Exercice 1

```
main(){

short A, B, *P;/*supposons que ces variables occupent la mémoire à partir de l'adresse 01A0 */
A = 10;
B = 50;
P = &A ;
B = *P;
*P = 20;
P = &B;
*P += 15;
}
```

Donnez les valeurs de A, B, P après chaque instruction

Exercice 2

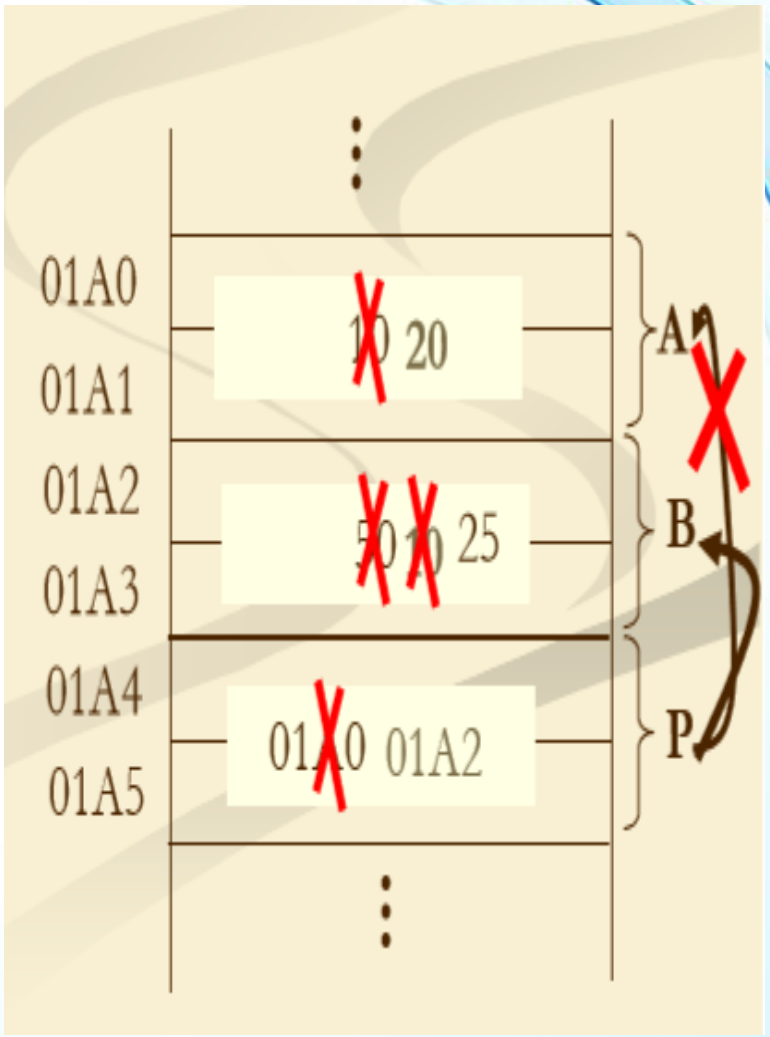
```
main(){

float a , *p;   /*supposons que ces variables sont représentées  en mémoire à partir de l'adresse 01BE*/
p = &a;
printf("Entrer une valeur réelle:");
scanf("%f",p);  // on saisie la valeur 1.4
printf("\nAdresse de a = %x  Contenu de a = %f",p,*p);
*p += 0.4;
 printf("\na = %f  ", a);
}
```

Donnez les valeurs de a et p après chaque instruction

Solution 1

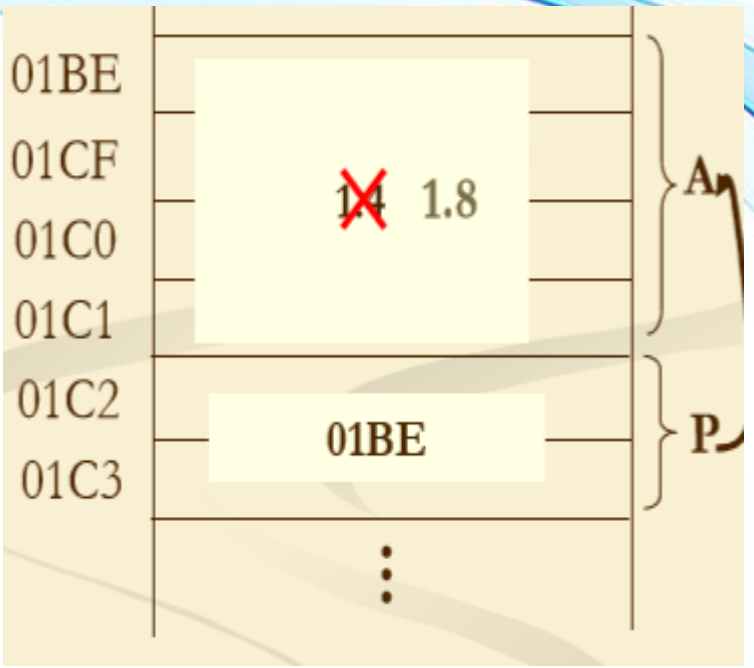
```
main(){
short A, B, *P;/*supposons que ces variables occupent la mémoire à partir de l'adresse 01A0 */
A = 10;
B = 50;
P = &A ; // se lit mettre dans P l'adresse de A
B = *P; /* mettre dans B le contenu de l'emplacement mémoire pointé par P */
*P = 20; /*mettre la valeur 20 dans l'emplacement
mémoire pointé par P */
P = &B; // P pointe sur B
*P += 15; // additionner 15 au contenu de l'adresse sur
laquelle pointe P
}
```



Solution 2

```
main(){

float a , *p;  /*supposons que ces variables sont représentées en mémoire à partir de l'adresse 01BE*/
p = &a;
printf("Entrer une valeur réelle:");
scanf("%f",p);  // on saisie la valeur 1.4
printf("\nAdresse de a = %x Contenu de a = %f",p,*p);
*p += 0.4;
 printf("\na = %f  ", a);
}
```



Affichage sur Ecran

Entrer une valeur réelle : 1.4
Adresse de a : 01BE Contenu de a = 1.4
a = 1.8

Pointeurs et tableaux

- Comme on l'a déjà vu au chapitre « Tableaux », 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

- Si on déclare un tableau statique et un pointeur ainsi:

```
int tab[10];
int *p;
```

alors l'affectation : **p=&tab[0]** est équivalente à **p=tab** et ***p** désigne **tab[0].**

- Les crochets sont une simplification d'écriture: **tab[i]⇔ *(tab+i)**

- Il existe tout de même une différence entre pointeur et tableau: un nom de tableau n'est pas une variable, on ne peut donc rien affecter au nom de tableau tab=p (incorrecte) , contrairement au pointeur alors **p=tab**

Pointeurs et tableaux

- 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 et tableaux: 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+1);
            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 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 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 pt sur **char *pt**.
  - L'instruction: **T = (char *) malloc(1000);** fournit l'adresse d'un bloc de 1000 octets libres et l'affecte à T. S'il n'y a pas assez de mémoire, T 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.

La fonction malloc et free

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);
}
```

La fonction calloc

- La fonction **calloc(K,N)** retourne un pointeur vers une zone mémoire formée d'un tableau de K éléments de taille identique égale à N octets .

- Elle retourne la valeur Null en cas d'échec.

- La zone mémoire réservé par calloc est constituée de K x N octets, tous initialisés à la valeur 0.

- Le tableau renvoyé par calloc est de type char*.
- Exemple:

```
int n=100;
long *tab=(long*) calloc (n, sizeof(long));  /* n fois 0 */
```

Fonction realloc

- La fonction **realloc(p,N)** permet de modifier la taille de la zone mémoire(déjà réservé par malloc ou calloc) pointée par le pointeur p à N octets.
- Sa définition se trouve dans la bibliothèque standard <stdlib.h>
- Elle retourne un pointeur sur une zone mémoire de N octets.
- Elle retourne la valeur Null en cas d'echec.
- Elle préserve les premiers octets pointés par le pointeur p.