Cours Programmation I (chapitres 5&6)

Licence Fondamentale SMI (semestre 3)

Pr. Mouad BEN MAMOUN

ben_mamoun@fsr.ac.ma

Année universitaire 2014/2015

Chapitre 5

Les tableaux

Exemple introductif

- Supposons qu'on veut conserver les notes d'une classe de 30 étudiants pour extraire quelques informations. Par exemple : calcul du nombre d'étudiants ayant une note supérieure à 10
- Le seul moyen dont nous disposons actuellement consiste à déclarer 30 variables, par exemple **N1, …, N30**. Après 30 instructions de saisie, on doit écrire 30 instructions if pour faire le calcul

```
nbre = 0
if (N1  >10)  nbre++ ;
….
if (N30>10)  nbre++ ;
```

c'est lourd à écrire

- Heureusement, les langages de programmation offrent la possibilité de rassembler toutes ces variables dans **une seule structure de donnée** appelée **tableau**

Tableaux

- Un **tableau** est une variable structurée composée d'un nombre de variables simples de même type désignées par un seul identificateur
- Ces variables simples sont appelées *éléments ou composantes* du tableau, elles sont stockées en mémoire à des emplacements contigus (l'un après l'autre)
- Le type des éléments du tableau peut être :
  - simple : char, int, float, double, …
  - pointeur ou structure
- On peut définir des tableaux :
  - à une dimension (tableau unidimensionnel ou vecteur)
  - à plusieurs dimensions (tableau multidimensionnel)

Déclaration des tableaux

- La déclaration d'un tableau à une dimension s'effectue en précisant le type de ses éléments et sa dimension (le nombre de ses éléments) :
  - Syntaxe en C : **Type identificateur[dimension];**
  - Exemple : **float notes[30];**
- La déclaration d'un tableau permet de lui réserver un espace mémoire dont la taille (en octets) est égal à : dimension * taille du type
- ainsi pour :
  - short A[100]; // on réserve 200 octets (100* 2octets)
  - char mot[10]; // on réserve 10 octets (10* 1octet)

Initialisation à la déclaration

- On peut initialiser les éléments d'un tableau lors de la déclaration, en indiquant la liste des valeurs respectives entre accolades. Ex:
  - int A[5] = {1, 2, 3, 4, 5};
  - float B[4] = {-1.5, 3.3, 7e-2, -2.5E3};

- Si la liste ne contient pas assez de valeurs pour toutes les composantes, les composantes restantes sont initialisées par zéro
  - Ex: short T[10] = {1, 2, 3, 4, 5};

- la liste ne doit pas contenir plus de valeurs que la dimension du tableau. Ex: short T[3] = {1, 2, 3, 4, 5};➔ Erreur

- Il est possible de ne pas indiquer la dimension explicitement lors de l'initialisation. Dans ce cas elle est égale au nombre de valeurs de la liste. Ex: short T[ ] = {1, 2, 3, 4, 5};➔ tableau de 5 éléments

Accès aux composantes d'un tableau

- L'accès à un élément du tableau se fait au moyen de l'indice. Par exemple, **T[i]** donne la valeur de l'élément i du tableau T

- En langage C l'indice du premier élément du tableau est 0. L'indice du dernier élément est égal à la dimension-1

Ex: int T[5] = {9, 8, 7, 6, 5};➔

T[0]=9, T[1]=8, T[2]=7, T[3]=6, T[4]=5

Remarques:

- on ne peut pas saisir, afficher ou traiter un tableau en entier, ainsi on ne peut pas écrire printf(" %d",T) ou scanf(" %d",&T)

- On traite les tableaux élément par élément de façon répétitive en utilisant des boucles

Tableaux : saisie et affichage

- Saisie des éléments d'un tableau T d'entiers de taille n :

```
for(i=0;i<n;i++)
      { printf ("Entrez T[%d ] \n ",i );
        scanf(" %d" , &T[i]);
      }
```

- Affichage des éléments d'un tableau T de taille n :

```
for(i=0;i<n;i++)
      printf (" %d \t",T[i]);
```

Tableaux : exemple

- Calcul du nombre d'étudiants ayant une note supérieure à 10 :

```
main ( )
    {       float notes[30];
             int nbre,i;
            for(i=0;i<30;i++)
                 { printf ("Entrez notes[%d] \n ",i);
                   scanf(" %f" , &notes[i]);
                 }
               nbre=0;
               for (i=0; i<30; i++)
                        if (notes[i]>10) nbre++;
               printf (" le nombre de notes > à 10 est égal à : %d", nbre);

    }
```

Exercice

- Ecrire un programme qui permet de saisir un tableau de 10 réels et une valeur donnée et d'afficher si cette valeur se trouve dans le tableau ou non.

Tableaux à plusieurs dimensions

On peut définir un tableau à n dimensions de la façon suivante:

- **Type Nom_du_Tableau[D1][D2]…[Dn];** où Di est le nombre d'éléments dans la dimension i

- **Exemple :** pour stocker les notes de 20 étudiants en 5 modules dans deux examens, on peut déclarer un tableau :

float notes[20][5][2];

(notes[i][j][k] est la note de l'examen k dans le module j pour l'étudiant i)

Tableaux à deux dimensions (Matrices)

- Syntaxe : Type nom_du_Tableau[nombre_lignes][nombre_colonnes];
- Ex: **short A[2][3];** On peut représenter le tableau A de la manière suivante :

A[0][0]	A[0][1]	A[0][2]
A[1][0]	A[1][1]	A[1][2]

- Un tableau à deux dimensions A[n][m] est à interpréter comme un tableau unidimensionnel de dimension n dont chaque composante A[i] est un tableau unidimensionnel de dimension m.
- Un tableau à deux dimensions A[n][m] contient n***** m composantes. Ainsi lors de la déclaration, on lui réserve un espace mémoire dont la taille (en octets) est égal à : n*m* taille du type

Initialisation à la déclaration d'une Matrice

- L'initialisation lors de la déclaration se fait en indiquant la liste des valeurs respectives entre accolades ligne par ligne

- Exemple :
  - **float A[3][4] = {{-1.5, 2.1, 3.4, 0}, {8e-3, 7e-5,1, 2.7 }, {3.1, 0, 2.5E4, -1.3E2}};**

    A[0][0]=-1.5 , A[0][1]=2.1, A[0][2]=3.4, A[0][3]=0
    A[1][0]=8e-3 , A[1][1]=7e-5, A[1][2]=1, A[1][3]=2.7
    A[2][0]=3.1 , A[2][1]=0, A[2][2]=2.5E4, A[2][3]=-1.3E2

- On peut ne pas indiquer toutes les valeurs: Les composantes manquantes seront initialisées par zéro

- Comme pour les tableaux unidimensionnels, Il est défendu d'indiquer trop de valeurs pour une matrice

Matrices : saisie et affichage

- Saisie des éléments d'une matrice d'entiers A[n][m] :

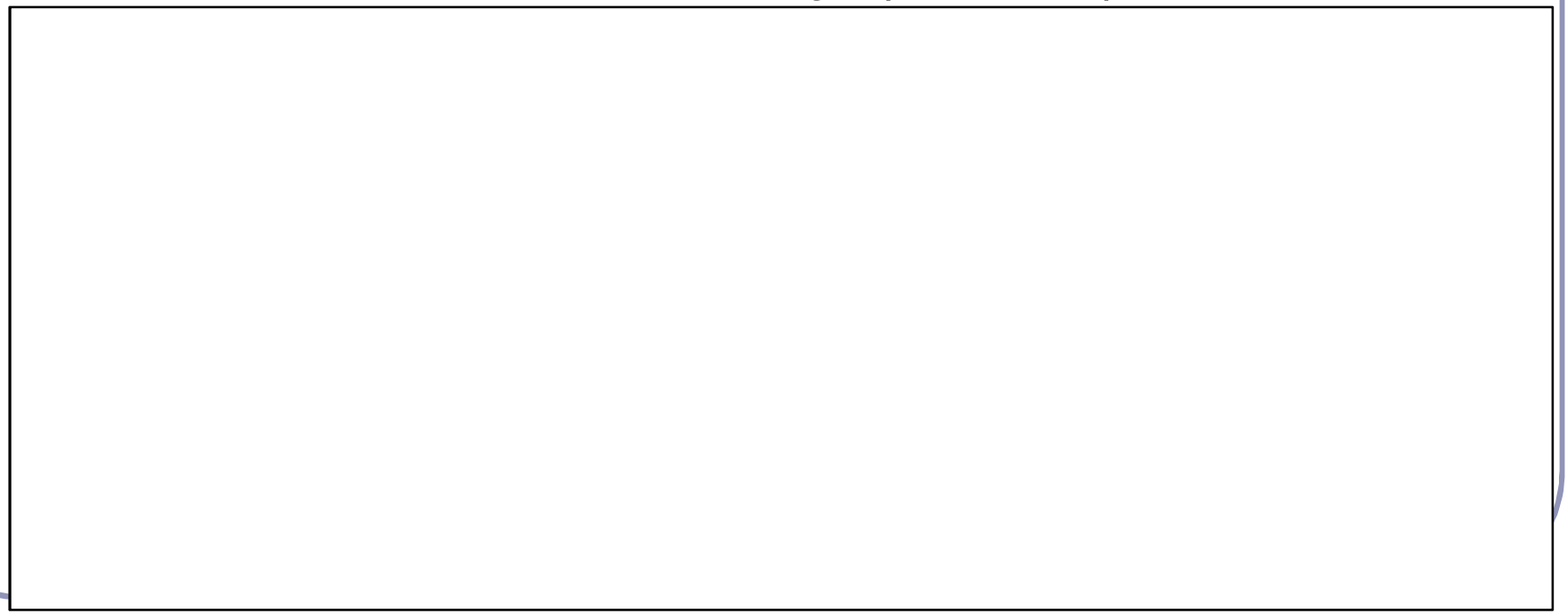
```
for(i=0; i<n; i++)
      for(j=0; j<m; j++)

    { printf ("Entrez la valeur de A[%d][%d] \n ",i,j);
      scanf(" %d" , &A[i][j]);
    }
```

- Affichage des éléments d'une matrice d'entiers A[n][m] :

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

Exercice

- Ecrire un programme qui permet de :
  - Saisir une matrice de réels de 10 lignes et 20 colonnes et une valeur réelle x
  - Afficher l'indice de la première colonne qui contient x s'il se trouve dans la matrice, sinon le message que x n'est pas dans la matrice



Exercice

Ecrire un programme qui transfère les éléments d'une matrice d'entiers A[n][m] (dimensions maximales: 10 lignes et 10 colonnes) dans un tableau T à une dimension n*m.

Exemple

$$\begin{pmatrix} 2 & 0 & 1 \\ 7 & 3 & 5 \\ 9 & 1 & 4 \\ 8 & 5 & 0 \end{pmatrix} \Rightarrow \begin{pmatrix} 2 & 0 & 1 & 7 & 3 & 5 & 9 & 1 & 4 & 8 & 5 & 0 \end{pmatrix}$$

Représentation d'un tableau en mémoire

- La déclaration d'un tableau provoque la réservation automatique par le compilateur d'une zone contiguë de la mémoire.

- La mémoire est une succession de cases mémoires. Chaque case est une suite de 8 bits (1 octet), identifiée par un numéro appelé **adresse**.
  (on peut voir la mémoire comme une armoire constituée de tiroirs numérotés. Un numéro de tiroir correspond à une adresse)

- Les adresses sont souvent exprimées en hexadécimal pour une écriture plus compacte et proche de la représentation binaire de l'adresse. Le nombre de bits d'adressage dépend des machines.

- En C, **l'opérateur &** désigne **adresse de**. Ainsi, printf(" adresse de a= %x ", &a) affiche l'adresse de la variable a en hexadécimal

Représentation d'un tableau à une dimension en mémoire

- En C, le nom d'un tableau est le représentant de l'adresse du premier élément du tableau (pour un tableau T: **T=&T[0]**)

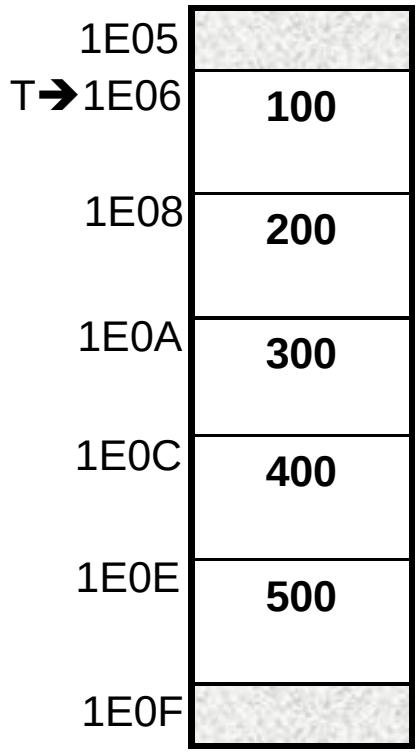
- Les composantes du tableau étant stockées en mémoire à des emplacements contigus, les adresses des autres composantes sont calculées (automatiquement) relativement à cette adresse :

  &T[i]= &T[0]+sizeof(type)*i

- Exemple : **short T[5] = {100, 200, 300, 400, 500};** et supposons que **T=&T[0]** =1E06

- On peut afficher et vérifier les adresses du tableau:

```
for(i=0;i<5;i++)
printf("adresse de T[%d]= %x\n",i,&T[i]);
```



Représentation d'un tableau à deux dimensions en mémoire

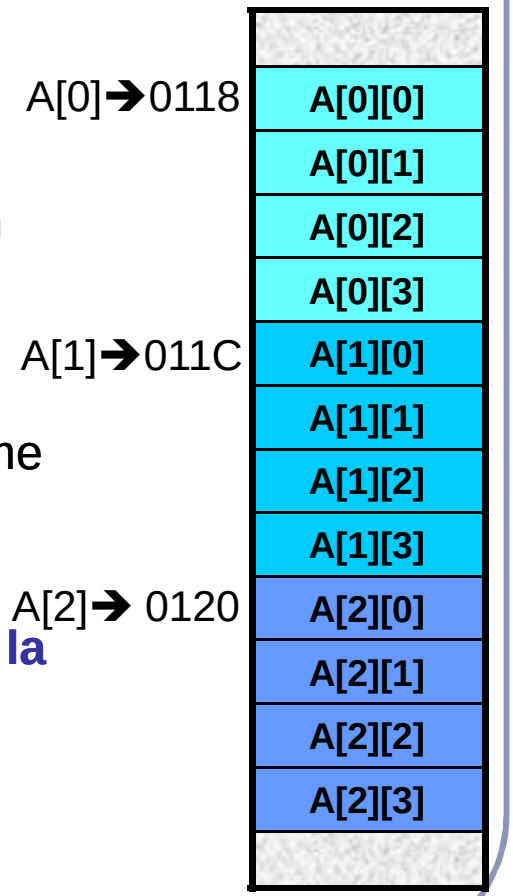
- Les éléments d'un tableau sont stockées en mémoire à des emplacements contigus ligne après ligne

- Comme pour les tableaux unidimensionnels, le nom d'un tableau A à deux dimensions est le représentant de l'adresse du premier élément : **A=&A[0][0]**

- Rappelons qu'une matrice A[n][m] est à interpréter comme un tableau de dimension n dont chaque composante A[i] est un tableau de dimension m.

A[i] et &A[i][0] représentent l'adresse du 1er élément de la ligne i (pour i de 0 à n-1)

- Exemple : **char A[3][4];** A=&A[0][0] =0118

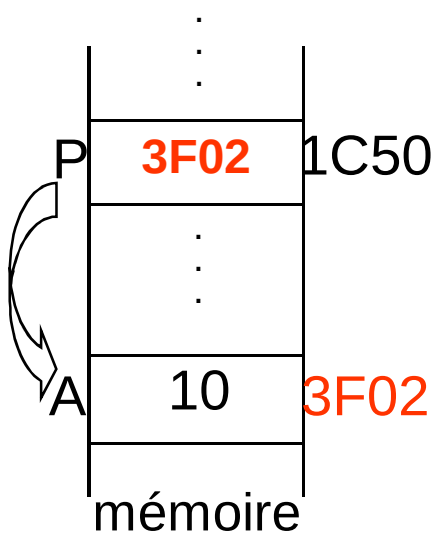


Chapitre 6

Les pointeurs

Pointeurs : définition

- Un pointeur est une variable spéciale qui peut contenir l'adresse d'une autre variable.
- Exemple : Soit A une variable contenant la valeur 10 et P un pointeur qui contient l'adresse de A *(on dit que P pointe sur A)* .
- Remarques :
  - Le nom d'une variable permet d'accéder *directement* à sa valeur (**adressage direct**).
  - Un pointeur qui contient l'adresse de la variable, permet d'accéder *indirectement* à sa valeur (**adressage indirect**).
  - Le nom d'une variable est lié à la même adresse, alors qu'un pointeur peut pointer sur différentes adresses



Intérêts des pointeurs

- Les pointeurs présentent de nombreux avantages :
  - Ils sont indispensables pour permettre le passage par référence pour les paramètres des fonctions
  - Ils permettent de créer des structures de données (listes et arbres) dont le nombre d'éléments peut évoluer dynamiquement. Ces structures sont très utilisées en programmation.
  - Ils permettent d'écrire des programmes plus compacts et efficaces

Déclaration d'un pointeur

- En C, chaque pointeur est limité à un type de donnée (même si la valeur d'un pointeur, qui est une adresse, est toujours un entier).
- Le type d'un pointeur dépend du type de la variable pointée. Ceci est important pour connaître la taille de la valeur pointée.
- On déclare un pointeur par l'instruction : **type *nom-du-pointeur ;**
  - type est le type de la variable pointée
  - ***** est l'opérateur qui indiquera au compilateur que c'est un pointeur
  - Exemple :
    int *pi; //pi est un pointeur vers une variable de type int
    float *pf; //pf est un pointeur vers une variable de type float
- Rq: la valeur d'un pointeur donne l'adresse du premier octet parmi les n octets où la variable est stockée

Opérateurs de manipulation des pointeurs

- Lors du travail avec des pointeurs, nous utilisons :
  - un **opérateur 'adresse de': &** pour obtenir l'adresse d'une variable
  - un **opérateur 'contenu de': *** pour accéder au contenu d'une adresse

- Exemple1 :

  - **int * p;** //on déclare un pointeur vers une variable de type int
  - **int i=10, j=30;** // deux variables de type int
  - **p=&i;** // on met dans p, l'adresse de i (p pointe sur i)
  - **printf("*p = %d \n",*p);** //affiche : *p = 10
  - ***p=20;** // met la valeur 20 dans la case mémoire pointée par p (i vaut 20 après cette instruction)
  - **p=&j;** // p pointe sur j
  - **i=*p;** // on affecte le contenu de p à i (i vaut 30 après cette instruction)

Opérateurs de manipulation des pointeurs

- Exemple2 : **float a, *p;**
  p=&a;
  printf("Entrez une valeur : \n");
  scanf("%f ",p); //supposons qu'on saisit la valeur 1.5
  printf("Adresse de a= %x, contenu de a= %f\n" , p,*p);
  ***p+=0.5;**
  printf ("a= %f\n" , a); //affiche a=2.0

- **Remarque :** si un pointeur P pointe sur une variable X, alors *P peut être utilisé partout où on peut écrire X
  - X+=2 équivaut à *P+=2
  - ++X équivaut à ++ *P
  - X++ équivaut à (*P)++ // les parenthèses ici sont obligatoires car l'associativité des opérateurs unaires * et ++ est de droite à gauche

Initialisation d'un pointeur

- A la déclaration d'un pointeur p, on ne sait pas sur quel zone mémoire il pointe. Ceci peut générer des problèmes :
  - **int *p;**

    ***p = 10;** //provoque un problème mémoire car le pointeur p n'a pas été initialisé

- **Conseil :** Toute utilisation d'un pointeur doit être précédée par une initialisation.
- On peut initialiser un pointeur en lui affectant :
  - l'adresse d'une variable (Ex: **int a, *p1; p1=&a;**)
  - un autre pointeur déjà initialisé (Ex: **int *p2; p2=p1;**)
  - la valeur 0 désignée par le symbole NULL, défini dans <stddef.h>.

    Ex: **int *p; p=0;ou p=NULL;** (on dit que p pointe 'nulle part': aucune adresse mémoire ne lui est associé)
- Rq: un pointeur peut aussi être initialisé par une allocation dynamique (voir fin du chapitre)

Pointeurs : exercice

```
main()
 {  int A = 1, B = 2, C = 3, *P1, *P2;
    P1=&A;
    P2=&C;
    *P1=(*P2)++;
     P1=P2;
     P2=&B;
    *P1-=*P2;
     ++*P2;
    *P1*=*P2;
     A=++*P2**P1;
     P1=&A;
    *P2=*P1/=*P2;
 }
```

Donnez les valeurs de A, B,C,P1 et P2 après chaque instruction

Opérations arithmétiques avec les pointeurs

- La valeur d'un pointeur étant un entier, certaines opérations arithmétiques sont possibles : ajouter ou soustraire un entier à un pointeur ou faire la différence de deux pointeurs

- Pour un entier i et des pointeurs p, p1 et p2 sur une variable de type T
  - **p+i (resp p-i)** : désigne un pointeur sur une variable de type T. Sa valeur est égale à celle de p incrémentée (resp décrémentée) de i*sizeof(T).

  - **p1-p2 :** Le résultat est un entier dont la valeur est égale à (différence des adresses)/sizeof(T).
- Remarque:
  - on peut également utiliser les opérateurs ++ et -- avec les pointeurs
  - la somme de deux pointeurs n'est pas autorisée

Opérations arithmétiques avec les pointeurs

Exemple :

```
float *p1, *p2;
float z =1.5;
p1=&z;
printf("Adresse p1 = %x \n",p1);
p1++;
p2=p1+1;
printf("Adresse p1 = %x \t Adresse p2 = %x\n",p1,p2);
printf("p2-p1 = %d \n",p2-p1);
```

Affichage :

Adresse p1 = 22ff44

Adresse p1 = 22ff48 Adresse p2 = 22ff4c

p2-p1=1

Pointeurs et tableaux

- Comme on l'a déjà vu au chapitre 5, le nom d'un tableau T représente l'adresse de son premier élément (T=&T[0]). Avec le formalisme pointeur, on peut dire que T est un **pointeur constant** sur le premier élément du tableau.

- En déclarant un tableau T et un pointeur P du même type, l'instruction P=T fait pointer P sur le premier élément de T (P=&T[0]) et crée une liaison entre P et le tableau T.

- A partir de là, on peut manipuler le tableau T en utilisant P, en effet :

  - **P pointe sur T[0] et *P désigne T[0]**
  - **P+1 pointe sur T[1] et *(P+1) désigne T[1]**
  - **….**
  - **P+i pointe sur T[i] et *(P+i) désigne T[i]**

Pointeurs et tableaux : exemple

- Exemple: **short x, A[7]={5,0,9,2,1,3,8};**
  short *P;
  P=A;
  x=*(P+5);

- Le compilateur obtient l'adresse P+5 en ajoutant 5*sizeof(short) = 10 octets à l'adresse dans P

- D'autre part, les composantes du tableau sont stockées à des emplacements contigus et &A[5]= &A[0]+sizeof(short)*5=A+10

- Ainsi, x est égale à la valeur de A[5] (x= A[5])

Pointeurs : saisie et affichage d'un tableau

Version 1:

```
main()
 { float T[100] , *pt;
   int i,n;
   do {printf("Entrez n \n " );
       scanf(" %d" ,&n);
      }while(n<0 ||n>100);

   pt=T;
   for(i=0;i<n;i++)
      { printf ("Entrez T[%d] \n ",i );
        scanf(" %f" , pt+i);
       }

    for(i=0;i<n;i++)
        printf (" %f \t",*(pt+i));
}
```

Version 2: sans utiliser i

```
main()
 { float T[100] , *pt;
   int n;
   do {printf("Entrez n \n " );
       scanf(" %d" ,&n);
      }while(n<0 ||n>100);


    for(pt=T;pt<T+n;pt++)
{ printf ("Entrez T[%d] \n ",pt-T );
        scanf(" %f" , pt);
       }

    for(pt=T;pt<T+n;pt++)
        printf (" %f \t",*pt);
}
```

Pointeurs et tableaux : exercice

- Soit P un pointeur qui 'pointe' sur un tableau A:

  int A[] = {12, 23, 34, 45, 56, 67, 78, 89, 90};

  int *P; P = A;

- Quelles valeurs ou adresses fournissent ces expressions:
  - **a) *P+2**
  - **b) *(P+2)**
  - **c) &A[4]-3**
  - **d) A+3**
  - **e) &A[7]-P**
  - **f) P+(*P-10)**
  - **g) *(P+*(P+8)-A[7])**

Pointeurs et tableaux : exercice

- Soit P un pointeur qui 'pointe' sur un tableau A:

  int A[] = {12, 23, 34, 45, 56, 67, 78, 89, 90};

  int *P; P = A;

- Quelles valeurs ou adresses fournissent ces expressions:
  - **a) *P+2** => la valeur 14
  - **b) *(P+2)** => la valeur 34
  - **c) &A[4]-3** => l'adresse de la composante A[1]
  - **d) A+3** => l'adresse de la composante A[3]
  - **e) &A[7]-P** => la valeur (indice) 7
  - **f) P+(*P-10)** => l'adresse de la composante A[2]
  - **g) *(P+*(P+8)-A[7])** =>la valeur 23

Pointeurs et tableaux à deux dimensions

- Le nom d'un tableau A à deux dimensions est un pointeur constant sur le premier élément du tableau càd A[0][0].

- En déclarant un tableau A[n][m] et un pointeur P du même type, on peut manipuler le tableau A en utilisant le pointeur P en faisant pointer P sur le premier élément de A (P=&A[0][0]), Ainsi :

  - **P pointe sur A[0][0] et *P désigne A[0][0]**
  - **P+1 pointe sur A[0][1] et *(P+1) désigne A[0][1]**
  - **....**
  - **P+M pointe sur A[1][0] et *(P+M) désigne A[1][0]**
  - **....**
  - **P+i*M pointe sur A[i][0] et *(P+i*M) désigne A[i][0]**
  - **....**
  - **P+i*M+j pointe sur A[i][j] et *(P+i*M+j) désigne A[i][j]**

Pointeurs : saisie et affichage d'une matrice

```
#define N 10
#define M 20
main( )
  { int i, j, A[N][M], *pt;
    pt=&A[0][0];
    for(i=0;i<N;i++)
         for(j=0;j<M;j++)
         { printf ("Entrez A[%d][%d]\n ",i,j );
           scanf(" %d" , pt+i*M+j);
          }

    for(i=0;i<N;i++)
        { for(j=0;j<M;j++)
             printf (" %d \t",*(pt+i*M+j));
          printf ("\n");
        }
  }
```

Pointeurs et tableaux : remarques

En C, on peut définir :

- **Un tableau de pointeurs :**

  Ex : **int *T[10];** //déclaration d'un tableau de 10 pointeurs d'entiers

- **Un pointeur de tableaux :**

  Ex : **int (*pt)[20];** //déclaration d'un pointeur sur des tableaux de 20 éléments

- **Un pointeur de pointeurs :**

  Ex : **int **pt;** //déclaration d'un pointeur pt qui pointe sur des pointeurs d'entiers

Allocation dynamique de mémoire

- Quand on déclare une variable dans un programme, on lui réserve implicitement un certain nombre d'octets en mémoire. Ce nombre est connu avant l'exécution du programme

- Or, il arrive souvent qu'on ne connaît pas la taille des données au moment de la programmation. On réserve alors l'espace maximal prévisible, ce qui conduit à un gaspillage de la mémoire

- Il serait souhaitable d'allouer la mémoire en fonction des données à saisir (par exemple la dimension d'un tableau)

- Il faut donc un moyen pour allouer la mémoire lors de l'exécution du programme : c'est l'allocation dynamique de mémoire

La fonction malloc

- La fonction **malloc** de la bibliothèque <*stdlib*> permet de localiser et de réserver de la mémoire, sa syntaxe est : **malloc(N)**
- Cette fonction retourne un pointeur de type char * pointant vers le premier octet d'une zone mémoire libre de N octets ou le pointeur NULL s'il n'y a pas assez de mémoire libre à allouer.
- Exemple : Si on veut réserver la mémoire pour un texte de 1000 caractères, on peut déclarer un pointeur p sur **char** (**char *p**).
  - L'instruction: **p = malloc(1000);** fournit l'adresse d'un bloc de 1000 octets libres et l'affecte à p. S'il n'y a pas assez de mémoire, p obtient la valeur zéro (NULL).
- Remarque : Il existe d'autres fonctions d'allocation dynamique de mémoire dans la bibliothèque <*stdlib*>

La fonction malloc et free

- Si on veut réserver de la mémoire pour des données qui ne sont pas de type char, il faut convertir le type de la sortie de la fonction malloc à l'aide d'un cast.

- Exemple : on peut réserver la mémoire pour 2 variables contiguës de type int avec l'instruction : p = (int*)malloc(2 * sizeof(int)); où p est un pointeur sur **int** (**int *p**).

- Si on n'a plus besoin d'un bloc de mémoire réservé par **malloc**, alors on peut le libérer à l'aide de la fonction **free** , dont la synatxe est : **free(pointeur)**;

- Si on ne libère pas explicitement la mémoire à l'aide de **free**, alors elle est libérée automatiquement à la fin du programme.

malloc et free : exemple

Saisie et affichage d'un tableau

```
#include<stdio.h>
#include<stdlib.h>
main()
 { float  *pt;
   int i,n;
   printf("Entrez la taille du tableau \n" );
   scanf(" %d" ,&n);

   pt=(float*) malloc(n*sizeof(float));
   if (pt==Null)
    {
      printf( " pas assez de mémoire \n" );
     system(" pause " );
    }
```

```
printf(" Saisie du tableau \n " );
for(i=0;i<n;i++)
  { printf ("Élément  %d ? \n ",i+1);
    scanf(" %f" , pt+i);
   }

printf(" Affichage du tableau \n " );
for(i=0;i<n;i++)
     printf (" %f \t",*(pt+i));
free(pt);
}
```

Exercice

- Ecrire un programme qui permet d'allouer la mémoire dynamiquement à un tableau T de n réels, de saisir le tableau T et une valeur donnée x. Ensuite, le programme affiche si cette valeur se trouve dans le tableau ou non. Utilisez uniquement le formalisme pointeur.