Chapitre 8

LES POINTEURS

Objectif:

① Définir ce qu'est un pointeur.

② Connaître les différentes manières d'usage des pointeurs et des tableaux.

Pré requis:

Les Chapitres précédents de ce cours de langage C

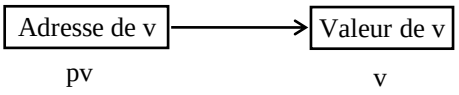
PLAN

Chapitre 8

LES POINTEURS

I/ DEFINITION

Un pointeur est une variable qui représente l'adresse d'une variable.



II/ CONCEPTS FONDAMENTAUX

Dans la mémoire de l'ordinateur, chaque donnée mémorisée occupe une ou plusieurs cases mémoire contigu. Le nombre de cases mémoire, pour stocker une donnée, dépend du type de donnée.

Exemple :

1 char occupe une case mémoire (8 bits)

1 int occupe deux cases mémoire (16 bits)

1 float occupe quatre cases mémoire (32 bits).

Soit v une variable de type quelconque. On peut accéder à cette donnée si l'on connaît l'emplacement de la première cellule (case) mémoire (c-à-d l'adresse).

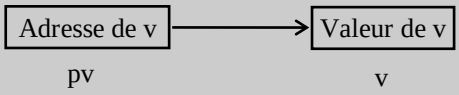
L'adresse de l'emplacement mémoire de v, s'obtient grâce à l'expression &v. & est un opérateur unaire appelé opérateur d'adressage.

Soit pv l'adresse de v, on a donc

pv = &v

pv est dite pointeur vers v.

Donc pv est l'adresse de v et non sa valeur.



on a donc pv = &v ➔ pv égal adresse de v

ou v = *pv ➔ v égal contenue de l'adresse pv.

* est un opérateur unaire appelé opérateur d'indirection.

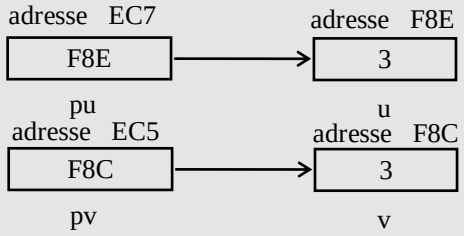
Exemple 1:

```
#include <stdio.h>
main()
{
        int u = 3
        int v
        int *pv         /* pointeur vers un entier */
        int *pu         /* pointeur vers un entier */
        pu = &u         /* on assigne l'adresse de u à pu */
        v = *pu         /* on assigne la valeur de u à v */
        pv = &v         /* on assigne l'adresse de v à pv */
        printf ("\n u = %d      &u = %X      pu = %X      *pu = %d",u,&u,pu,*pu);
        printf ("\n v = %d      &v = %X      pv = %X      *pv = %d",v,&v,pv,*pv);
}
```

on a u = 3 &u = F8E pu = F8E *pu = 3

v = 3 &v F8C pv =F8C *pv = 3

état en mémoire



Remarques:

1- L'opérateur d'adressage (&) agit obligatoirement sur des opérandes associés à une adresse unique, tels que des variables ou des éléments de tableau.

Par exemple : on ne peut pas faire &(2*(u+v))

2- L'opérateur d'indirection (*) ne peut s'appliquer qu'a des opérandes qui sont des pointeurs.

Exemple2:

```
#include <stdio.h>
main()
{
        int u1,u2;
        int v=3;
        int *pv;
        u1=2*(v+5);
        pv=&v
        u2=2*(*pv+5);
        printf("\n u1=%d       u2=%d"u1,u2);
}
```

résultat u1=12 u2=1

III/ LA DECLARATION DES POINTEURS

Un pointeur doit être déclaré avant de l'utiliser dans un programme. La forme générale d'une déclaration d'un pointeur est donc:

```
type_d'objet    *ptvar;
```

type_d'objet est type de donnée que désigne le pointeur.

Exemple:

```
float   p,v;
float    *pv;
```

ici p et v sont deux variables en virgule flottante.

Pv est un pointeur vers une donnée en virgule flottante.

On remarque que pv représente une adresse, non une valeur en virgule flottante.

- On peut affecter à un pointeur une variable initiale.

Exemple:

```
float u,v;                          float u,v;
float  *pv=&v;        ⇔             float  *pv;
                                    pv=&v
```

- Un pointeur qui ne pointe sur aucune mémoire doit être initialisé à zéro '0' ou NULL.

NULL est une constante symbolique égal à zéro définie dans "stdio.h".

IV/ PASSAGE DE POINTEUR A UNE FONCTION

Les pointeurs sont fréquemment passés en arguments de fonctions. Ceci autorise la fonction à avoir accès, à modifier et à renvoyer des objets appartenant au programme appelant. Cette technique de passage d'argument par des pointeurs est dite par référence ou par adresse. En passant un argument par référence, seule l'adresse de la donnée est connue par la fonction. Le contenu de cette adresse est alors accessible librement par le programme et par la fonction.

Exemple1:

```
#include<stdio.h>
main()
{       int u=1,v=3;
        void fonc1(int u, int v);
        void fonc2(int *pu, int *pv);
        printf("\n avant d'appeler fonc1    :u=%d      v=%d",u,v);
        fonc1(u,v);
        printf("\n Après appel de fonc1        u=%d  v=%d",u,v);
        printf("\n avant d'appeler fonc2    :u=%d      v=%d",u,v);
        fonc2(&u,&v);
        printf("\n Après appel de fonc2        u=%d  v=%d",u,v);
}
/*-------------------------------------------------------------------------------------------------*/
void  fonc1(int u,int v)
{
 u=0;  v=0;
printf("\n dans fonc1 u=%d    v=%d",u,v);
return;
}
/*-------------------------------------------------------------------------------------------------*/
void  fonc2(int *u,int *v)
{
 *pu=0;  *pv=0;
printf("\n dans fonc2   *pu=%d         *pv=%d",*pu,*pv);
return;
}
```

Résultat:

```
avant d'appeler  fonc1:        u=1     v=3
dans fonc1              :      u=0     v=0
après appel de fonc1    :      u=1     v=3
avant d'appeler fonc2   :      u=1     v=3
dans fonc2              :      *pu=0   *pv=0
après appel de fonc2    :      u=0     v=0
```

Remarque:

En utilisant les pointeurs comme paramètres de fonction, on agit directement sur les variables du programme et on ne travailles pas sur des copies de ces variables.

Exemple 2: Passage d'un tableau à une fonction qui renvoi un pointeur vers l'un des éléments de ce tableau.

```
#include<stdio.h>
main()
{
int i=0;
char c;
char z[100];    /* déclaration d'un tableau  */
char   *pz;     /* déclaration d'un pointeur  */
char   *ftraite(char  z[]);  /* déclaration d'une fonction */
do
 {
        printf("\nZ[%d]=",i);
        c=getch();
        z[i++]=c;
}
while(c!='\n');
z[--i]='\0';
pz=ftraite(z);
printf("le dernier réel est",*pz);
}
/*-------------------------------------------------------------------------------------------------*/

char  *ftraite(char f[])
```

```
{
char   *pf;
int i;
for(i=0;f[i]!='\0';i++)
   pf=&f[i-1];
return(pf);
}
```

V/ POINTEURS ET TABLEAUX UNIDIMENSIONNELS

1- Accès au éléments d'un tableau

Le nom d'un tableau est un pointeur vers son premier élément. Donc si X est un tableau unidimensionnel, alors l'adresse de sont premier élément est &X[0]ou X, l'adresse de l'élément suivant est &X[1] ou (X+1) ....etc.

Donc l'adresse du $i^{ème}$ élément est &X[i] ou (X+i).

Exemple:

```
#include<stdio.h>
main()
{
 int X[10]={10,11,12,13,14,15,16,17,18};
int i;
for(i=0;i<=9;++i)
        printf("\n i=%d   x[i]=%d   *(X+i)=%d   &X[i]=%X  X+i=%X",
                        i,X[i],*(X+i),&X[i],X+i);
}
```

Donc X+i=l'adresse du i ème élément d'un tableau X X+i=&X[i]

et *(X+i) = le i ème élément d'un tableau X *(X+i)=X[i].

2- Déclaration de tableaux définis comme pointeurs

Pour déclarer un tableau en utilisant seulement les pointeurs, on doit déclarer tout d'abord un pointeur de type désiré, puis allouer de la mémoire nécessaire pour contenir les élément de ce tableau. Une fois terminé, on doit libérer cette mémoire.

↗ la déclaration de la mémoire est déjà vue. (ex int *X est un pointeur vers un entier)

↗ L'allocation de la mémoire peut se faire à l'intérieur du programme suivant les besoins. Cette opération est assurée par la fonction **malloc**, de la bibliothèque **alloc.h**, de la manière suivante:

```
        X=(int  *)malloc(nbre*sizeof(int));
```

avec nbre est le nombre d'éléments souhaité.

Cette opération est dite allocation dynamique de la mémoire.

↗ La libération de la mémoire se fait à l'aide de la fonction free du bibliothèque alloc.h comme suit:

```
        free(X);
```

Exemple:

Refaire l'exemple de tri d'une liste de nombre:

```
#include<stdio.h>
main()
{
int i,n;
int *X;
void  trier(int n, int *X);
printf("\n nombre de valeurs à trier ?");
scanf("%d",&n);
x=(int  *)malloc(n*sizeof(int));
for(i=0;i<n;++i)
   {
        printf("i=%d    X=",i+1);
        scanf("%d",X+i);
   }
trier(n,X);
printf("\n\n liste triée   \n");
for(i=0;i<n;++i)
  printf("i=%d x=%d\n",i+1,*(X+i));
}
/*-------------------------------------------------------------------------------------------------*/
```

```
void trier(int n, int *X)
{
  int i, element,temp;
for(element=0;element<n-1;++element)
  for (i=element,i<n,++i)
  if(*(X+1)<*(X+element))
          {
           temp=*(X+element);
          *(X+element)=*(X+i);
          *(X+i)=temp;
          }
return;
}
```

VI/ POINTEURS ET TABLEAUX MULTIDIMENSIONNELS

Un tableau multidimensionnel peut être représenté sous forme de pointeur vers un groupe de tableaux consécutifs ou sous forme d'un tableaux de pointeurs.

Déclaration:

↗ Pointeur vers un groupe de tableaux

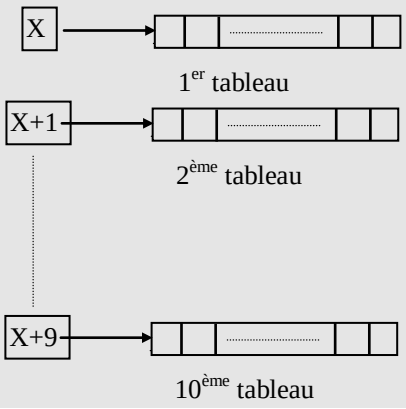
syntaxe:

```
type_donnée     (* vatptr) [taille1][taille2].....[taillen];
```

Exemple:

```
int  (*X)[20]
```

X est un pointeur vers des tableaux de 20 entiers.



X[2][5] est équivalent à *(*(X+2)+5)

L'allocation de la mémoire se fait comme suit:

```
X=(int *)malloc(10*20*sizeof(int));
```

Exercice: En utilisant les tableaux de type pointeur vers un groupe de tableaux, faire l'addition de deux tableaux d'entiers.

↗ Tableau de pointeurs

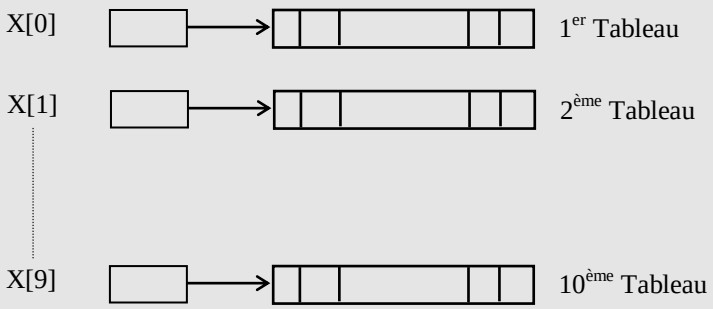
syntaxe

```
type_donnée     *X[taille1][taille2] . . . .[taille n-1];
```

Exemple:

```
int   *X[20];
```

X est un tableau de pointeurs vers des tableaux de 20 entiers.



X[2][5] est équivalent à *(X[2]+5)

Remarque:

Il faux toujours faire l'allocation de l'espace mémoire nécessaire à l'aide de la fonction malloc.

L'allocation de la mémoire se fait comme suit:

```
for(i=0;i<10;i++)
        X[i]=(int   *)malloc(20*sizeof(int));
```

Exercice: refaire l'exercice d'addition de deux tableaux d'entiers en utilisant les tableaux de pointeurs.

ISET DE SFAX
II N2

TP N°5
PROGRAMMATION C

A.U 1997/1998
Cours de Mr: TAYARI Lassaad

Les pointeurs

EX N°1 En utilisant les pointeurs, écrire un programme C qui permet de trier une liste de nombres entiers.

Il est préférable de créer une fonction qui effectue l'opération de tri.

EX N°2: En utilisant les pointeurs:

1- Ecrire une fonction C qui permet se saisir les éléments d'une matrice d'entiers de n lignes et m colonnes.

2- Ecrire une fonction C qui permet d'afficher les éléments d'une matrice de n lignes et m colonnes.

3- Ecrire une fonction qui permet d'additionner deux matrices de n lignes et m colonnes.

4- Ecrire une fonction qui permet la multiplication de deux matrices de n lignes et m colonnes.

5- Ecrire une fonction qui affiche le menu suivant:

M E N U

1- Saisir une matrice

2- Afficher une matrice

3- Addition de deux matrices

4- Multiplication de deux matrices

5- Fin de travail

Choix=?

6- Ecrire une fonction main qui appelle les fonctions précédentes afin de réaliser un programme de traitement de matrices. Dans cette fonction on doit déclarer trois matrices A, B et C. Pour les deux premières option (Saisir une matrice et Afficher une matrice) le programme doit demander a l'utilisateur le nom de la matrice (A, B ou C), pour les deux autres option le programme doit faire l'opération sur les matrices A et B puis mettre le résultat dans la matrice C.

Bon Travail