1. Définition d'un tableau

Un tableau est un ensemble d'éléments de même type. Ces éléments ont donc tous la même taille, et ils ont tous une adresse qui correspond au même type d'objet.

2. Tableaux à une dimension

2.1. Déclaration

La déclaration d'un tableau à une dimension réserve un espace de mémoire contiguë dans lequel les éléments du tableau peuvent être rangés. Comme le montre la figure 1, le nom du tableau seul est une constante dont la valeur est l'adresse du début du tableau. Les éléments sont accessibles par : le nom du tableau, un crochet ouvrant, l'indice de l'élément et un crochet fermant.

L'initialisation d'un tableau se fait en mettant une accolade ouvrante, la liste des valeurs servant à initialiser le tableau, et une accolade fermante. La figure montre l'espace mémoire correspondant à la définition d'un tableau de dix entiers avec une initialisation selon la ligne :

```
int tab[10] = {9,8,7,6,5,4,3,2,1,0};
```

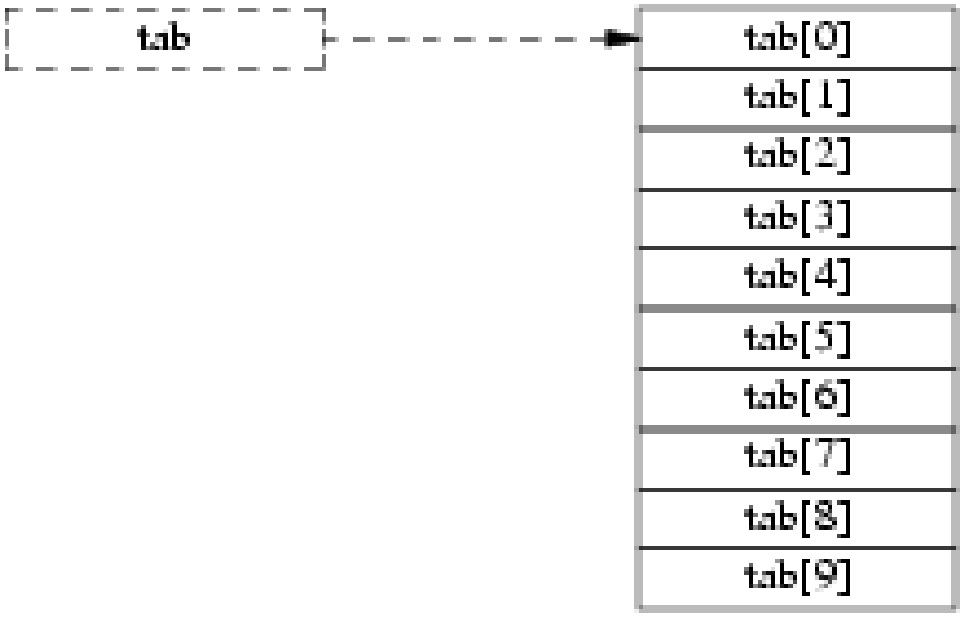


Figure 1: Un tableau de 10 entiers

2.3. Initialisation / lecture / affichage d'un tableau

Exemple 1: Lecture

```
int main(int argc, char *argv[])
{
    long tab1[4] = {0, 1, 2, 3}, tab2[6],i = 0;

    for (i = 0 ; i < 6 ; i++)
    {
        scanf("%ld\n", tab2[i]);
    }

    return 0; }
```

Exemple 2: Initialisation/ Affichage

```
int main(int argc, char *argv[])
{
    long tableau[4], i = 0;

    // Initialisation du tableau
    for (i = 0 ; i < 4 ; i++)
    {
        tableau[i] = 0;
    }

    // Affichage de ses valeurs pour vérifier
    for (i = 0 ; i < 4 ; i++)
    {
        printf("%ld\n", tableau[i]);
    }

    return 0;
}
```

Exercice 1

Ecrire un programme C qui permet de:

- Créer un tableau de 10 éléments de type entier
- Lire les éléments de ce tableau
- Faire la somme et le produit de ces éléments
- Impression des résultats

Correction de l'exercice

```
#include<stdio.h>
#define N 10
int main(void)
{
  int i,tab[N];
  float moy, prod;
  moy=0;
  printf("Entrer les valeurs du tableau \n");
```

```
  for (i=0;i<N;i++)
  {
          printf("tab[%d] = ",i);
          scanf("%d",&tab[i]);
          printf("\n");
          moy=moy+tab[i];
           prod=prod+tab[i]
  }
  moy=moy/N;
  printf("La moyenne du tableau vaut %f\n",moy);
printf("Le produit du tableau vaut %f\n",prod);

  return 0;
}
```

Exercice 2

Un terme d'un polynôme de degré 3 peut être représenté par un tableau de 8 éléments ou deux éléments consécutifs représentent un coefficient et un exposant.

En utilisant un tableau de termes pour représenter un polynôme, écrire un programme qui lit un polynôme P et calcule sa valeur P(X) pour une valeur X donnée.

Exemple :

Le polynôme $P(X) = 5X^3+X^2-3\ X^1-7$ est représenté par :

5	3	1	2	-3	1	-7	0

Pour X=1 P(X)= -4

3. Tableaux bidimensionnels, matrices

Une matrice à deux dimensions, est représentée sous la forme d'un tableau bidimensionnel que l'on déclare comme ci-dessous:

Par exemple si les éléments du tableau sont de type réels.

```
float tab[2][2];
```

L'élément d'une matrice tab_{ij} est désigné en lanage C par `tab[i][j]`. Par convention i désigne la ligne et j la colonne.

On peut initialiser un tel tableau au moment de sa déclaration grâce à la construction suivante:

```
int tab[2][2] = {{3, 4},{0, 2} };
```

3.1. Accès aux éléments d'un tableau à deux dimensions

Les tableaux à deux dimensions sont en tout point semblables aux tableaux à une dimension évoqués précédemment. Comme, il faut utilisez deux indices un pour la ligne et un autre pour la colonne. l'indice de la première ligne est 0 et l'indice de la première colonne est aussi 0.

l'accès à l'élément (première ligne , première colonne) est `tab[0][0]`
l'accès à l'élément (première ligne , deuxième colonne) est `tab[0][1]`
ainsi de suite;

l'accès à l'élément (deuxième ligne , première colonne) est `tab[1][0]`
l'accès à l'élément (deuxième ligne , deuxième colonne) est `tab[1][1]`

ainsi de suite....

Exemple
Donnez l'affichage de ce programme:

```
#include <stdio.h>
void main()
{
int i,j;
int m[4][4];
for(i=0;i<4;i++)
for(j=0;j<4;j++)
m[i][j] = i + j;


for(i=0;i<4;i++) {
for(j=0;j<4;j++)
printf("%d ",m[i][j]);
printf("\n");
}
}
```

```
0  1  2  3
1  2  3  4
2  3  4  5
3  4  5  6
```

Remarque
Comme pour les tableaux à une dimension le nom symbolique associé au tableau a en fait pour valeur l'adresse du premier élément du tableau `tab[0][0].`

La figure 2 donne les adresses et les noms des différents éléments constitués par la définition du tableau à deux dimensions suivant : `int tab[8][5];`

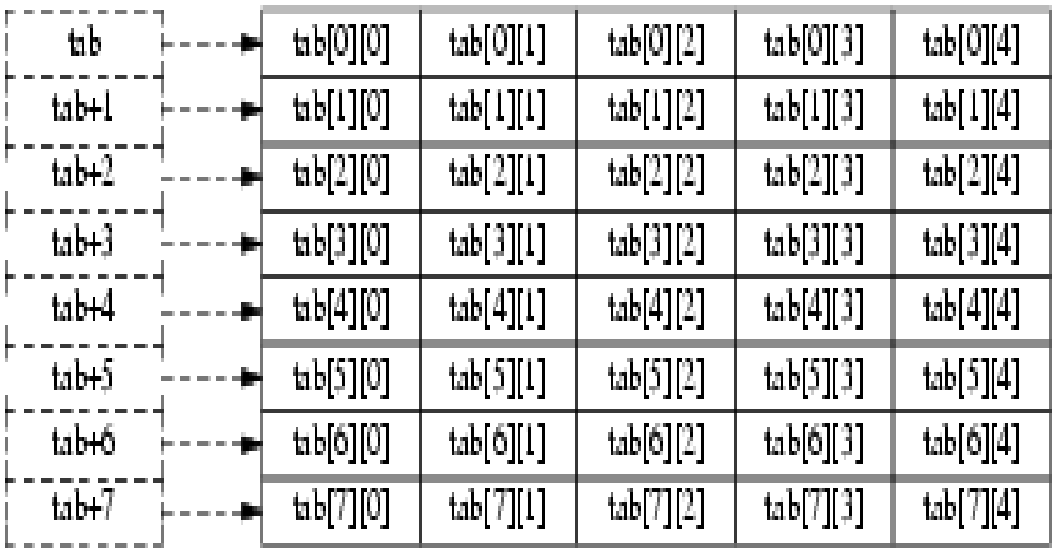


Figure 2. Tableau à deux dimensions

Exercice 1: Programme du triangle de Pascal

```
      1
      1  1
      1  2  1
      1  3  3  1
      1  4  6  4  1
      1  5 10 10  5  1
            etc.
```

```
#include<stdio.h>
#define TM 100
int main(void)
{  int i,n,j;
   int tab[TM][TM];
   printf("Calculons C(n,p) a l'aide du triangle de pascal\n");
   printf("Entrez la valeur de n  ");
   scanf("%d",&n);
   tab[0][0]=1;
   for(i=1;i<=n;i++)
   {
           for(j=0;j<i;j++)
           {
                   if(j==0)
                           tab[i][j]=1;
                   else
                           tab[i][j]=tab[i-1][j-1]+tab[i-1][j];
           }
           tab[i][j]=1;
   }

   printf("\n Le triangle de Pascal pour n=%d est \n",n);
   for(i=0;i<=n;i++)
   {
           for(j=0;j<=i;j++)
```

```
            {
                    printf("%d ",tab[i][j]);
            }
            printf("\n");
  }
   printf("\n");
   return(0);
}
```

Exercice 2: Produit de deux matrices

Ecrire une programme qui calcule le produit *C* d'une matrice *A* ayant *nla* lignes et *nca* colonnes par une matrice *B* ayant *nlb* lignes et *ncb* colonnes. On rappelle que le produit $A_{nla,nca} \times B_{nlb,ncb}$ est une matrice $C_{nla,ncb}$ définie par :

$$C_{i,j} = \sum_{k=0}^{nca-1} A_{i,k} \times B_{k,j}$$

Exercice 3: Transposée d'une matrice

On demande d'écrire un programme qui initialise une matrice M avec les valeurs M[i][j] = 3i + j, et affiche la matrice. On demande ensuite de transposer la matrice, c'est-à-dire d'échanger les éléments M[i][j] et M[j][i], pour toutes les valeurs de i et de j.

Exercice 4: Matrice carrée

Soit Mat (n, m) matrice carrée d'entier. Ecrire un programme c qui permet de :

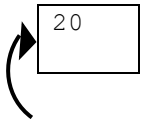
1- calculer la somme des éléments de la première diagonale.
2- calculer le produit des éléments de la deuxième diagonale.

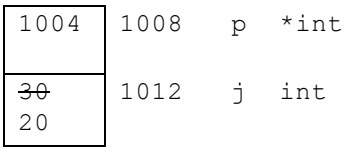
4. Pointeurs

Le pointeur est une variable destinée à contenir une adresse mémoire. Elle indique (pointe) l'emplacement d'une autre variable dans la mémoire. Il est reconnu syntaxiquement par * lors de sa déclaration **int *p;**

Supposons que i se trouve à l'adresse 1004 en mémoire et contient la valeur 20. Le langage C dispose d'un opérateur pour prendre l'adresse d'une variable, l'opérateur unaire **&**. l'instruction **p=&i;** met dans la variable p la valeur 1004, adresse de l'emplacement mémoire correspondant à la variable i. On dit alors que **p pointe sur i** ou que p est un pointeur sur i.

Exemple





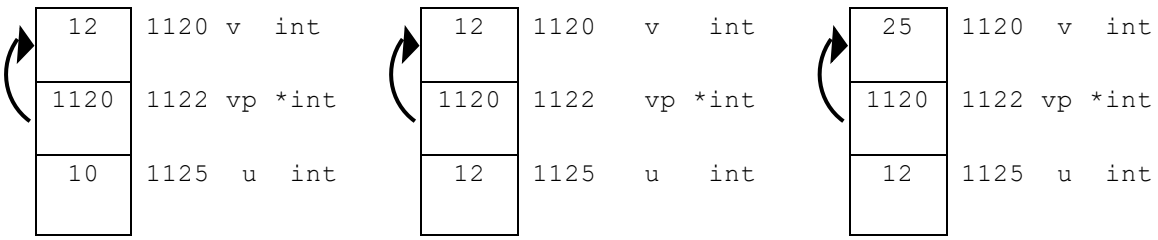
Le langage C offre également un opérateur permettant d'accéder à la valeur stockée dans une case mémoire dont on connaît l'adresse, il s'agit de l'opérateur unaire *****. Dans notre exemple, ***p** a pour valeur l'entier stocké à l'adresse 1004, c'est-à-dire 20. Ainsi, si j est de type int, la construction suivante: **j = *p**; met la valeur 20 dans la variable j.

Considérons le programme suivant qui résume ces notions:

```
void main()
{

int v=12;
int u=10;
int *vP; /*pointeur sur int */
vP = &v; /*affectation du pointeur */
u = *vP;
printf("u=%d v=%d\n",u,v);
*vP = 25;
printf("u=%d v=%d\n",u,v);

}
```



5. Pointeurs et tableaux

1- La valeur d'une variable de type tableau est l'adresse du premier élément du tableau. Comme tableau est un pointeur, on peut utiliser le symbole * pour connaître la première valeur.

Exemple

```
int tab[10];
printf("%d", *tab);
```

2- Il est aussi possible d'avoir la valeur de la seconde case en tapant *(tab+ 1) (adresse de tab + 1). Les 2 lignes suivantes sont donc identiques :

`tab[1]` // Renvoie la valeur contenue dans la seconde case (la première case étant 0)
`*(tab+ 1)` // Identique : renvoie la valeur contenue dans la seconde case

Donc, quand vous écrivez tab[0], vous demandez la valeur qui se trouve à l'adresse tab + 0 case (c'est-à-dire 9). Si vous écrivez tab[1], vous demandez la valeur se trouvant à l'adresse tab+ 1 case (c'est-à-dire 8). Et ainsi de suite pour les autres valeurs

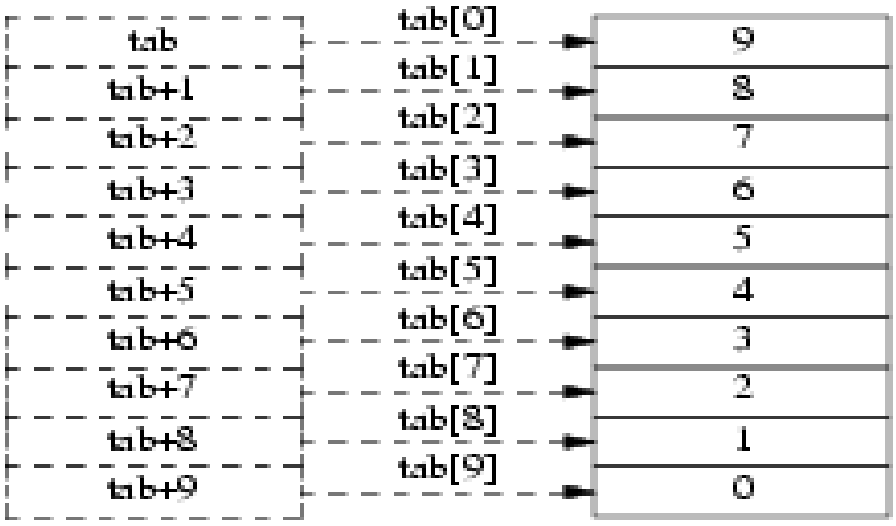


Figure 3: Un tableau à une dimension en mémoire

Le langage C associe au nom tab l'adresse où commence le tableau tab en mémoire.
tab est le premier élément, tab+1 est le deuxième, tab+2 est le troisième et ainsi de suite...

5.1. Accès à un tableau à une dimension avec un pointeur

La figure 4 est un exemple dans lequel sont décrites les différentes façons d'accéder au éléments d'un tableau `tab` et du pointeur `pt` après la définition suivante :

```
int tab[8];
int *pt = tab;
```

Après son exécution la mémoire se trouve dans l'état suivant:

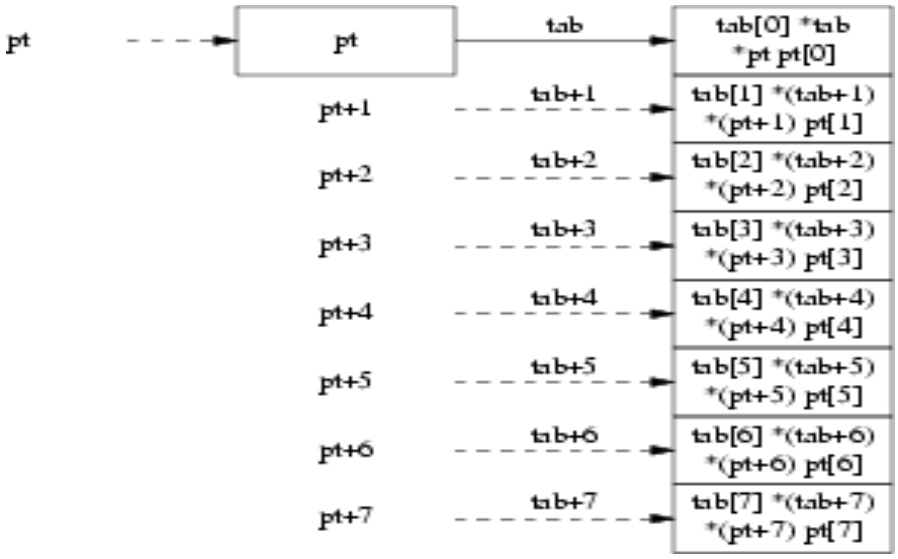


Figure 4. Différentes façons d'accéder au éléments d'un tableau à une dimension

5.2. Accès à un tableau à deux dimensions avec un pointeur

La figure 5 décrit les différentes façons d'accéder aux variables d'un tableau de six fois cinq entiers (certains diraient un tableau de 6 lignes et 5 colonnes) à partir d'un pointeur qui peut contenir une adresse de sous-tableau (de cinq entiers).

```
int tab[6][5]; tableau de 6 fois 5 entiers.
int (*pt)[5]= tab;
```

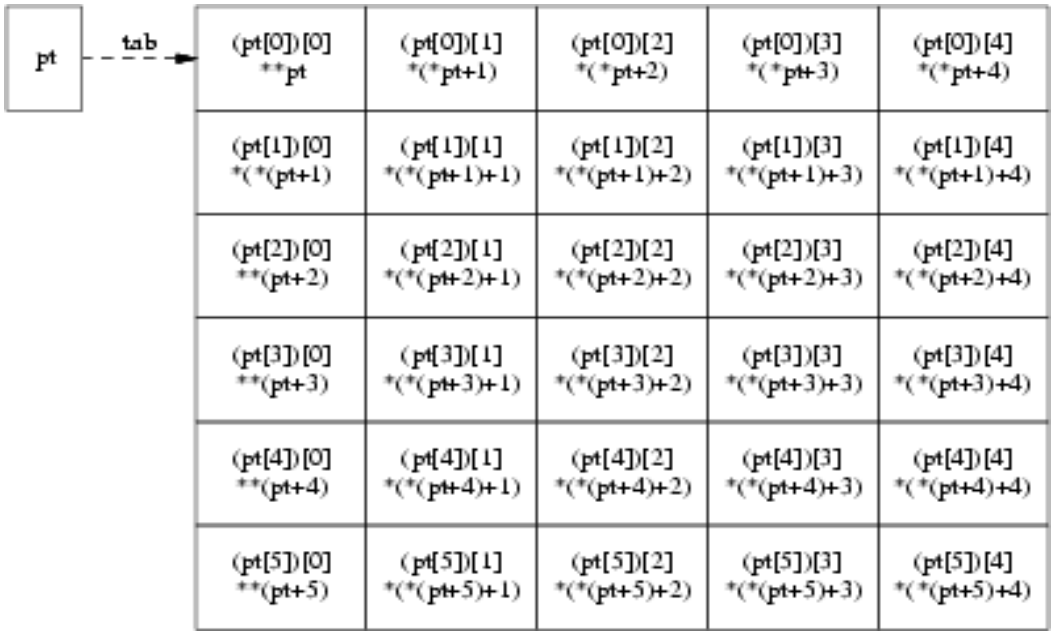


Figure 5. Accès à un tableau à deux dimensions avec un pointeur

Le programme ci dessus montre les différentes manières d'accéder aux éléments du tableau défini ci-dessus par le pointeur associé.

Programme

```
#include <stdio.h>

 int
 main (int argc, char *argv[])
 {
 int tab[6][5];
 int (*pt)[5]= tab;
 int i,j;

 for(i=0;i<6;i++)
 for(j=0;j<5;j++)
 tab[i][j]= i*10+j;

 for(i=2;i<6;i++)
 for(j=3;j<5;j++){
 printf("tab[%d][%d] = %d\t",i,j,tab[i][j]);
 printf("%d\t",*(tab[i]+j));
 printf("%d\t",*(*(tab+i)+j));
 printf("%d\t",*(*(pt+i)+j));
 printf("%d\t",*(pt[i]+j));
 printf("%d\n",pt[i][j]);
 }
 return 0;
 }
```

Données écrites sur le fichier standard de sortie.

```
tab[2][3] = 23 23 23 23 23 23
tab[2][4] = 24 24 24 24 24 24
tab[3][3] = 33 33 33 33 33 33
tab[3][4] = 34 34 34 34 34 34
tab[4][3] = 43 43 43 43 43 43
tab[4][4] = 44 44 44 44 44 44
tab[5][3] = 53 53 53 53 53 53
tab[5][4] = 54 54 54 54 54 54
```

5.3. Arithmétique des pointeurs

Les pointeurs étant des variables contenant des adresses numériques donc en fait des nombres entiers, on peut leur appliquer des opérations arithmétiques, notamment incrémentation et décrémentation. Toutefois il y a quelques subtilités de taille qui distinguent l'arithmétique des pointeurs de celle des simples entiers. Ceci nous amène à regarder l'utilisation de pointeurs pour manipuler des tableaux, en prenant les variables

Considérons le programme suivant:

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

```
main()
{
int tab[10]= {9,8,7,6,5,4,3,2,1,0};
int *pt1, *pt2;
pt1 = tab; // initialise le pointeur pt1 avec l'adresse du début de
tableau.
pt2 = tab+1; // initialise le pointeur pt2 avec l'adresse du 2ème
élément du début de tableau  tab
printf("%d %d\n",*pt1,*pt2);
}
```

`pt1 += 1 ; //` fait avancer, le pointeur `pt1` d'une case ce qui fait qu'il contient l'adresse du 2ème élément du tableau. donc: `pt1` est équivalent à `pt2` maintenant:

Remarque 1

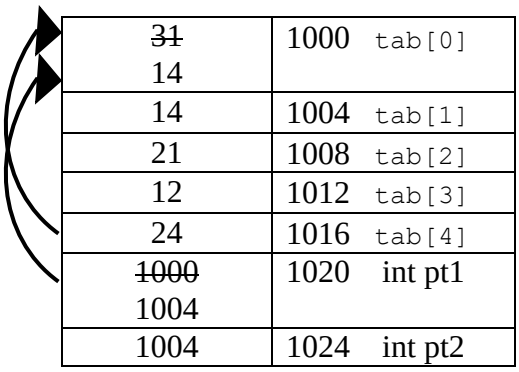
Quel que soit le type pointé, si pt est un pointeur sur un élément d'un tableau, alors pt+1 est un pointeur sur l'élément suivant, pt+i est un pointeur sur le ième élément suivant.

Remarque 2

Il y a une différence fondamentale entre les deux expressions pt1=pt2 et *pt1=*pt2.
Considérons le programme suivant:

```
#include <stdio.h>
main()
{
int tab[5] = {31, 14, 21, 12, 24};
int *pt1, *pt2;
pt1 = tab;
pt2 = tab+1;
pt1=pt2;
*pt1=*pt2;
}
```

La configuration résultant de ces deux expressions est illustrée dans la figure suivante:



~~31~~ 14	1000 tab[0]
14	1004 tab[1]
21	1008 tab[2]
12	1012 tab[3]
24	1016 tab[4]
~~1000~~ 1004	1020 int pt1
1004	1024 int pt2

La première expression pt1=pt2; fait passer la valeur de pt1 de 1000 à 1004 qui est la valeur de pt2. pt1 se retrouve ainsi pointant sur le même emplacement mémoire que pt2.
La deuxième expression, *pt1=*pt2; fait passer la valeur pointée par pt1 de 31 à 14 qui est la valeur pointée par pt2.