg art1,art2 ;
```

5. Exemple de structures :

```
struct personne { char nom[30] ; char prenom[20] ;  
                  float heures[31] ; } employe ,courant ;
```

employe.heure[4] : 5eme élément tableau heure

employe.nom [0] : premier caractère champ nom

&courant.heure[4] : @ 5eme élément tableau heure

courant.nom : @ tableau nom

```
struct point { char nom ;  
               int x,y ; } ;
```

```
struct point  coubre[50] ;
```

```
courbe[i].nom=...;
```

```
courbe[4].x=...;
```

```
struct date { int jour,mois,année ;} ;
```

```
struct personne { char nom [30] ;char prenom [20] ;
```

```
                struct date. date_embauche ;
```

```
                struct date. date_poste ; } employe,courrant ;
```

```
employe.date_embauche.année=1998 ;
```

```
courant.date_embauche= employe.date_poste ;
```

6. structure transmise en argument de fonction :

transmise par valeur :

```
struct enreg { int a ; float b ;};
main() {
    struct enreg x;
    void fct (struct enreg y);
    x.a=1 ;x.b=12.5 ;
    printf ("\n avant appel :%d%e", x.a, x.b) ;
    fct(x) ;
    printf ("\n apres appel : %d %e", x.a, x.b) ; }
void fct (struct enreg s)
{ s.a=0 ; s.b =1 ;
  printf ("\n dans fct :%d%e",s.a, s.b) ; }
```

```
avant appel : 1    1.25000e+01
dans fct :      0    1.00000e+00
après appel : 1    1.25000e+00
```

transmise par adresse :

```
struct enreg { int a ; floatb ;} ;
main ()
    struct enreg *;
    void fct (struct enreg *)
    x.a=1 ; x.b=12.5;
    printf ("\n avant appel : %d%e", x.a, x.b) ;
    fct (&x) ;
    printf ("\n apres appel: %d%e" , x.a, x.b) ;}
void fct (struct enreg *s)
{ s->a =0; s->b=1;
  printf ("\n dans fct : %d%e", s->a, s->b);}
```

```
avant appel : 1    1.25000e+01
dans fct :      0    1.00000e00
apres appel : 0    1.00000e+00
```

XIII. La gestion dynamique :

Outils de base de la gestion dynamique: **malloc** & **free** (stdlib.h et alloc.h):

Exemple :

```
# include <stdlib.h>
```

```
.....
```

```
char * adr;
```

```
.....
```

```
adr = malloc(50);
```

```
.....
```

```
for (i=0;i<50;i++)
```

```
*(adr+i) = 'x';
```

```
float *pi ;
```

```
pi=(float*)malloc (size of (foat));
```

```
*pi= 3.14;
```

```
.....
```

```
free (pi);
```

Rq: ne plus appeler pi après free

–malloc (50) alloue un emplacement de 50 octets

–la fct malloc() peut échouer , s'il n y plus de mémoire disponible , elle renvoie la valeur 0 qui se note null.

–float * pi =(float*)malloc(sizeof(float));

```
if (pi==NULL) { printf ("échec de malloc : plus de mémoire !\n") ;
```

```
else { *pi=3.14 ; .....;free(p)} ;
```

pointeur & tableaux : on peut créer un tableau avec `malloc()` :

```
float *t = (float*)malloc(10*sizeof (float)) ;  
if (t!=null) { int i;  
for (i=0; i<10; i++);  
t[i]=i;  
free(t);  
.....  
float * p =t; /* pointeur sur élément du tableau */  
int i ;  
for ( i=0, i<10,i++)  
{ *p=i;  
  p=p+1;}  
free(t) ;}  
.....  
char *pcar.  
pcar=(char*)malloc(10*sizeof(char));
```

tableau de structures:

```
typedef struct { char nom[20]; char prenom [20]; } fiche;  
fiche  *pf;  
pf= (fiche*) malloc (15* sizeof (fiche));  
printf(" nom :%s" , pf[5].nom) ;  
free(pf) ;
```

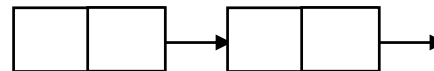
.....

```
struct point { int x; int y;};  
struct point  pi;  
p.x=3 ; p.y=4 ;  
struct point  *q;  
q->x=3; q->y=4;
```

liste chaînée:

Déclaration:

```
struct element { int num;  
                 float x;  
                 float y;  
                 struct element *suivant ; }
```



Exemple 1:

```
struct liste {  
    char chaine[16];  
    struct liste * psuivant ;};  
  
struct liste *nouveau ;  
struct liste *tete ;  
tete =NULL ;
```

ajout d'un élément :

```
nouveau=(liste*)malloc( sizeof (struct liste)) ;  
nouveau -> psuivant = tete ;  
tete= nouveau ;
```

Exemple 2 :

```
typedef struct cellule { int elt ;  
                        struct cellule * suiv ;  
                    } Liste ;  
  
Liste *l=NULL ; /* initialisation liste chaînée */
```

fonction inseretete : insère un nouvel élément en tête de la liste l et retourne un pointeur sur la liste.

```
Liste * insereentete (int x, Liste*l)
```

```
{ Liste *c ;
```

```
c= (liste *)malloc(sizeof(liste)) ;
```

```
if (c!= NULL) { c->elt= x; /* assignation de x */
```

```
                c->suiv=l; /* assignation du suivant */
```

```
                l=c ; /* chainage*/
```

```
                } return(l) ;}
```

fonction appartient: qui renvoie 1 si la valeur x appartient à la liste, 0 sinon.

```
int appartient (int x, liste*l) ;
```

```
{ if (l== NULL) return 0 ;
```

```
  else if (l->elt==x) return 1;
```

```
        else return(appartient(x,l->suiv)); }
```