

# Programmation en C

Les pointeurs, les structures et les fonctions

John Samuel  
CPE Lyon

Année: 2018-2019

Courriel: john(dot)samuel(at)cpe(dot)fr



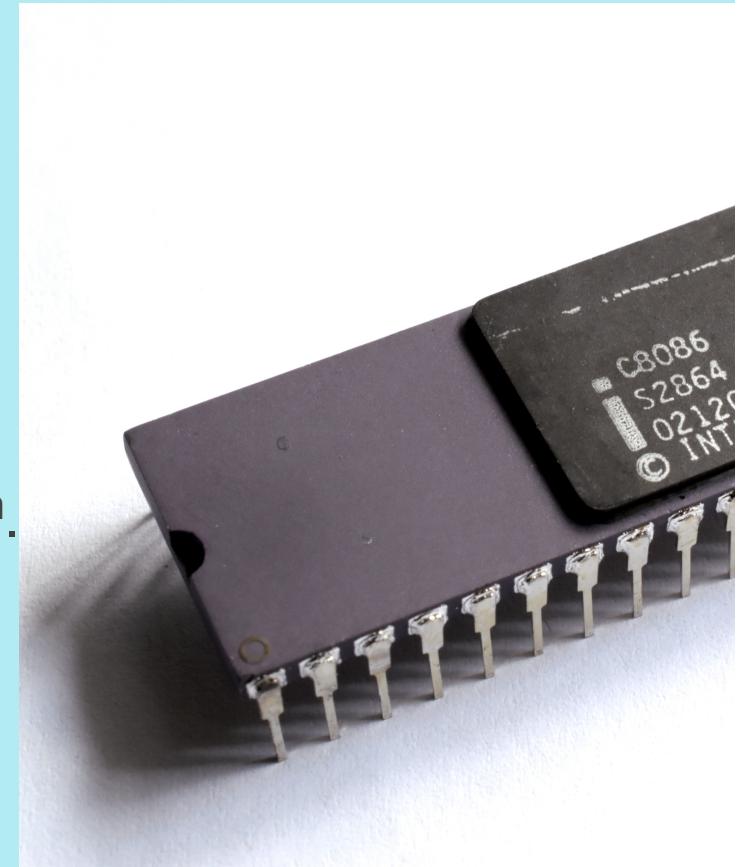
## Objectifs

- Les pointeurs
- Les structures
- Les unions
- Introduction aux fonctions

## Architecture

- 16 bits: E.g., Intel C8086
- 32 bits: E.g., Intel 80386, Intel Pentium II/III, AMD Athlon
- 64 bits: E.g., AMD Athlon 64, Intel Core 2

Une architecture n-bit peut adresser une mémoire de taille  $2^n$ .



# Microprocesseur

## Architecture

- 16 bits: 65,536 adresses (octets)
- 32 bits: 4 Gio ( $2^{32}$ :4,294,967,295)adresses (octets)
- 64 bits: 16 Eio ( $2^{64}$ ) adresses (octets)

0	0	0	0	0	1	0	0	0
1	0	0	0	1	1	0	0	1
2	0	0	1	0	1	0	1	0
3	0	0	1	1	1	0	1	1
					...			
					...			
					...			
$2^n - 4$	0	1	0	0	1	1	0	0
$2^n - 3$	0	1	0	1	1	1	0	1
$2^n - 2$	0	1	1	0	1	1	1	0
$2^n - 1$	0	1	1	1	1	1	1	1

# Mémoire vive

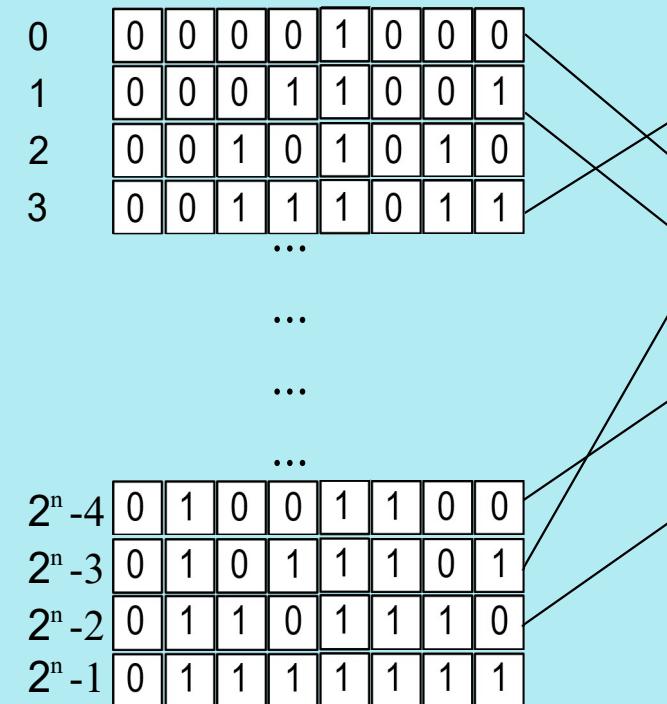
La capacité de la **mémoire vive** d'un ordinateur est de 4GO, 8GO, 16GO etc.



# Mémoire Virtuelle

## Mémoire virtuelle

- Mémoire virtuelle permet l'utilisation de la mémoire de masse comme extension de la mémoire vive.
- **Adresses:** Les adresses virtuelles et les adresses physiques
  - Les adresses virtuelles sont utilisées par un programme
  - Les adresses physiques sont utilisées par une puce mémoire
- L'**unité de gestion mémoire** traduit des adresses virtuelles en adresses physiques.



La conversion des adresses virtuelles

# Bit de poids fort et faible

Dans une représentation binaire donnée,

- **Le bit de poids fort (MSB):** le bit qui représente la plus grande puissance de 2
- **Le bit de poids faible (LSB):** le bit qui représente la plus petite puissance de 2

$2^7$

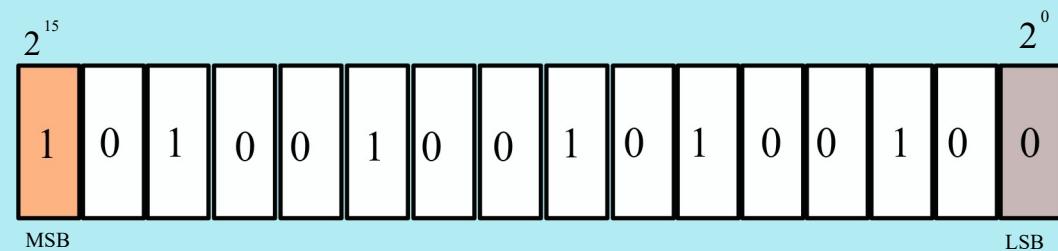
1

MSB

MSB e

# Endianness (Boutisme)

Dans une représentation binaire donnée (16 bit),



MSB et LSB (16 bit)

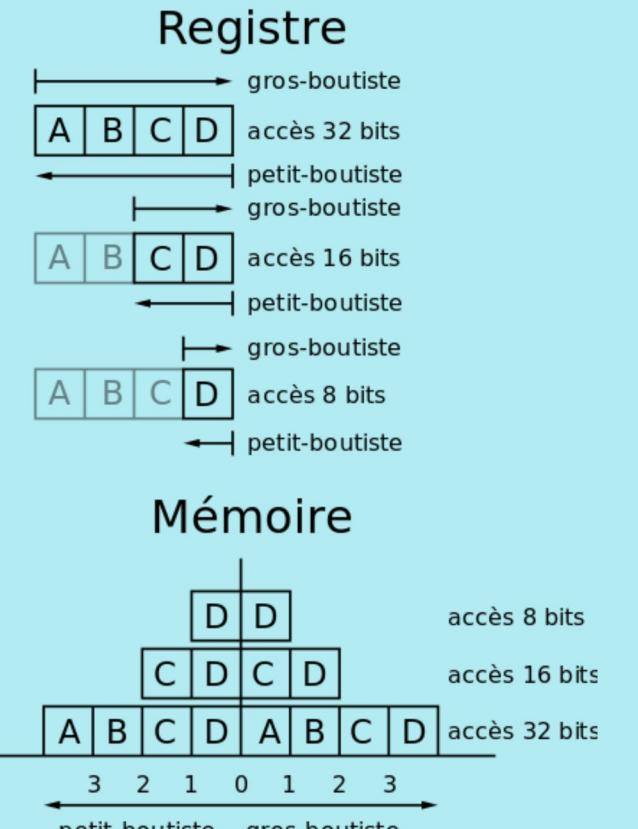
# Endianness (Boutisme)

## Endianness (Boutisme)

est l'ordre dans lequel les octets sont organisés en mémoire.

- Big-endian (grand-boutiste)
- Little-endian (petit-boutiste)

**Remarque:** On va utiliser petit-boutiste dans notre cours



Boutisme: grand-boutiste et p

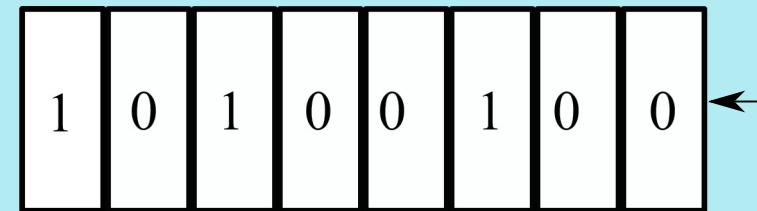
# Pointeurs

```
char c = 'a';
char *my_char_addr = &c;
```

char c = 0xa4

Remarque: my\_char\_addr = 0xab71

&c: 0xab71



une variable char

# Pointeurs

```
short s = 0xa478;  
short *my_short_addr = &s;
```

short s = 0xa478;

Remarque: my\_short\_addr = 0xab71

&s: 0xab71

0xab72

0	1	1	1	1	0	0	0
1	0	1	0	0	1	0	0

une variable short

# Pointeurs

```
int i = 0xa47865ff;  
int *my_int_addr = &i;
```

Remarque: my\_int\_addr = 0xab71

int i = 0xa47865ff;

&i: 0xab71

0xab72

0xab73

0xab74

1	1	1	1	1	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	1	0	0	0
1	0	1	0	0	1	0	0

une variable int

# Pointeurs

```
long int li = 0xa47865ff;  
long int *my_long_int_addr = &li;
```

**Remarque:** my\_long\_int\_addr = 0xab71

long int li = 0xa47865ff;

&li: 0xab71	1	1	1	1	1	1	1	1
0xab72	0	1	1	0	0	1	0	1
0xab73	0	1	1	1	1	0	0	0
0xab74	1	0	1	0	0	1	0	0
0xab75	0	0	0	0	0	0	0	0
0xab76	0	0	0	0	0	0	0	0
0xab77	0	0	0	0	0	0	0	0
0xab78	0	0	0	0	0	0	0	0

une variable long int

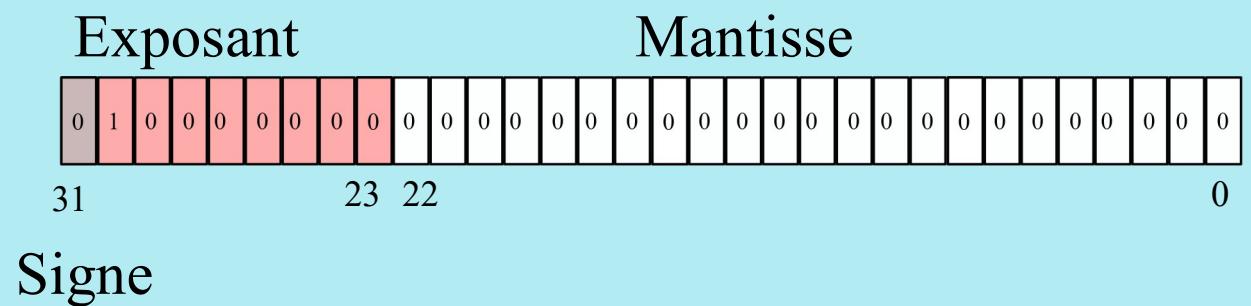
13 < | >

# Pointeurs: les nombres en flottant

## float (32-bit)

- signe: bit 31
- exposant: bits 23-30
- mantisse: bits 0-22

Remarque: IEEE 754



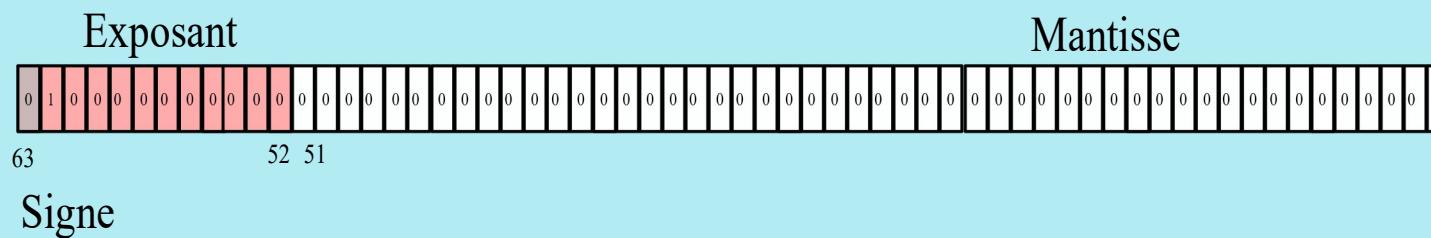
une variable float

# Pointeurs: les nombres en flottant

double (64-bit)

- signe: bit 63
- exposant: bits 52-62
- mantisse: bits 0-51

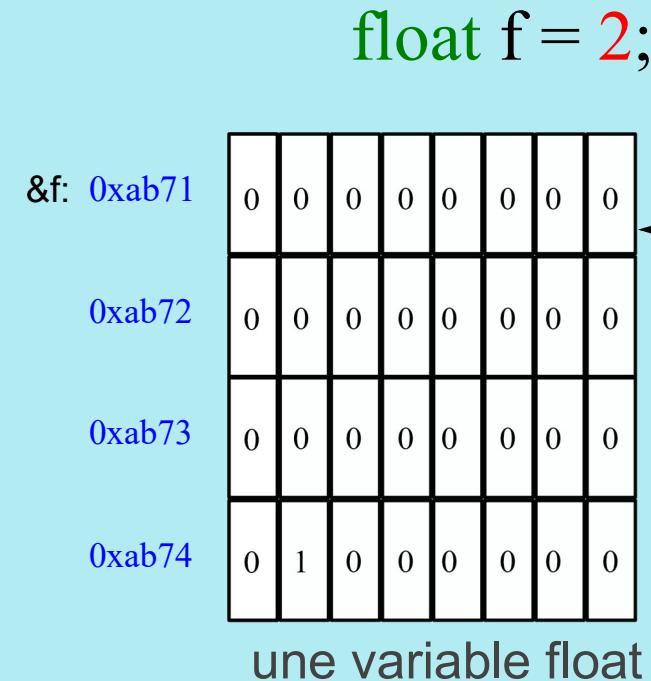
Remarque: IEEE 754



# Pointeurs

```
float f = 2;  
float *my_float_addr = &f;
```

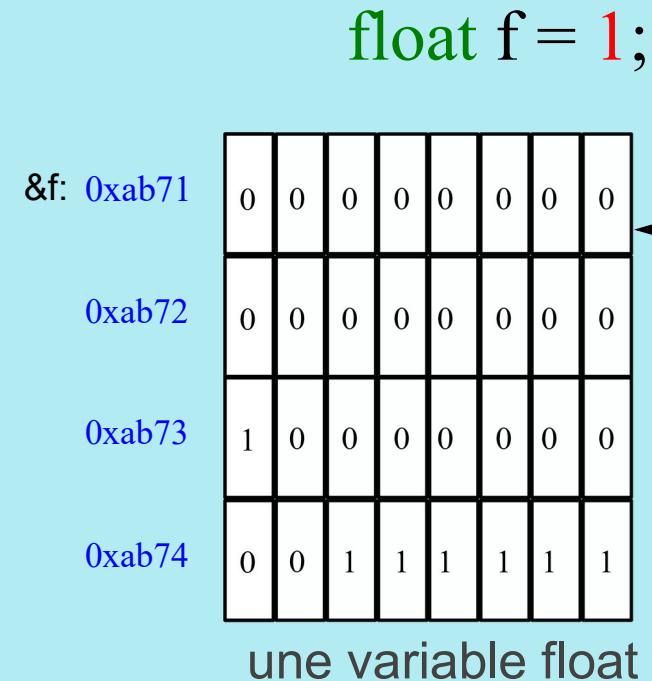
Remarque: my\_float\_addr = 0xab71



# Pointeurs

```
float f = 1;  
float *my_float_addr = &f;
```

Remarque: my\_float\_addr = 0xab71



# Pointeurs

```
double d = 2;  
double *my_double_addr = &d;
```

**Remarque:** my\_double\_addr = 0xab71

double d = 2;

&d: 0xab71

0xab72

0xab73

0xab74

0xab75

0xab76

0xab77

0xab78

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0

# Pointeurs

```
double d = 2;  
double *my_double_addr = &d;
```

**Remarque:** my\_double\_addr = 0xab71

double d = 1;

&d: 0xab71

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
1	1	1	1	0	0	0	0
0	0	1	1	1	1	1	1

# printf: Afficher à l'écran

Type	Code de conversion
pointeur	%p

```
printf("char : %p\n", my_char_addr);
printf("short : %p\n", my_short_addr);
printf("int : %p\n", my_int_addr);
printf("long int : %p\n", my_long_int_addr);
printf("float : %p\n", my_float_addr);
printf("double : %p\n", my_double_addr);
```

# L'opérateur de déréférenciation

**L'opérateur de déréférenciation \*** désigne l'objet pointé par un pointeur

```
char c = 'a';
char *my_char_addr = &c;
printf ("%c", *my_char_addr);
```

# L'opérateur de déréférenciation

L'opérateur \* est utilisé pour manipuler un objet pointé

## Exemple

```
char c = 'a';
char *my_char_addr = &c;
c = 'b';
*my_char_addr = 'c';
printf ("%c", c);
```

## Résultat

```
c
//le caractère pointé par my_char_addr prend la valeur 'c'
```

# L'opérateur de déréférenciation

L'opérateur \* est utilisé pour manipuler un objet pointé

## Exemple

```
int i = 0x20;
int *my_int_addr = &i;
*my_int_addr = 1;
*my_int_addr = *my_int_addr + 1; //i = 0x2
i = i + 3; //i = 0x5
printf ("%x", *my_int_addr);
```

## Résultat

5

# Les pointeurs et les tableaux

Un **tableau** est un ensemble d'éléments homogènes.

```
char a[8] = {0xff, 0x65, 0x78,  
             0xa4, 0x0, 0x0, 0x0, 0x0};  
char *ptr = &a[0];
```

ptr → 0xab71

	1	1	1	1	1	1	1	1
0xab72	0	1	1	0	0	1	0	1
0xab73	0	1	1	1	1	0	0	0
0xab74	1	0	1	0	0	1	0	0
0xab75	0	0	0	0	0	0	0	0
0xab76	0	0	0	0	0	0	0	0
0xab77	0	0	0	0	0	0	0	0
0xab78	0	0	0	0	0	0	0	0

# Pointers for continuous data blocks

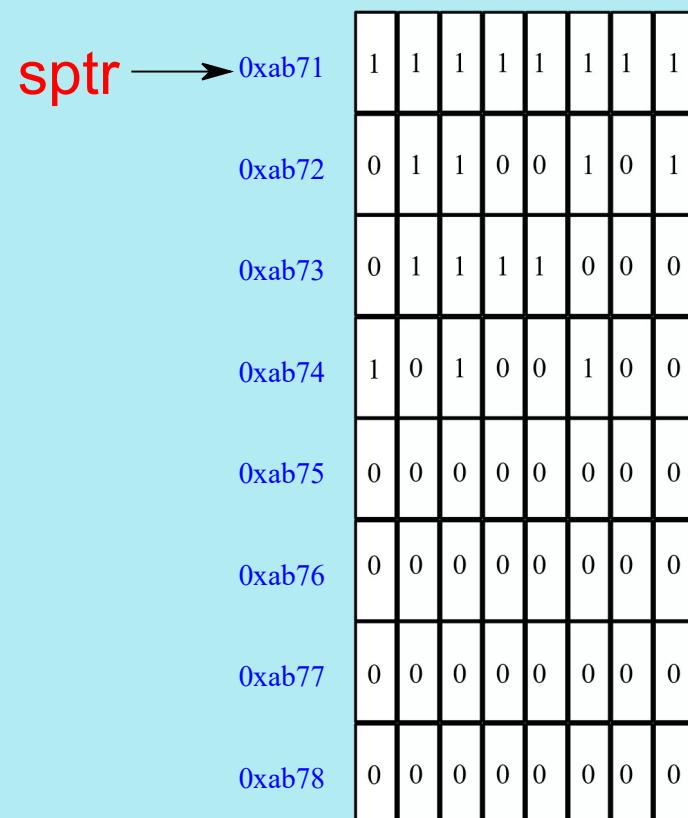
```
char a[8] = {0xff, 0x65, 0x78,  
             0xa4, 0x0, 0x0, 0x0, 0x0};  
char *ptr = &a[0];  
ptr = ptr + 1;
```

ptr → 0xab72

0xab71	1	1	1	1	1	1	1	1
0xab72	0	1	1	0	0	1	0	1
0xab73	0	1	1	1	1	0	0	0
0xab74	1	0	1	0	0	1	0	0
0xab75	0	0	0	0	0	0	0	0
0xab76	0	0	0	0	0	0	0	0
0xab77	0	0	0	0	0	0	0	0
0xab78	0	0	0	0	0	0	0	0

# Les pointeurs et les tableaux

```
short s[4] = {0x65ff, 0xa478}  
short *sptr = &s[0];
```



# Les pointeurs et les tableaux

```
short s[4] = {0x65ff, 0xa478}  
short *sptr = &s[0];  
sptr = sptr + 1;
```

Remarque: Notez-bien la position de **sptr**.

0xab71	1	1	1	1	1	1	1	1
0xab72	0	1	1	0	0	1	0	1
0xab73	0	1	1	1	1	0	0	0
0xab74	1	0	1	0	0	1	0	0
0xab75	0	0	0	0	0	0	0	0
0xab76	0	0	0	0	0	0	0	0
0xab77	0	0	0	0	0	0	0	0
0xab78	0	0	0	0	0	0	0	0

# Questions

Imaginez les trois variables suivantes: `int *iptr, float *fptr, double *dptr`. Elles ont toutes la même valeur (0x4).

```
iptr = iptr + 2;  
fptr = fptr + 2;  
dptr = dptr + 2;
```

Donnez les valeurs de `iptr, fptr, dptr`.

# Tableaux: indices

```
short s[4] = {0x65ff, 0xa478}  
short *sptr = &s[0];
```

ou

```
short *sptr = s;
```

# Tableaux: indices

```
short s[4] = { 0x65ff, 0xa478 }           printf( "%hd", sptr[3] )
short *sptr = s;
```

```
printf( "%hd", * (sptr+3) )
```

ou

```
printf( "%hd", * (s+3) )
```

ou

```
printf( "%hd", s[3] )
```

ou

# Les tableaux et les pointeurs

```
short s[2] = {0x65ff, 0xa478}
```

## Les notations équivalents

### Notation indicielle

`&s[0]`

`&s[i]`

`s[0]`

`s[i]`

### Notation pointeur

`s`

`s+i`

`*s`

`*(s+i)`

# Les opérateurs et les pointeurs

```
short *sptr1, *sptr2;
```

Opérateur	Exemple
pointeur + entier	sptr1 + 2
pointeur - entier	sptr1 - 2
pointeur1 - pointeur2	sptr1 - sptr2

# Les pointeurs génériques

**void \***

```
char a = 'a';
char *cptr = &a;
int i = 1;
cptr = &i;
```

```
$ gcc ...
warning: assignment from incompatible pointer type [-Wincompatible-
pointer-types]
cptr = &i;
```

# Les pointeurs génériques

**void \***

```
char a = 'a';
char *cptr = &a;
void *vptr = &a;
```

```
int i = 1;
vptr = &i;
```

```
float f = 1;
vptr = &f;
```

# Conversion de type

## Sans perte d'information

- char-> short
- short-> int
- int-> long int
- int-> float

## Exemple

```
short s = 0xffff;  
int i = s
```

**Remarque:** N'oubliez pas d'utiliser sizeof.

## Les conversions explicites

```
short s = 0xffff;  
float f = 30.999;  
int i = (int) s;  
int j = (int) f;
```

Remarque: Une conversion explicite (typecasting): (**type**)

# Conversion de type

**void \***

```
char a = 'a';
char b = 'b';
char *cptr = &a;
void *vptr = &b;

cptr = (char *)vptr;
```

# Question

**Écrivez un programme en utilisant char \* et les opérateurs de pointeurs pour voir les octets d'une variable long int.**

## Tableau

est un ensemble d'éléments homogènes

**Mais comment travailler avec un ensemble d'éléments hétérogènes?**

## Base de données de gestion des étudiants

```
char prenom[135][30];  
char nom[135][30];  
char rue[135][30];  
char ville[135][30];  
short notes[135];
```

**Question:** Quelles sont les problèmes?

## Une structure

est un ensemble d'éléments hétérogènes.

```
struct etudiant{  
    char prenom[30];  
    char nom[30];  
    char rue[30];  
    char ville[30];  
    short notes;  
};
```

## La déclaration et l'utilisation d'une structure

```
struct etudiant dupont;  
  
strcpy(dupont.prenom, "Dupont");  
strcpy(dupont.nom, "Pierre");  
strcpy(dupont.rue, "Boulevard du 11 novembre 1918");  
strcpy(dupont.ville, "Villeurbanne");  
dupont.notes = 19;
```

## Tableaux de structures

```
struct etudiant etudiant_cpe[135];  
  
strcpy(etudiant_cpe[0].prenom, "Dupont");  
strcpy(etudiant_cpe[0].nom, "Pierre");  
strcpy(etudiant_cpe[0].rue, "Boulevard du 11 novembre 1918");  
strcpy(etudiant_cpe[0].ville, "Villeurbanne");  
etudiant_cpe[0].notes = 19;
```

## Une structure dans une structure

```
struct adresse{  
    char rue[30];  
    char ville[30];  
};  
struct etudiant{  
    char prenom[30];  
    char nom[30];  
    struct adresse adresse;  
    short notes;  
};
```

## Une structure dans une structure

```
struct etudiant{  
    char prenom[30];  
    char nom[30];  
    struct adresse{  
        char rue[30];  
        char ville[30];  
    } adresse;  
    short notes;  
};
```

## Une structure dans une structure

```
struct etudiant etudiant_cpe[135];  
  
strcpy(etudiant_cpe[0].prenom, "Dupont");  
strcpy(etudiant_cpe[0].nom, "Pierre");  
strcpy(etudiant_cpe[0].adresse.rue, "Boulevard du 11 novembre 1918");  
strcpy(etudiant_cpe[0].adresse.ville, "Villeurbanne");  
etudiant_cpe[0].notes = 19;
```

## Créer (et Renommer) un type

```
typedef struct etudiant{  
    char prenom[30];  
    char nom[30];  
    struct adresse{  
        char rue[30];  
        char ville[30];  
    } adresse;  
    short notes;  
} etudiant;
```

## La déclaration

```
etudiant dupont;
```

## Renommer un type

```
typedef int integer;  
integer i = 10;
```

# Bonjour le Monde!!

```
/* Fichier: bonjour2.c
 * affiche un message à l'écran en utilisant un variable
 * auteur: John Samuel
 * Ceci est un commentaire sur plusieurs lignes
 */
#include <stdio.h> // en-têtes(headers)

int main() {
    int year = 2017; //déclaration d'un variable
    printf("Bonjour le Monde!!! C'est l'annee %d", year);
    return 0;
}
```

## Prototype

```
int add( int, int);
```

## L'implémentation

```
int add( int a, int b ) {  
    return a + b;  
}
```

## Prototype (operators.h)

```
int add( int, int);
```

## L'implémentation (operators.c)

```
int add( int a, int b ) {  
    return a + b;  
}
```

## Prototype

```
type fonction( [type, ]*);
```

## L'implémentation

```
type fonction( [type variable, ]* ) {  
    [return valeur];  
}
```

**Remarque:** type: void, les types de base, les types composés

## L'utilisation

```
#include "operators.h" // en-têtes (headers)

int num1 = 20, num2 = 60;
int sum = add( num1, num2 );
```

**Remarque:** Regardez l'utilisation de " " dans en-têtes

## Prototype (name.h)

```
void print( char *, int);
```

Remarque: Il n'y a pas de return

## L'implémentation (name.c)

```
#include <stdio.h> // en-têtes (headers)

void print( char * name, int size ) {
    int i;
    for( i = 0; i < size; i++ ) {
        printf( "%c", name[i] );
    }
}
```

# Fonctions

```
/* Fichier: bonjour3.c
 * affiche un message à l'écran en utilisant print
 * auteur: John Samuel
 */
#include "name.h" // en-têtes(headers)

int main() {
    print( "Bonjour le Monde!!!!");
    char message[] = "Pierre";
    print("Je suis ");
    print(message);
    return 0;
}
```

## La compilation

```
$ gcc -o bonjour bonjour3.c name.c
```

## L'exécution

```
$./bonjour  
Bonjour le Monde!!! Je suis Pierre
```

## Les outils Linux

- \$ ls -l
- \$ cd dossier
- \$ cat fichier
- ...

# L'interface en ligne de commande

```
/* Fichier: bonjour4.c
 * affiche un message à l'écran en utilisant les arguments de la ligne
 * de commandes.
 * auteur: John Samuel
 */

#include "name.h" // en-têtes(headers)

int main(int argc, char ** argv) {
    print("Bonjour le Monde. Je suis ");
    print(argv[1]);
    return 0;
}
```

## La compilation

```
$ gcc -o bonjour bonjour4.c name.c
```

## L'exécution

```
$./bonjour John  
Bonjour le Monde. Je suis John
```

# L'interface en ligne de commande

```
/* Fichier: bonjour4.c
 * affiche un message à l'écran en utilisant les arguments de la ligne
 * de commandes.
 * auteur: John Samuel
 */

#include "name.h" // en-têtes(headers)

int main(int argc, char ** argv) {
    print("Bonjour le Monde. Je suis ");
    if ( argc == 2 ) {
        print(argv[1]);
    }
    return 0;
}
```

# La saisie d'une valeur par l'utilisateur

```
/* Fichier: addition.c
 * affiche un message à l'écran en utilisant scanf.Remarque: Regardez l'op-
 * auteur: John Samuel
 */

#include <stdio.h> // en-têtes(headers)

int main(int argc, char ** argv) {
    int num1, num2;
    printf("Tapez numéro 1");
    scanf("%d", &num1);
    printf("Tapez numéro 2");
    scanf("%d", &num2);
    printf("La somme: %d\n", num1 + num2);
    return 0;
}
```

# La saisie d'une valeur par l'utilisateur

```
/* Fichier: addition.c
 * affiche la somme de deux numéros saisis par l'utilisateur
 * auteur: John Samuel
 */

#include <stdio.h> // en-têtes(headers)

int main(int argc, char ** argv) {
    int num1, num2;
    printf("Tapez deux numéros: ");
    scanf("%d %d", &num1, &num2);
    printf("La somme: %d\n", num1 + num2);
    return 0;
}
```

**Remarque:** Regardez l'opérateur &.

# La saisie d'une valeur par l'utilisateur

```
/* Fichier: bonjour5.c
 * affiche un message à l'écran en utilisant scanf.
 * auteur: John Samuel
 */

#include <stdio.h> // en-têtes(headers)

int main(int argc, char ** argv) {
    char nom[50];
    printf("Bonjour. Votre nom? ");
    scanf("%s", nom);
    printf("Bonjour %s!", nom);
    return 0;
}
```

**Remarque:** Regardez **s** sans l'opérateur **&**.

## Mots clés

## Code de conversion

char

c

unsigned char

hhu

short

hd

unsigned short

hu

int

d, i

unsigned int

u

long int

ld

unsigned long int

lu

## Mots clés

## Code de conversion

long long int

lld

unsigned long long int

llu

float

f, F

double

g, G

long double

Lg

string of characters

s

# Les manuels Linux

```
$ man ls  
$ man 3 scanf  
$ man 3 printf  
$ man 2 open  
$ man fopen  
..
```

# Les manuels Linux

NAME

scanf

...

SYNOPSIS

...

DESCRIPTION

...

RETURN VALUE

...

..

# La manipulation d'une chaîne de caractères

```
/* Fichier: string.c
 * manipulation d'une chaîne de caractères. Remarque: strlen calcule la taille d'u
 * auteur: John Samuel
 */

#include <stdio.h> // en-têtes(headers)
#include <string.h>

int main(int argc, char ** argv) {
    char nom[10];
    printf("Bonjour. Votre nom? ");
    scanf("%s", nom);
    printf("La taille: %lu\n", strlen(nom));
    return 0;
}
```

# La manipulation d'une chaîne de caractères

## La compilation

```
$ gcc -o strlen string.c
```

## 1<sup>ere</sup> Exécution

```
./strlen  
Bonjour. Votre nom? John  
La taille: 4
```

## 2<sup>eme</sup> Exécution

```
./strlen  
Bonjour. Votre nom? : ABCDEFGHIJKLMNOPQRSTUVWXYZ  
*** stack smashing detected ***: ./strlen terminated
```

# La manipulation d'une chaîne de caractères

```
/* Fichier: string.c
 * manipulation d'une chaîne de caractères.
 * auteur: John Samuel
 */

#include <stdio.h> // en-têtes(headers)
#include <string.h>

int main(int argc, char ** argv) {
    char nom[10];
    printf("Bonjour. Votre nom? ");
    scanf("%9s", nom);
    printf("La taille: %lu\n", strlen(nom, 10));
    return 0;
}
```

Remarque: `strlen` calcule la ta

# La manipulation d'une chaîne de caractères

```
/* Fichier: string.c
* sprintf.
* auteur: John Samuel
*/
#include <stdio.h> // en-têtes(headers)
#include <string.h>

int main(int argc, char ** argv) {
    char nom[10];
    printf("Bonjour. Votre nom? ");
    scanf("%9s", nom);
    printf("La taille: %lu\n", strlen(nom, 10));
    return 0;
}
```

Remarque: `strlen` calcule la ta

# La manipulation d'une chaîne de caractères

```
/* Fichier: string.c
* sprintf.
* auteur: John Samuel
*/
#include <stdio.h> // en-têtes(headers)
#include <string.h>

int main(int argc, char ** argv) {
    char nom[10];
    printf("Bonjour. Votre nom? ");
    scanf("%9s", nom);
    printf("La taille: %lu\n", strlen(nom, 10));
    return 0;
}
```

Remarque: `strlen` calcule la ta

# La manipulation d'une chaîne de caractères

```
/* Fichier: string.c
* atoi
* auteur: John Samuel
*/
#include <stdio.h> // en-têtes(headers)
#include <string.h>

int main(int argc, char ** argv) {
    char nom[10];
    printf("Bonjour. Votre nom? ");
    scanf("%9s", nom);
    printf("La taille: %lu\n", strlen(nom, 10));
    return 0;
}
```

Remarque: `strlen` calcule la ta

# La manipulation d'une chaîne de caractères

```
/* Fichier: string.c
* sscanf
* auteur: John Samuel
*/
#include <stdio.h> // en-têtes(headers)
#include <string.h>

int main(int argc, char ** argv) {
    char nom[10];
    printf("Bonjour. Votre nom? ");
    scanf("%9s", nom);
    printf("La taille: %lu\n", strlen(nom, 10));
    return 0;
}
```

Remarque: `strlen` calcule la ta

# La manipulation d'une chaîne de caractères

```
/* Fichier: string.c
* sscanf
* auteur: John Samuel
*/
#include <stdio.h> // en-têtes(headers)
#include <string.h>

int main(int argc, char ** argv) {
    char nom[10];
    printf("Bonjour. Votre nom? ");
    scanf("%9s", nom);
    printf("La taille: %lu\n", strlen(nom, 10));
    return 0;
}
```

Remarque: `strlen` calcule la ta

# La manipulation d'une chaîne de caractères

## La compilation

```
$ gcc -o strnlen string2.c
```

## 1<sup>ere</sup> Exécution

```
$ ./strnlen  
Bonjour. Votre nom? John  
La taille: 4
```

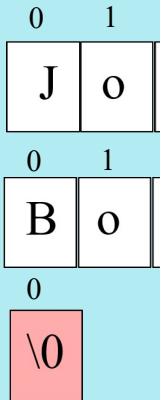
# La manipulation d'une chaîne de caractères

- `strcat` concatène deux chaînes de caractères
- `strncat` concatène deux chaînes de caractères avec une taille maximum donnée par l'utilisateur.
- `strcpy` copie deux chaînes de caractères
- `strncpy` copie deux chaînes de caractères avec une taille maximum donnée par l'utilisateur
- `strcmp` compare deux chaînes de caractères
- `strncmp` compare deux chaînes de caractères avec une taille maximum donnée par l'utilisateur

# La manipulation d'une chaîne de caractères

```
/* Fichier: string3.c
 * manipulation d'une chaîne de caractères.
 * auteur: John Samuel
 */
#include <stdio.h> // en-têtes(headers)
#include <string.h>

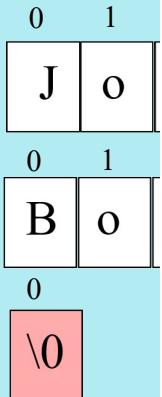
int main(int argc, char ** argv) {
    char message[] = {"Bonjour"}, nom[] = {"John"}, string[20] = "";
    strncat(string, message, 19);
    strncat(string, " ");
    strncat(string, nom, 19);
    printf("%s", message));
    return 0;
}
```



# La manipulation d'une chaîne de caractères

```
/* Fichier: string4.c
 * manipulation d'une chaîne de caractères.
 * auteur: John Samuel
 */
#include <stdio.h> // en-têtes(headers)
#include <string.h>

int main(int argc, char ** argv) {
    char message[] = {"Bonjour"}, nom[] = {"John"}, string[20] = "";
    strncpy(string, message, 19);
    strcpy(string, " ");
    strncpy(string, nom, 19);
    printf("%s", message));
    return 0;
}
```



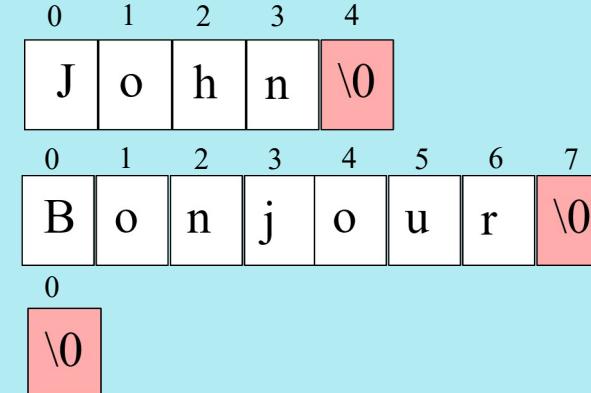
# La manipulation d'une chaîne de caractères

## La compilation

```
$ gcc -o strncat string3.c  
$ gcc -o strncpy string4.c
```

## 1<sup>ere</sup> Exécution

```
./strncat  
Bonjour John
```



## 2<sup>eme</sup> Exécution

```
./strncpy  
John
```

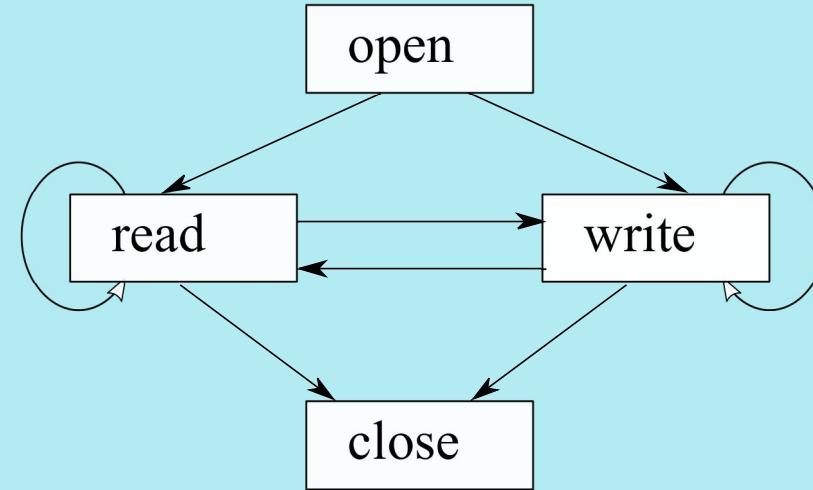
# La manipulation de fichiers

```
/* Fichier: file.c
 * manipulation d'une fichier
 * auteur: John Samuel
 */

#include <stdio.h> // en-têtes(headers)
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

int main(int argc, char ** argv) {
    char content[1000];
    int fd, count, size;

    fd = open("./file.c", O_RDONLY);
    size = read(fd, content, 1000);
    for (count = 0; count < size; count++) {
        printf("%c", content[count]);
    }
    close(fd);
    return 0;
}
```

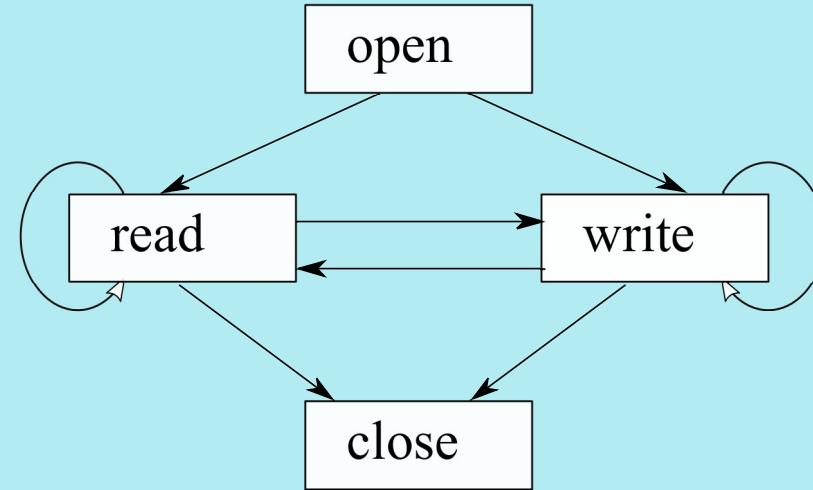


# La manipulation de fichiers

```
/* Fichier: file.c
 * manipulation d'une fichier
 * auteur: John Samuel
 */

#include <stdio.h> // en-têtes(headers)
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

int main(int argc, char ** argv) {
    char content;
    int fd, count, size;
    fd = open("./file.c", O_RDONLY);
    while (1) {
        size = read(fd, &content, 1);
        if (size < 1) {
            break;
        }
        printf("%c", content);
    }
    close(fd);
    return 0;
}
```

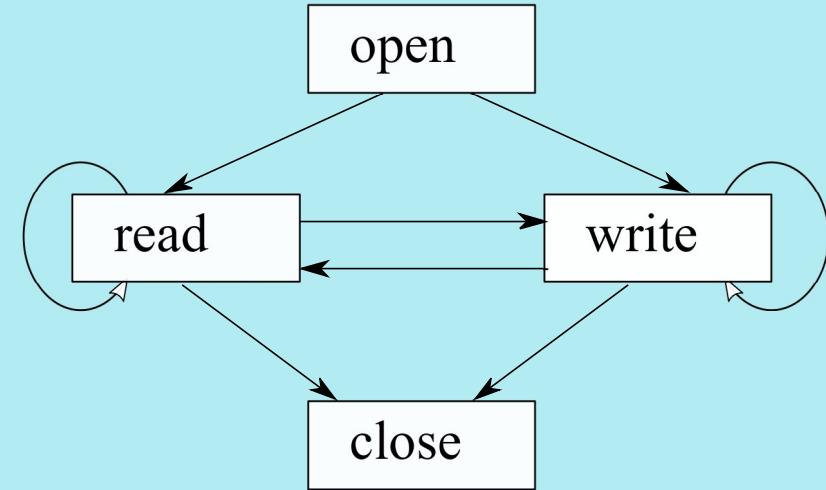


# La manipulation de fichiers

```
/* Fichier: file.c
 * manipulation d'une fichier
 * auteur: John Samuel
 */
#include <stdio.h> // en-têtes(headers)
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

int main(int argc, char ** argv) {
    char content[] = "Bonjour";
    int fd, count, size;

    fd = open("./message.txt", O_CREAT|O_WRONLY);
    size = write(fd, &content, sizeof(content));
    close(fd);
    return 0;
}
```



# Allocation dynamique de mémoire

```
#include <stdlib.h>

void * malloc( size_t size);
void * calloc( size_t nmemb, size_t size);
void free( void * ptr); // désallocation ou libération de mémoire
```

# Allocation dynamique de mémoire

```
/* Fichier: memory.c
 * Allocation dynamique de mémoire
 * auteur: John Samuel
 */

#include <stdio.h> // en-têtes(headers)
#include <stdlib.h>

int main(int argc, char ** argv) {
    char *content = malloc(10 * sizeof(char));
    strncat(content, "Bonjour", 10);
    free(content); //libération de mémoire
    return 0;
}
```

# Allocation dynamique de mémoire

```
/* Fichier: memory.c
 * Allocation dynamique de mémoire
 * auteur: John Samuel
 */

#include <stdio.h> // en-têtes(headers)
#include <stdlib.h>

int main(int argc, char ** argv) {
    char *content = calloc(10, sizeof(char));
    strncat(content, "Bonjour", 10);
    free(content); // libération de mémoire
    return 0;
}
```

# Question

**Écrivez un programme qui réserve et libère un espace mémoire pour un tableau d'entiers (int, long int, short ou long long int) et un tableau de nombres en flottant (float, double ou long double) en utilisant malloc, calloc et free.**

## Une union

est déclarée comme une structure.

```
union etudiant{  
    char prenom[30];  
    char nom[30];  
    char rue[30];  
    char ville[30];  
};
```

**Remarque:** La taille d'une union est égale à la taille du élément plus grand.

## La déclaration et l'utilisation d'une union

Remarque: Tous les éléments partagent la même mémoire

```
union etudiant dupont;
```

```
strcpy(dupont.nom, "Dupont");
printf("%s", dupont.prenom);
printf("%s", dupont.rue);
printf("%s", dupont.ville);
```

## L'affichage

```
Dupont
Dupont
Dupont
```

# Les unions

```
struct content{
    char type[30];
    union{
        char * ccontent;
        int * icontent;
        float * fcontent;
    };
};
```

# Les unions

```
/* Fichier: memory3.c
 * Allocation dynamique de mémoire
 * auteur: John Samuel
 */

#include <stdio.h> // en-têtes(headers)
#include <stdlib.h>

int main(int argc, char ** argv) {
    struct content c;
    strcpy(c.type, "char", 10);
    if (strcmp(c.type, "char") == 0) {
        c.ccontent = calloc(10, sizeof(char));
    }
    else if (strcmp(c.type, "int") == 0) {
        c.icontent = calloc(10, sizeof(int));
    }
    return 0;
}
```

# Les unions

```
union point3d{  
    int value[3];  
    struct{  
        int x;  
        int y;  
        int z;  
    };  
};
```

# Les unions

```
/* Fichier: union.c
 * la déclaration et l'utilisation d'une union
 * auteur: John Samuel
 */

#include <stdio.h> // en-têtes(headers)

int main(int argc, char ** argv) {
    union point3d p;
    p.values[0] = 0x10;
    p.values[1] = 0x45;
    p.values[2] = 0x78;

    printf("%x %x %x\n", p.x, p.y, p.z);
    return 0;
}
```

# Question

**Écrivez une structure pour la représentation d'une couleur RGB (rouge, vert, bleu: chaque couleur prend un octet) en utilisant union et struct.**

## Références

- Langage C, Claude Delannoy
- [https://fr.wikipedia.org/wiki/Architecture\\_32\\_bits](https://fr.wikipedia.org/wiki/Architecture_32_bits)
- <https://en.wikipedia.org/wiki/X86>
- <https://fr.wikipedia.org/wiki/Endianness>
- [https://fr.wikipedia.org/wiki/IEEE\\_754](https://fr.wikipedia.org/wiki/IEEE_754)

## Crédits d'images

- [Wikimedia Commons](#)