Programmation avancée en C pour l'embarqué

STUDENT Edition

Ver 2.1-0

S. Bazine <sbazine.ens@gmail.com>

2025-2026

Table des matières

1 Données et manipulation du Binaire

1.1 Types standards en C

Types standards des variables

— Le `C` dispose de 6 types de variable :

Type	signification	val. min	val. max
`char`	caractère codé sur 1 octet (8 bits)	-2^7	2^7-1
`short`	entier codé sur 2 octet	-2^{15}	$2^{15}-1$
`int`	entier codé sur 4 octets	-2^{31}	$2^{31}-1$
`long`	entier codé sur 8 octets	-2^{63}	$2^{63}-1$
`float`	réel codé sur 4 octets	-10^{38}	10^{38}
`double`	réel codé sur 8 octets	-10^{308}	10^{308}

— Le préfixe `unsigned` permet de forcer les variables à prendre des valeurs positives.

1.2 La bibliothèque `stdint.h`

la bibliothèque `stdint.h`

— La bibliothèque `stdint.h` déclare des ensembles de types entiers ayant des largeurs spécifiées et définit des ensembles de macros correspondants :

Type	signification	val. min	val. max
`uint8_t`	entier non signé codé sur 1 octet (8 bits)	0	$2^8-1=255$
`int8_t`	entier signé codé sur 1 octet (8 bits)	-2^7	2^7-1
`uint16_t`	entier non signé codé sur 2 octet	0	$2^{16}-1=65535$
`int16_t`	entier signé codé sur 2 octet	-2^{15}	$2^{15}-1$
`uint32_t`	entier non signé codé sur 4 octets	0	$2^{32}-1=4294967295$
`int32_t`	entier signé codé sur 4 octets	-2^{31}	$2^{31}-1$
`uint64_t`	entier non signé codé sur 8 octets	0	$2^{64}-1=18446744073709551615$
`int64_t`	entier signé codé sur 8 octets	-2^{63}	$2^{63}-1$

1.3 Opérateurs logiques et opérateurs binaires

Opérateurs logiques et opérateurs binaires

— opérateurs logiques `==`, `<`, `<=`, `>`, `>=`, `!=`, `&&`, `||`.

```c
char a,b,c;
a=5; b=0; c=1;
b && (c > 0)          /* FAUX */
(a > c) && (b = 3)    /* VRAI et b reçoit 3 */
```

⇒ *Attention à la différence entre* `=` *et* `==`

— opérateurs binaires `!, &, |, ^, <<, >>` :

```c
char a,b=10;
a = b & 3;     /* a = 2 = 0b0000 0010 */
a = a << 1;      /* a = 4 = 0b0000 0100 */
```

Exercice 1 :

Donner un programme C permettant de :

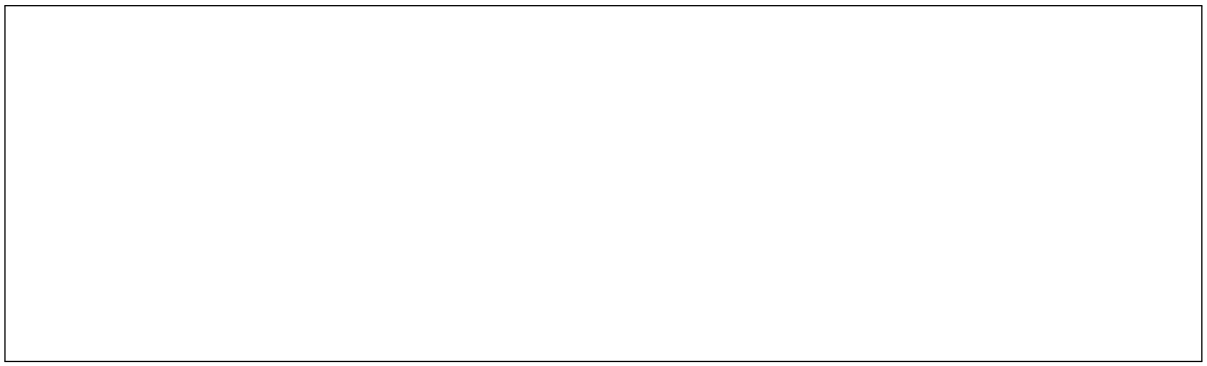
1. Définir deux entiers `i` et `j` initialisés avec les valeurs `10` et `3` respectivement.
2. Affecter les valeurs `0x0FF` et `0xF0F` respectivement à `i` et `j`,
3. afficher en hexadécimal les résultats de :

```c
i & j
i | j
i ^ j
i << 2
j >> 2
```

4. Donner le résultat de l'exécution de ce programme.

Réponses exercice 1 (Questions 1, 2 et 3) :

Réponses exercice 1 (Question 4) :



2 Les tableaux

2.1 Tableau mono-dimensionnel

Les Tableaux

Déclaration et Syntaxe :

— Syntaxe : `type nomtab[<taille>]<=\{val1, val2, ...\}>;`

`type` : le type de chaque case du tableau,

`nomtab` : le nom du tableau ; c'est un pointeur sur la première case du tableau.

`<taille>` : la taille explicite du tableau,

`<={val1, val2, ...}>` : suite de valeurs d'initialisation.

— le nombre des cases du tableau peut être explicite via le paramètre `taille`, ou bien déduit de la suite des valeurs d'initialisation.

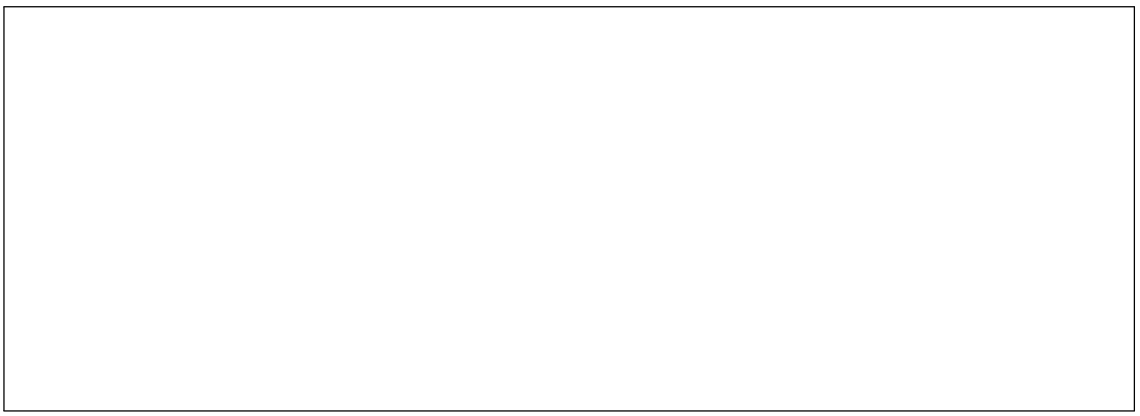
Les Tableaux

Déclaration et mémoire :

— Exemple :

```
1 float tabF[5];
2 int   i, tabI[]={1,2,3,4,5};
3 for (i=0; i<5; i++)
4     tabF[i] = tabI[i]*.1;
```

Aperçu de la mémoire :



— Initialisation partielle :

```
1 char ch1[10]={'H','e','l','l','o',0x41,0x31};
```

2.2 Chaîne de caractères

Les tableaux :

Les chaînes de caractères

le compilateur rajoute toujours le caractère `null` à la fin du tableau.

— tableau de caractères :

```
1 char ch2[10]="Hello";
2 char ch3[]="Hello";
```

— Affichage :

```
1 printf("%s\n%s\n%s\n",ch1,ch2,ch3);
```

2.3 Tableau multidimensionnel

Déclaration de tableau multidimensionnel

— syntaxe :

`type nomtab[<taille1>]...[<taille N>]<=\{val1, val2, ...\}>;`

`<taillei>` : le nombre des cases de la i^{ème} dimension.

— initialisation partielle :

```
1 int tab2D1[2][3]={{11,12,13},{21}};
2 int tab2D2[2][3]={11,12,13,21};
3 int tab2D3[][3]={11,12,13,21,22,23,31};
4 tab2D3[2][1]=32;
```

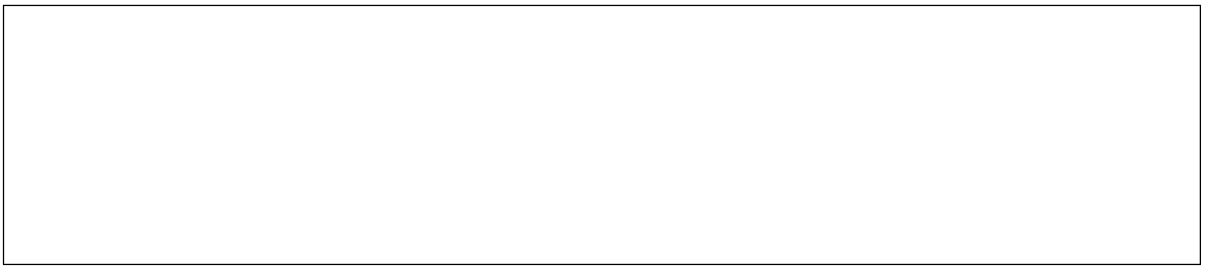
Manipulation de tableaux multidimensionnels

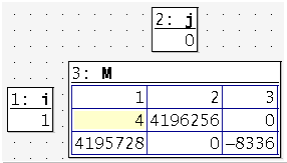
Soit le programme `manipTab.c` suivant :

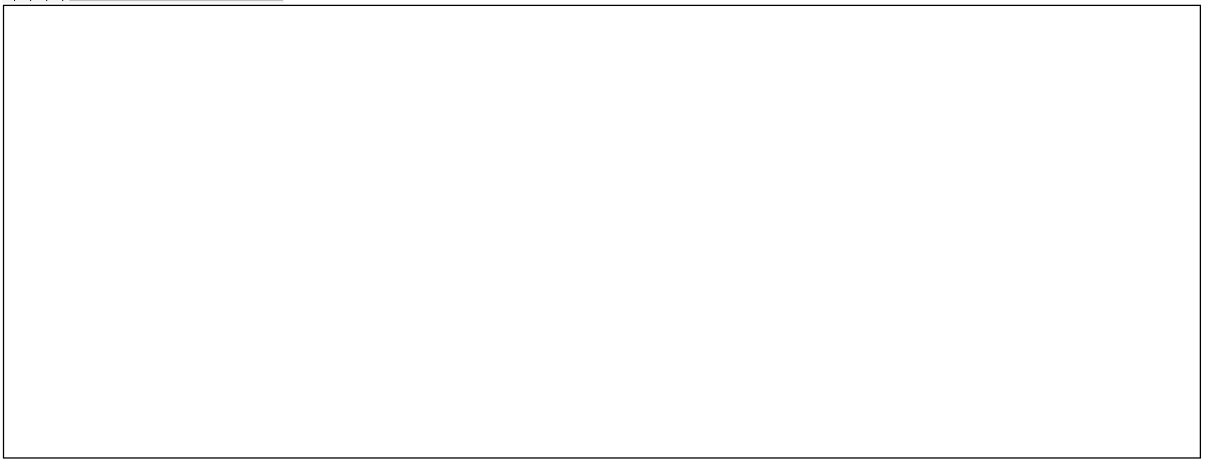
```c
#include <stdio.h>
int main(){
  int i,j, M[3][3];
  printf("Saisie de 3x3 cases:\n");
  for(i=0;i<3;i++)
    for(j=0;j<3;j++){
      printf("M(%d,%d)=",i,j);
      scanf("%d",&M[i][j]);
    }
  printf("Affichage de la matrice:\n");
  for(i=0;i<3;i++){
    for(j=0;j<3;j++)
      printf("%d\t",M[i][j]);
    printf("\n");
  }
  return 0;
}
```

Exécution :

```
Saisie de 3x3 cases :
```







3 Les Structures

3.1 Déclaration des structures en C

Déclaration des Structures

— Les structures permettent de manipuler des valeurs de type différent au sein d'une même entité.

— syntaxe :

```
struct NomStruct{
   type1 nomChamp1<[n1]>;
   ...
   typei nomChampi<[ni]>;
}<nomVar1, ...nomVari>;
```

`NomStruct :` Étiquette de la structure,

`nomChampi :` le nom du champ i à l'intérieur de la structure. Il peut être un tableau de taille `[ni]`.

`nomVari :` Le nom de la variable de type `NomStruct`.

— déclaration des variables de type structure :

```
struct NomStruct nomVar1<[n1]>, ...nomVari<[ni]>;
```

Déclaration des Structures

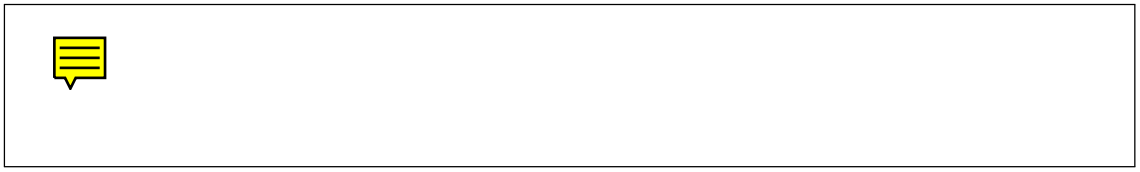
— Exemple :

```
struct Contact{
  char Name[30];
  int  Phone;
  char Email[40];
}contact, Contacts[100];
```

`Contact:` Structure constituée de trois champs `Name, Phone` et `Email` ;

`contact:` Variable de type structure `Contact`. Elle est de taille **74 Octets**.

`Contacts :`



3.2 Type structures

Définir un nouveau type `typedef struct :`

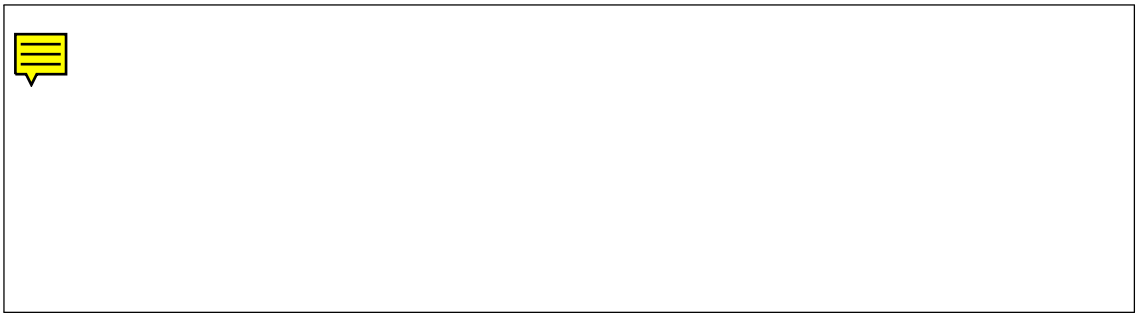
— Type : `UartConfig_t`

```c
#include <stdint.h>
typedef struct {
    uint32_t baudRate;
    uint8_t dataBits;
    uint8_t parity;
    uint8_t stopBits;
    uint8_t TxGPIO;
    uint8_t RxGPIO;
} UartConfig_t;

```

— Type : `Device_t` :

```c
typedef struct {
    char deviceName[20];
```

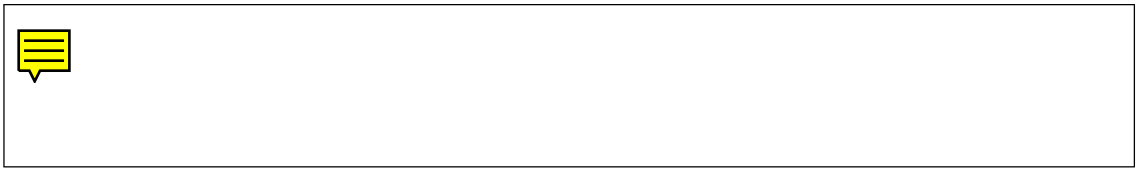


Déclaration et manipulation de type structure :

— Utilisation :

— tailles :

```c
    printf("Size of Device     = %lu Bytes\n",sizeof(GPS));
    printf("Size of UartConfig = %lu Bytes\n",sizeof(UART1));
```



3.3 Affectation et modification des structures en C

Affectation et modification

— Affectation :

— Modification/Initialisation :

```
9     GPS.Config.baudRate = 115200;
10    sprintf(GPS.deviceName, "GPS Module");
11    for(int i=0; i<sizeof(GPS.TransmitBuffer); i++){
12        GPS.ReceiveBuffer[i] = GPS.TransmitBuffer[i] = 0;
13    }
14
```

Affichage I :

```
23    printf("==========Dvice Config============\n");
24    printf("Device: %s\n",      Modem4G.deviceName);
25    printf("\tBaud Rate: %u\n", Modem4G.Config.baudRate);
26    printf("\tData Bits: %u\n", Modem4G.Config.dataBits);
27    printf("\tParity: %u\n",    Modem4G.Config.parity);
28    printf("\tStop Bits: %u\n", Modem4G.Config.stopBits);
29    printf("\tTx/Rx GPIOs: %u/%u\n", Modem4G.Config.TxGPIO, Modem4G.Config.RxGPIO);
30    printf("==================================\n");
```

Affichage II :

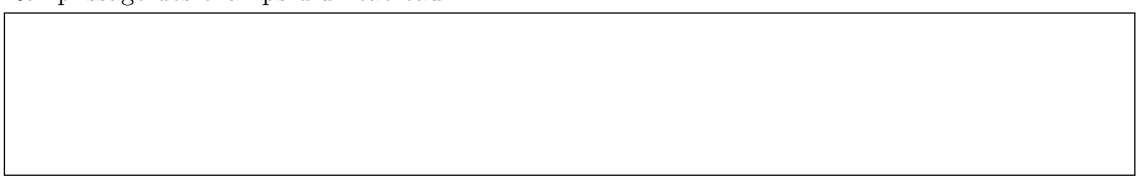
```
==========Dvice Config============
Device: 4G Module
   Baud Rate: 9600
   Data Bits: 8
   Parity: 0
   Stop Bits: 1
   Tx/Rx GPIOs: 14/15
==================================
```

Vous pouvez envisager de regrouper les lignes précédentes dans une fonction `AfficheDev(Device_t*)` (La section suivante va expliquer l'utilité d'utiliser des pointeurs ...)

3.4 Tableau de structure en C

Tableau de structures :

— Remplissage des champs d'un tableau :



— Affichage :

```c
printf("Device: %s\n",      devices[1].deviceName);
printf("\tBaud Rate: %u\n", devices[1].Config.baudRate);
printf("\tData Bits: %u\n", devices[1].Config.dataBits);
```

```
Device: "Device 1!!
    Baud Rate: 115200
    Data Bits: 8
    ...
```

4 Manipulation des pointeurs en C

4.1 Déclaration et manipulation

Déclaration et manipulation des pointeurs

— Un pointeur est une variable capable de contenir l'adresse d'autres variables.

— Un pointeur est associé à un seul type de variable.

— Exemple :

```
3       unsigned int a=0xaaaaaa;
4       unsigned short b=0xbbb;
5       unsigned char c=0xc;
6       int* pta=&a;
7       short* ptb=&b;
8       char* ptc=&c;
```

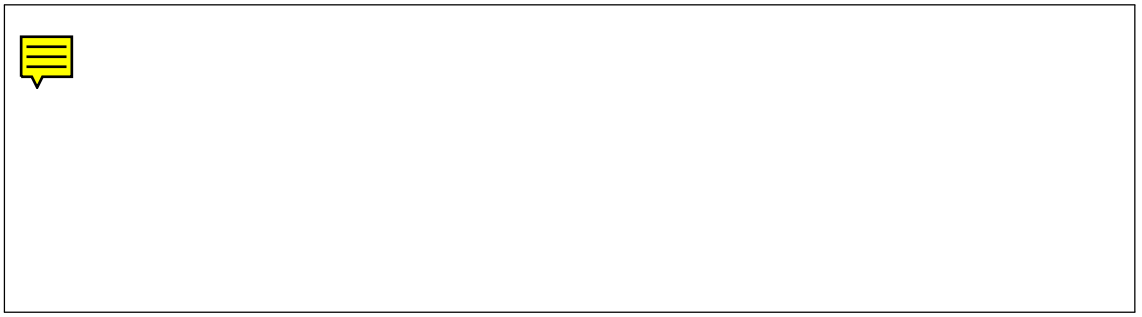
Pointeur vs variable

```
10      printf("c\t(%x)\t|%02x|\n", ptc,c);
11      printf("b\t(%x)\t|%04x|\n", ptb,b);
12      printf("a\t(%x)\t|%08x|\n", pta,a);
```

— Exécution

```
c       (a9b9d359)      |0c|
b       (a9b9d35a)      |0bbb|
a       (a9b9d35c)      |00aaaaaa|
```

— Remarques :



int pointer

— Soit à présent les variables suivantes :

```
3       unsigned int a = 0xaaaaaa;
4       unsigned int b = 0xbbb;
5       unsigned int c = 0xc;
6       int *pta = &a;
7       int *ptb = &b;
8       int *ptc = &c;
```

— Affichage :

— Exécution

```
a     (990239e4)     |00aaaaaa|
b     (990239e8)     |00000bbb|
c     (990239ec)     |0000000c|
```

@ d'un pointeur :

```
13          printf("pta\t(%x)\t|%016x|\n", &pta, pta);
14          printf("ptb\t(%x)\t|%016x|\n", &ptb, ptb);
15          printf("ptc\t(%x)\t|%016x|\n", &ptc, ptc);
```

— Exécution

```
pta      (990239f0)       |00000000990239e4|
ptb      (990239f8)       |00000000990239e8|
ptc      (99023a00)       |00000000990239ec|
```

— Interprétations :

Application directe (I) :

— Donner un programme, nommé `Int2Bytes.c`, capable d'afficher en décimale et en hexadécimale un entier, puis de le découper sur 4 Octets et de les afficher en hexadécimale.

```
1  #include <stdio.h>
2  int main(){
3          unsigned int i, a=0xDDCCBBAA;
4          unsigned int* pta=&a;
5          unsigned char* ptc=(char*)&a;
6          printf("a\t(%x)\t|%08x|\n", pta,a);
7          printf("pta\t(%x)\t|%08x|\n", &pta,pta);
```

```
11          return 0;
12  }
```

— Execution :

```
a       (df919c60)      |ddccbbaa|
pta     (df919c68)      |df919c60|
ptc+0   (df919c60)      |aa|
ptc+1   (df919c61)      |bb|
ptc+2   (df919c62)      |cc|
ptc+3   (df919c63)      |dd|
```

4.2 Pointeurs et fonctions

Visibilité des variables

— Une variable globale est visible (accessible) par toutes les fonctions du programme.

— Une variable locale n'est visible (accessible) qu'à l'intérieur de la fonction qui l'a déclarée.

— Les arguments d'une fonction sont des variables locales ; toute modification de ces variables n'est pas visible ni par le programme principal ni par les autres fonctions.

— Il est préférable de transmettre les paramètres volumineux par pointeurs ⇒ gain dans la transmission :

Transmission de paramètre par pointeur

— Paramètre d'Entrée (`E:`) on utilise des variables simples ex : `int`, `float`... ;

— Paramètres de Sortie (`S:`) et d'(`E/S:`) : il faut utiliser des pointeurs sur des variables externes ;

— Exemples de passage d'argument :

par valeur :

```
13  void permutVal(int a,int b){
14      int c = a ;
15      a = b ;
16      b = c ;
```

```
17      printf("a=%d, b=%d\n",a,b);
18  }
```

par pointeur :

Comparaison : la fonction main

— Soit la fonction main suivante ;

```
1  #include <stdio.h>
2  void permutVal(int ,int );
3  void permutPt(int* ,int* );
4  int main(){
5      int x=-1, y=15; printf("x=%d, y=%d\n",x,y);
6
7      printf("*** Passage de paramètres par valeur ***\n");
8      permutVal(x,y); printf("x=%d, y=%d\n",x,y);
9      printf("*** Passage de paramètres par pointeur ***\n");
10     permutPt(&x,&y); printf("x=%d, y=%d\n",x,y);
11         return 0;
12 }
```

Comparaison : Exécution

— Exécution:

```
x=-1, y=15
*** Passage de paramètres par valeur ***
a=15, b=-1
x=-1, y=15
*** Passage de paramètres par pointeur ***
x=15, y=-1
```

— avec la fonction `permutVal` *la permutation s'est déroulée en locale* sur les variables `a` et `b`.

4.3 Pointeurs et tableaux

Pointeurs et tableaux

— Le nom d'un tableau est un pointeur *constant* sur la première case du tableau ;

`tab`	=	`&tab[0]`	→	`tab[0]`	=	`*tab`
`tab+1`	=	`&tab[1]`	→	`tab[1]`	=	`*(tab+1)`
`tab+i`	=	`&tab[i]`	→	`tab[i]`	=	`*(tab+i)`

— Exemple :

```c
#include <stdio.h>
int main(){
        int tab[3] = {0xa, 0x3, 0xd};
        int * ptb = tab+2;
```

Pointeurs et tableaux : Exemple

```c
        printf("ptb\t(%x)\t|%x|\n", &ptb, ptb);
        return 0;
}
```

```
tab     (a854b29c)    |0000000a|
tab+1   (a854b2a0)    |00000003|
tab+2   (a854b2a4)    |0000000d|
ptb     (a854b288)    |a854b2a4|
```

Exemple : Remplissage d'un tableau

— Remplir les cases du tableau `T` à partir du clavier.

— Renvoyer le nombre des cases saisies. Sachant que la saisie se termine par la valeur 0.

```c
int saisieTab (int T[]){
        int *pt = T;
```

```c
        return pt-T;
}
```

Exemple : Affichage d'un tableau

— Parcourir un tableau et d'afficher ses cases sur 6 chiffres.

— Sachant que la valeur 0 est considérée comme une sentinelle de fin du tableau.

```
1 void afficheTab (int* T){
2                 int *ptr = T;
```

}

Exemple : main.c

```
1 #include <stdio.h>
2 int main (){
3         int Tab[100], N;
4         printf("Saisie (0 pour arrêter):");
5         N = saisieTab (T);
6         printf("Vous avez saisi %d nombres.\n", N);
7         afficheTab(T);
8         return 0;
9 }
```

— Exemple d'exécution:

```
Saisie (0 pour arrêter): 8 23 4 55 -6 11 -14 13 57 88 0
Vous avez saisi 10 nombres.
8     23    4     55    -6    11    -14   13    57    88
```

4.4 Pointeurs et tableau multidimensionnel

Pointeurs et tableau multidimensionnel

— Soit le programme suivant :

```
#include <stdio.h>
int main(){
        int i,Mat[3][4]={{1, 2, 3, 4},
                         {11,12,13,14},
                         {21,22,23,24}};
        int * pt = (int*)Mat;
        printf("Mat\t(%x)\t|%d|\n", Mat, *pt);
        pt = (int*)Mat+1;
        printf("Mat+1\t(%x)\t|%d|\n", Mat+1, *pt);

        for(i = 0; i < 3 ; i++){
                for(pt = (int*)(Mat+i); pt < Mat+i+1 ; pt++)
                        printf("%d␣\t", *pt);
                printf("\n");
        }
        return 0;
}
```

Exécution et interprétation

— Exécution :

```
Mat     (8cf9290)       |1|
Mat+1   (8cf92a0)       |11|
1       2       3       4
11      12      13      14
21      22      23      24
```

— `Mat` peut être considéré comme un tableau de trois cases dont chacune est un tableau de quatre cases d'entiers.

— Ainsi, la taille de chaque ligne est égale à 16 Octets.

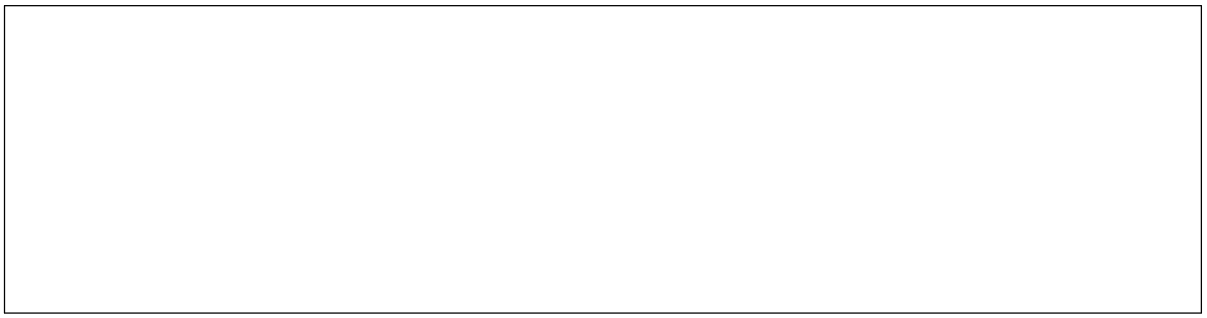
Mat	=	&Mat[0][0]	=	pt	→	Mat[0]=	1	2	3	4
Mat+1	=	&Mat[1][0]	=	pt+4	→	Mat[1]=	11	12	13	14
Mat+2	=	&Mat[2][0]	=	pt+2*4	→	Mat[2]=	21	22	23	24

Double pointeur

— Passage d'argument à fonction main :

```
#include <stdio.h>

int main(int argc, char **argv){
  int i;
  char** pt = argv;
  printf("␣argc␣=␣%d␣\n", argc);
```



```
  return 0;
}
```

Exécution et interprétation

— Exécution :

```
./ArgcArgv hello from CoursEmC
 argc = 4
argv = ((2ea44a38))
NumArg  (*pt)           |**pt|  :*pt
0       (2ea45117)      |.|     :./ArgcArgv
1       (2ea45122)      |h|     :hello
2       (2ea45128)      |f|     :from
3       (2ea4512d)      |C|     :CoursEmC
```

— `argv` peut être considéré comme un tableau de `argc` cases dont chacune est un tableau, de `char`, de taille différente.

`argv`	=	`&argv[0][0]`	→	`argv[0]=`	`./ArgcArgv`
`argv+1`	=	`&argv[1][0]`	→	`argv[1]=`	`Cours C embarqué`
`argv+2`	=	`&argv[2][0]`	→	`argv[2]=`	`ING1`