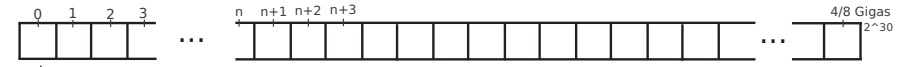


Organisation mémoire

001

Organisation mémoire

Chaque case est numérotée: son adresse



Rem. Pointeur NULL = Pointeur vers la case numéro 0

003

Organisation mémoire

Mémoire RAM = Suite élément mémoire



1011 0010

B2

1 case = 1 octet (+ petit élément accessible)
= 8 bits
(= 2 digits hexadecimals)

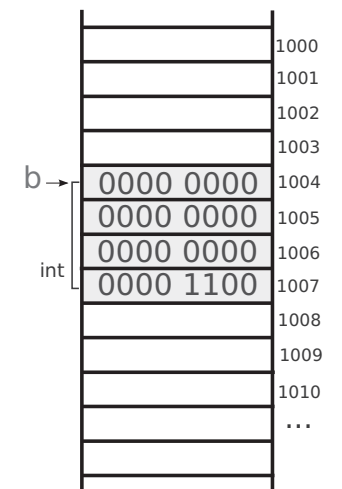
002

Variable

Instancier une variable revient à

```
int b=12;
```

- Prendre une case en mémoire
- Taille de la case donnée par le type
(gestion $\text{int} \leq 4$ octets géré par le code C)
- Désigner cette case par le nom de variable
(gestion $\text{nom}[b] \leq @1004$ géré par le code C)
- Remplir la case par la valeur désignée

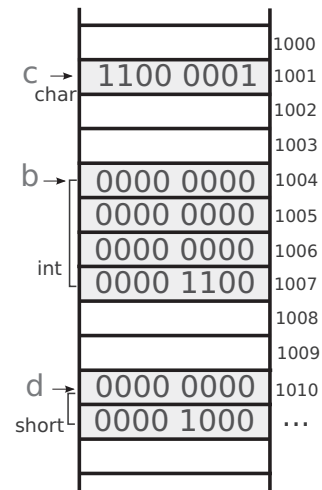


004

Variable

Plusieurs variables, pas forcément contigues

```
int b=12;
char c='a';
short d=8;
```



005

Variable adresse

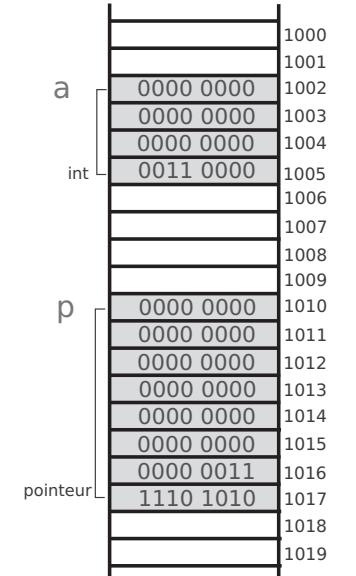
Une variable de type adresse/pointeur:

- Contient le numéro d'une case mémoire
- Le type encode la taille du type pointé
- Est stockée comme une variable de 4/8 octets

```
int a=48;
int* p=&a;
```

Le type pointé

- n'est pas nécessaire pour le stockage
- est utile pour lire la valeur de *p



007

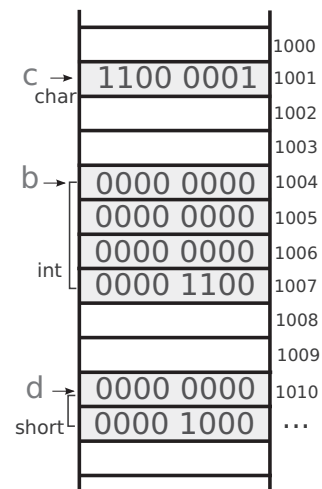
Lecture de variables

```
int b=12;
char c='a';
short d=8;
```

Au niveau assembleur:

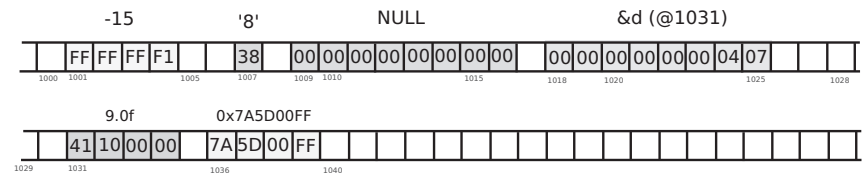
- Pour lire la valeur de b:
 - On se place sur l'élément @1004 en RAM
 - On lit 4 octets (int)
- Pour lire la valeur de c:
 - On se place sur l'élément @1001 en RAM
 - On lit 1 octet (char)
- Pour lire la valeur de d:
 - On se place sur l'élément @1010 en RAM
 - On lit 2 octets (short)

Géré automatiquement par le code C



006

Exemple



```
int a=-15;
char b='8';
int c=0x7A5D00FF;
float d=9.0f;
float *p1=NULL;
float *p2=&d;
```

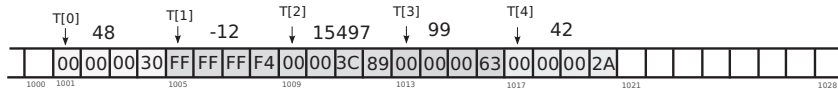
008

Elements contigus

En C, deux manière de définir des éléments contigus en mémoire.

- 1: Tableau (éléments de même types)

```
int T[5]={48, -12, 15497, 99, 42};
```



Rem: Le code C stocke que T vaut 1001 (à la compilation)

T[0] vient lire 4 octets (int) à partir de 1001
 T[1] vient lire 4 octets (int) à partir de 1001+1*4 (4=sizeof(int))
 T[2] vient lire 4 octets (int) à partir de 1001+2*4

Attention: *(T+i) doit lire 4 octets à partir de 1001+i*4

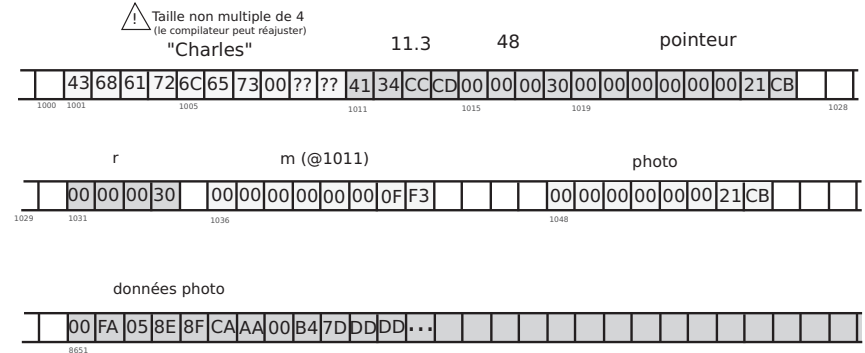
=> T+1 ne vaut pas 1002!
 T+1 vaut 1001+1*sizeof(int)=1005 ! Arithmétique des pointeurs dépend de la taille du type!
 T+i vaut 1001+i*sizeof(int)

009

Exemple

```
struct etudiant
{
    char nom[10];
    float moyenne;
    int rang;
    FILE* photo;
};
```

@8651
 FILE* photo=fopen(...)
 ...
 struct etudiant e={"Charles", 11.3, 48, photo};
 int r=e.rang;
 float *m=&e.moyenne;



011

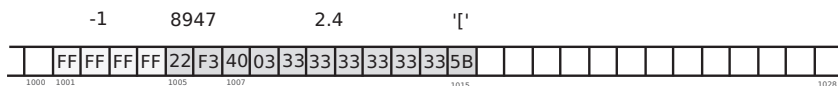
Elements contigus

En C, deux manière de définir des éléments contigus en mémoire.

- 2: Structures (types hétérogènes)

```
struct s
{ int a; short b; double c; char d; };
```

```
struct s={-1, 8947, 2.4, '['};
```

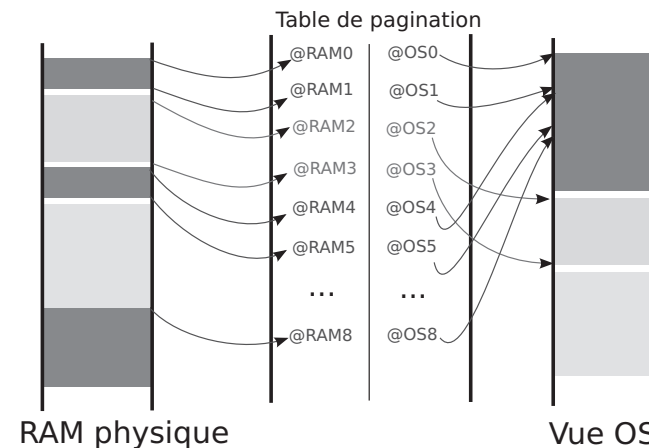


s.a vient lire 4 octets (type int) à partir de 1001
 s.b vient lire 2 octets (type short) à partir de 1005
 s.c vient lire 8 octets (type double) à partir de 1007
 s.d vient lire 1 octet (type char) à partir de 1015

010

Mémoire physique vs OS

En pratique:
 RAM fragmentée, pas forcément d'espace contigue
 L'OS "fait croire" qu'il y a de la mémoire contigue (pagination)



012

Endianness

Ex.
Le nombre stocké en "long long" valant 8

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 08

Il faut lire beaucoup d'octets avant de connaître la valeur

=> Sur les système **little endian**, on stocke/lit les octets de poids faibles d'abord.

Little Endian: (Intel, ...)

08 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Big Endian: (Motorola, anciens Macintosh)

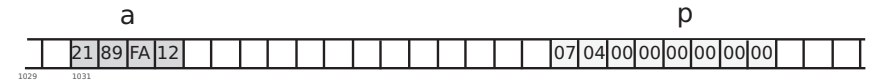
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 08

013

Lecture d'octets

Il est possible de lire n'importe quel octet de la mémoire et allant le "pointer" avec un char*

Ex. int a=0x12FA8921;
char *p=&a;



char c0=*p; //ou p[0]
printf("%x\n",c0);
> 21

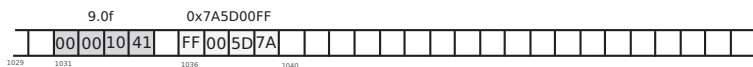
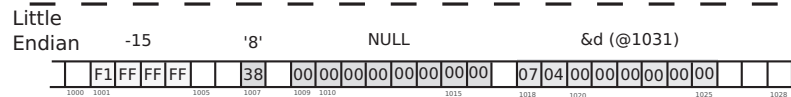
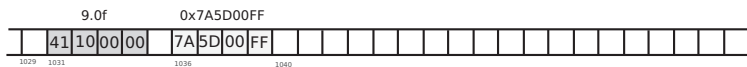
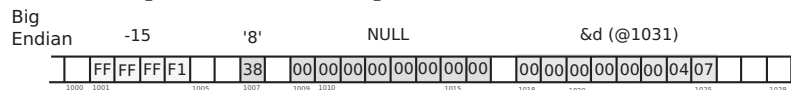
char c1=*(p+1); //ou p[1]
printf("%x\n",c1);
> 89

015

Endianness

En pratique, on inverse les octets de chaque entité

```
int a=-15; char b='8';
int c=0x7A5D00FF; float d=9.0f;
float *p1=NULL; float *p2=&d;
```

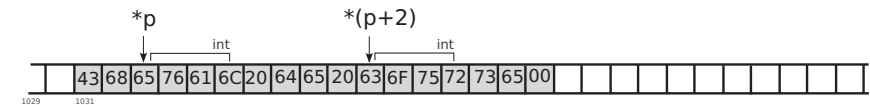


014

Lecture d'octets

Autre type de pointeurs

Ex. char T[17]="Cheval de course";
int *p=&T[2]; 43 68 65 76 61 6C 20 64 65 20 63 6F 75 72 73 65 00



printf("%x\n", *p); > 6C 61 76 65

printf("%d\n", *p); > 1818326629

printf("%x\n", p[2]); > 72 75 6F 63

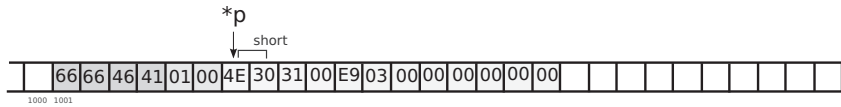
printf("%d\n", p[2]); > 1920298851

016

Exemple

```
struct noeud          struct noeud n={12.4,1,"N01",NULL};
{
  float valeur;
  short id;
  char nom[4];
  struct noeud* suivant;
};
```

```
12.4  -> 41 46 66 66
1      -> 00 01
"N01" -> 4E 30 31 00
NULL   -> 00 00 00 00 00 00 00 00
1001   -> 00 00 00 00 00 00 03 E9
```



```
short *p=&n.nom;
printf("%x\n", *p);      > 30 4E

printf("%x\n", p[1]);    > 00 31
printf("%s\n", p[1]);    > 49

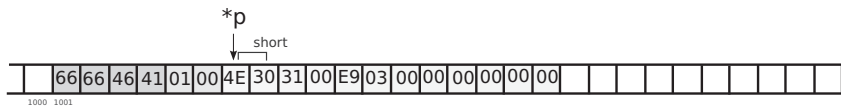
printf("%x\n", p[-1]);   > 00 01
printf("%s\n", p[-1]);   > 01
```

017

Exemple

```
struct noeud          struct noeud n={12.4,1,"N01",NULL};
{
  float valeur;
  short id;
  char nom[4];
  struct noeud* suivant;
};
```

```
12.4  -> 41 46 66 66
1      -> 00 01
"N01" -> 4E 30 31 00
NULL   -> 00 00 00 00 00 00 00 00
1001   -> 00 00 00 00 00 00 03 E9
```



```
short *p=&n.nom;
printf("%s\n", p-3); // ou &p[-3]
> ffDA
```

018