Séance 5 : Les pointeurs

Langage C : séance 5

Utilité des pointeurs :

- **Passage de paramètres par adresse dans une fonction :**

Les pointeurs sont utilisés pour le passage de paramètres dans une fonction dont on souhaite modifier le contenu.

- **Utilisation de la mémoire dynamique (ou Tas) :**

Une partie de la mémoire appelée « mémoire dynamique » est disponible par un programme en langage C pour y stocker des données. Mais son accès se fait uniquement par des pointeurs.

- **Construction de données récursives :**

La construction de données récursives (liste chaînée, arbre , …) ne peut se faire que dans la mémoire dynamique, donc avec l'utilisation de pointeurs pour y accéder.

Langage C : séance 5

Les opérateurs de base :

- **L'opérateur d'adresse & :**

Permet de récupérer l'adresse d'une variable.

Exemple : int i=2;

```
int * pti =  &i;
*pti=3;
```

- **Notion de pointeur :**

Déclaration et initialisation d'un pointeur :

type * nom_pointeur=NULL;

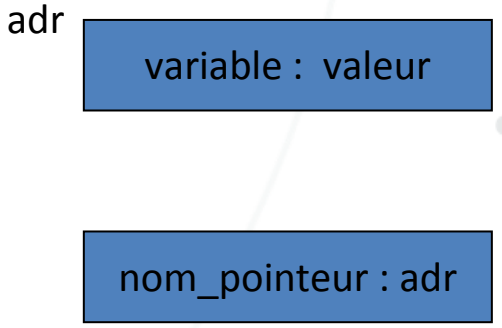
Affectation d'une valeur à un pointeur :

nom_pointeur= **&** variable;

Accès au contenu d'une adresse :

*nom_pointeur= valeur;

Affichage d'une adresse : printf''%p'', nom_pointeur);

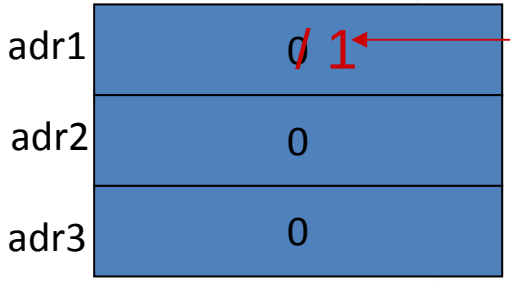


Langage C : séance 5

Les tableaux statiques :

- Déclaration d'un tableau statique :

int nom_tableau[3]={0,0,0};



Occupation mémoire de ce tableau :

3 (cases) x 4 (octets/int) = 12 octets consécutifs.

sizeof(nom_tableau) ➔ 12

Adresse mémoire de la première case :

« nom_tableau » ➔ adr1

printf(''%p'', nom_tableau); ➔ affiche la valeur de l'adresse en héxa.

Accès par un autre pointeur :

il est possible de faire :

```
int * ptr=nom_tableau;
*ptr=1;
ptr++;
```

Langage C : séance 5

Les tableaux statiques :

- Déclaration d'un tableau statique :

int nom_tableau[3]={0,0,0};

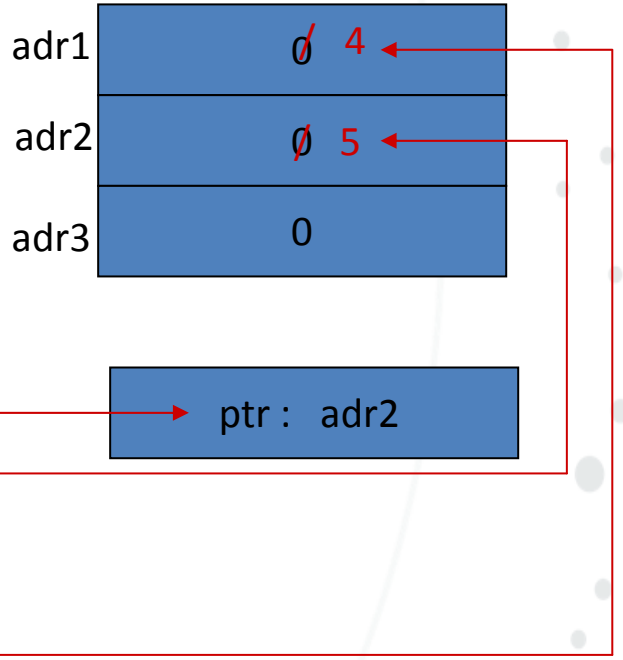
Accès par un autre pointeur :

il est possible de faire :

```
int * ptr=nom_tableau;

ptr++;
*ptr=5;

ptr--;
*ptr=4;
```



mais : nom_tableau=ptr; ← **interdit** car « nom_tableau » adresse constante.
« nom_tableau » est identique à « &nom_tableau » !!! (tableau statique)

Langage C : séance 5

Les tableaux statiques :

- Déclaration d'un tableau statique :

int nom_tableau[3]={0,0,0};

Accès par un autre pointeur :

int * ptr=nom_tableau;

Equivalence :

ptr[i] ➔ *(ptr + i)

On peut écrire : nom_tableau[i] ou *(nom_tableau + i)

mais attention :

sizeof (nom_tableau) ➔ 12

sizeof (ptr) ➔ 4 car ptr est un pointeur (adresse mémoire codé sur 4 octets).

sizeof(nom_tableau)/sizeof(nom_tableau[0]) ➔ nbre de cases du tableau !

Langage C : séance 5

Passage par adresse de paramètres :

Exemple :

```
void lireEntier(int * j) {
        scanf(''%d'' , j);
}

int main() {
int i=5;
lireEntier(& i);
printf(" i=%d'' ,i);
return 0;
}
```

Paramètre formel de la fonction

Variable locale (au main)

La variable i est un paramètre réel ou effectif passée par adresse lors de l'appel à la fonction

La valeur du paramètre réel (&i) correspond bien à une adresse !

Langage C : séance 5

Passage par adresse de paramètres :

Exemple :

```
void lireEntier(int * j) {
        scanf(''%d'' , j);
}

int main() {
int i=5;
lireEntier(& i);
printf(" i=%d'' ,i);
return 0;
}
```

Variable locale (au main)

Etape 1 de l'exécution :

création de la variable i dans la pile (zone des variables locales) et initialisation

Langage C : séance 5

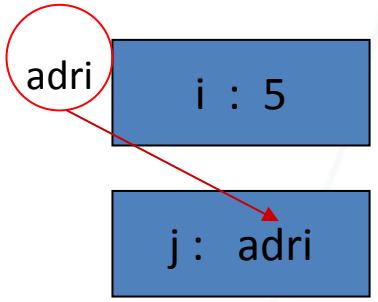
Passage par adresse de paramètres :

Exemple :

Paramètre formel de la fonction

```
void lireEntier(int * j) {
        scanf(''%d'' , j);
}

int main() {
int i=5;
lireEntier(& i);
printf(" i=%d'' ,i);
return 0;
}
```



Etape 2 de l'exécution :

création de la variable j dans la pile,

et initialisation avec la valeur du paramètre réel (&i ➔ adri).

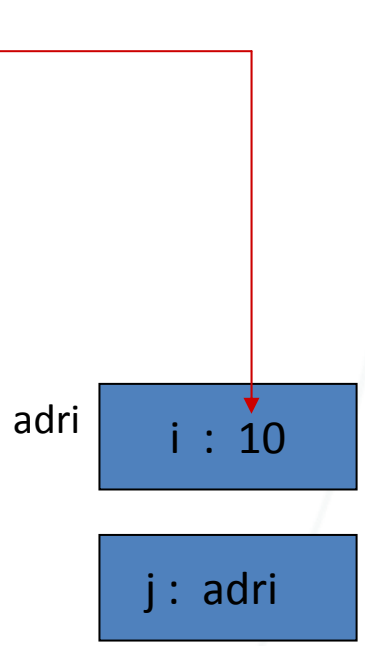
Langage C : séance 5

Passage par adresse de paramètres :

Exemple :

```
void lireEntier(int *j) {
        scanf(''%d'' , j);
}

int main() {
int i=5;
lireEntier(&i);
printf(" i=%d'' ,i);
return 0;
}
```



Etape 3 de l'exécution :

Saisie de la valeur d'un entier et stockage à l'adresse contenue dans le paramètre formel j.

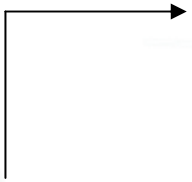
Langage C : séance 5

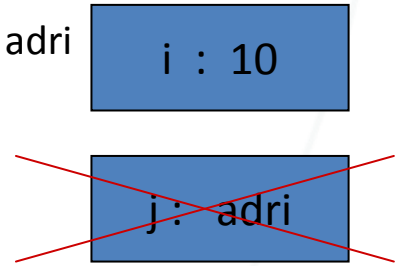
Passage par adresse de paramètres :

Exemple :

```
void lireEntier(int *j) {
        scanf(''%d'' , j);
}

int main() {
int i=5;
lireEntier(& i);
printf('' i=%d'' ,i);
return 0;
}
```





Etape 4 de l'exécution :

Retour dans le main : destruction des variables de la fonction lireEntier.

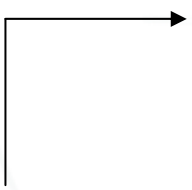
Langage C : séance 5

Passage par adresse de paramètres :

Exemple :

```
void lireEntier(int *j) {
        scanf(''%d'' , j);
}

int main() {
int i=5;
lireEntier(& i);
printf(" i=%d'' ,i);
return 0;
}
```



Etape 5 de l'exécution :

Affichage de la nouvelle valeur de i. ➔ *i=10*

Langage C : séance 5

Passage par adresse de paramètres :

Résumé du passage par adresse :

- **Le passage par adresse modifie le paramètre réel**.

```
void f(T *t1);
T t=val1;
f(&t);        // la fonction f peut modifier la valeur de t
// t peut avoir une nouvelle
```

- **Lors d'un passage par adresse, la valeur du paramètre réel ne peut pas être une constante**.

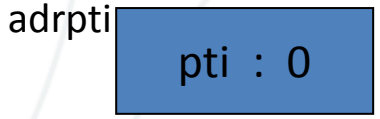
```
void f(int *t1);
f( 10);       // interdit
```

Langage C : séance 5

Déclaration et Réservation avec un pointeur (fonction malloc de <stdlib.h>):

- **déclaration et initialisation à NULL**.

```
int * pti= NULL;
// NULL a pour valeur 0
```

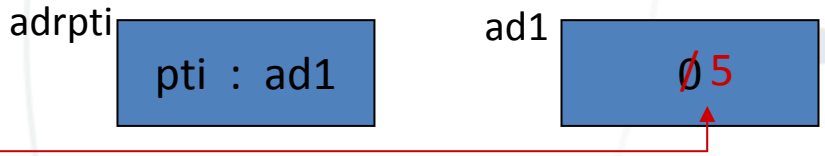


Équivalent à :

```
int *pti;
pti=NULL;
```

- **Réservation de l'espace en mémoire dynamique**.

```
pti=(int * )malloc( sizeof(int));
// réservation de l'espace pour un entier
*pti=5;
```



La fonction malloc retourne l'adresse de la zone mémoire allouée en mémoire dynamique, NULL si échec.
Elle reçoit en paramètre le nombre d'octets à réserver.

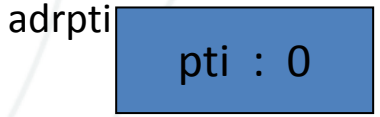
Les expressions suivantes :	ont pour valeur :
pti	ad1
&pti	adrpti
*pti	5

Langage C : séance 5

Déclaration et Réservation d'un tableau avec un pointeur :

- **déclaration et initialisation à NULL**.

```
int * pti= NULL;
// NULL a pour valeur 0
```



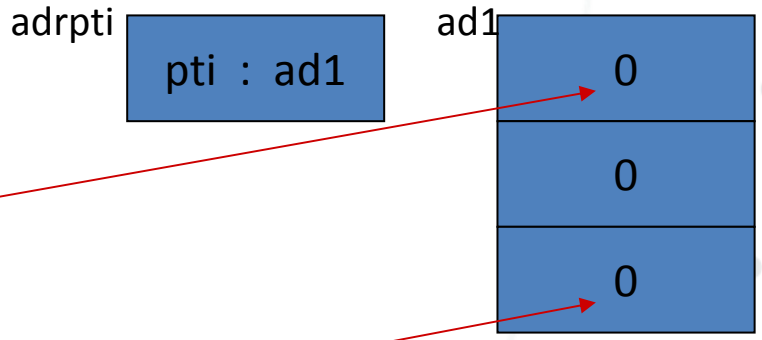
- **Réservation de l'espace en mémoire dynamique**.

```
pti=(int * )malloc( sizeof(int) * 3);
// réservation de l'espace pour stocker 3 entiers.
```

on peut écrire ensuite :

*pti ou pti[0]

*(pti+i) ou pti[i]



Langage C : séance 5

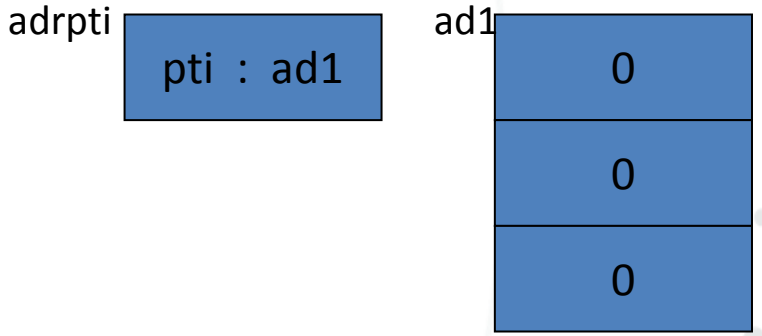
Déclaration et Réservation d'un tableau avec un pointeur :

- **déclaration et réservation**.

```
int * pti= (int * )malloc( sizeof(int) * 3);

on peut écrire :
for (i=0; i<3; i++)
                {
                printf(" valeur de pti[%d]=" ,i);
                scanf("%d" ,& pti[i]);
                // ou scanf ("%d" , pti+i );
                }

for (i=0; i<3; i++)
                {
                printf(" valeur de pti[%d]=%d" ,i, pti[i]);
                }
```

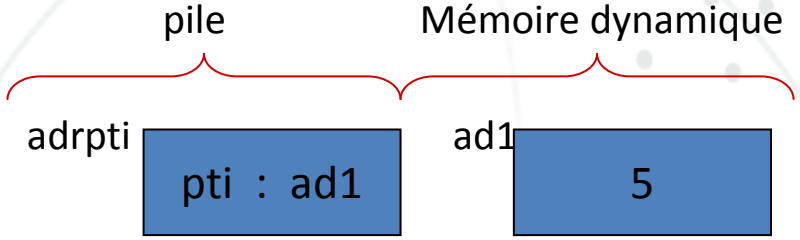


Langage C : séance 5

Libération mémoire en mémoire dynamique (fonction free de <stdlib.h>):

- **déclaration et réservation**.

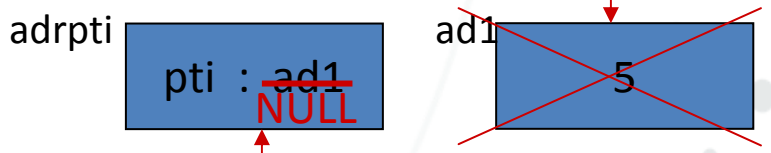
```
int * pti= (int * )malloc( sizeof(int) );
*pti=5;
```



- Une zone allouée dans le tas (avec un malloc) doit être libérée si elle ne doit plus être utilisée.
  - ➔ libération par la fonction « free » (de <stdlib.h>)

```
free(pti);
```

La fonction free libère la zone allouée par le malloc.




```
// on écrase la valeur de l'adresse pour ne plus y retourner :
pti=NULL;
```

Langage C : séance 5

Arithmétique des pointeurs :

Une variable pointeur représentant une adresse, on ne peut qu'incrémenter une valeur à un pointeur ou soustraire deux pointeurs (pas d'addition, de multiplication ou de division de deux pointeurs).

- Incrémentation d'un pointeur :

```
int tab[5]={1,2,3,4,5};
int i,*pti;
pti= &tab[0];
for (i=0 ; i<5; i++,pti++)
        printf(''%d \n'', *pti);
```

- Différence de deux pointeurs :

```
int tab[10]={1,2,3,4,5};
int i,*ptr1, *ptr2;
ptr1= &tab[0];
ptr2=&tab[4];
printf(''%d \n'', ptr2-ptr1);   // ➔  4
```

Langage C : séance 5

Autres fonctions d'allocation (dans <stdlib.h>) :

• **la fonction « calloc » : void*** calloc**(size_t** n**, size_t** taille**);**

(size_t correspond à un entier).

La fonction « calloc » permet d'allouer un nombre n (premier paramètre) de données de taille t (deuxième paramètre).

Exemple d'utilisation :

```
char * ch= calloc(10, sizeof(char));  // alloue un tableau de 10 char
```

Elle initialise également la zone allouée à 0 (ce que ne fait pas la fonction « malloc »).

• **la fonction « realloc » : void*** realloc**(void * ptr, size_t** taille**);**

Elle sert à réattribuer de la mémoire à un pointeur mais pour une taille mémoire différente. En cas d'agrandissement de la zone mémoire, elle déplace le contenu des données dans la nouvelle zone.

Exemple d'utilisation :

```
char * ch= calloc(10, sizeof(char));  // alloue un tableau de 10 char
strcpy(ch, ''bonjour'');
…
ch=realloc(ch, 20*sizeof(char));  // la nouvelle zone contient ''bonjour''
```

Langage C : séance 5

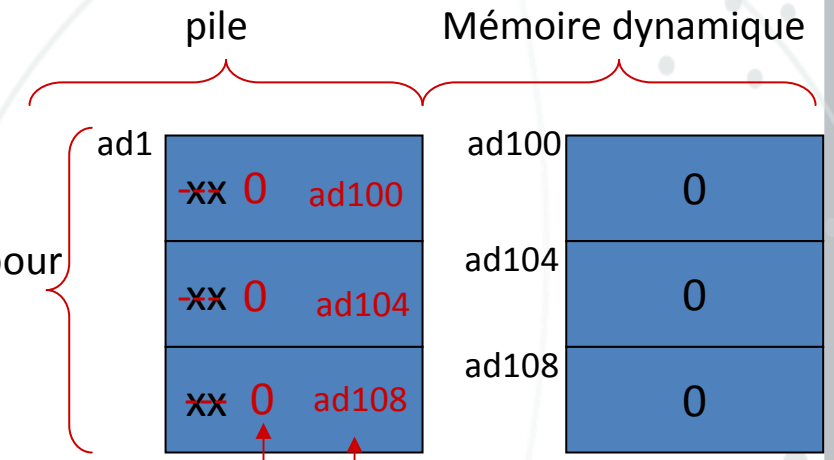
Déclaration et Réservation d'un tableau de pointeurs :

- **déclaration (en variable locale) :**

```
int * t[3];

int i;
for (i=0;i<3;i++)
        t[i]=NULL;
```

Zone réservée pour le tableau t



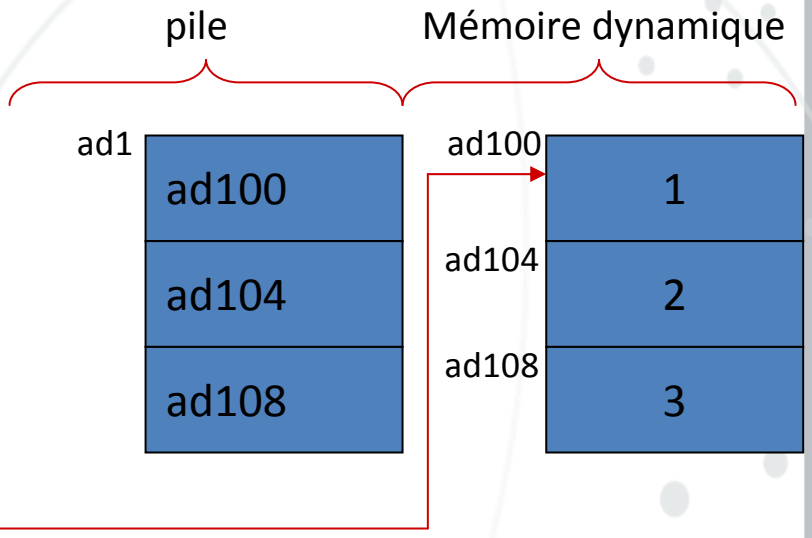
- **Réservation :**

```
for (i=0;i<3;i++)
        t[i]=(int *)calloc(1, sizeof(int));
```

xx ➔ n'importe quelle valeur

Langage C : séance 5

Déclaration et Réservation d'un tableau de pointeurs :



- **utilisation de la variable t précédente :**

```
// saisie des valeurs entières :
for (i=0;i<3;i++)
        {
        printf(''\nvaleur %d='',i);
        scanf(''%d'', t[i]);  // on tape 1,2,3
        }

// affichage des valeurs entières :
for (i=0;i<3;i++)
        printf(''\nvaleur =%d'', *(t[i]));
```

- **Libération :**

```
for (i=0;i<3;i++) {
        free(t[i]);
        t[i]=NULL;
}
```

Langage C : séance 5

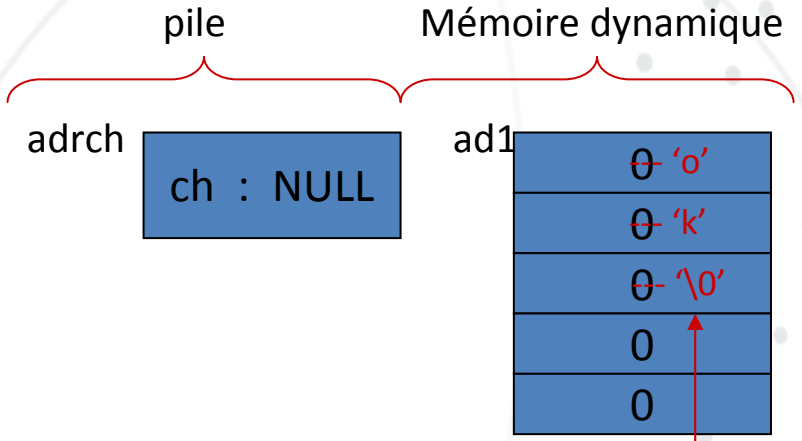
Déclaration et Réservation d'une chaîne de caractères dynamique :

- **déclaration :**

```
char * ch=NULL;
```

- **Réservation :**

```
ch=(char *)malloc(sizeof(char)*5);
// ou    ch=(char *)calloc(5,sizeof(char));
// ou    ch=(char *)realloc(ch,sizeof(char)*5);
```



- **affectation :**

```
strcpy(ch,''ok'');
// saisie clavier :
scanf(''%4s'',ch);
// affichage écran :
printf(''\n %s'',ch);
```

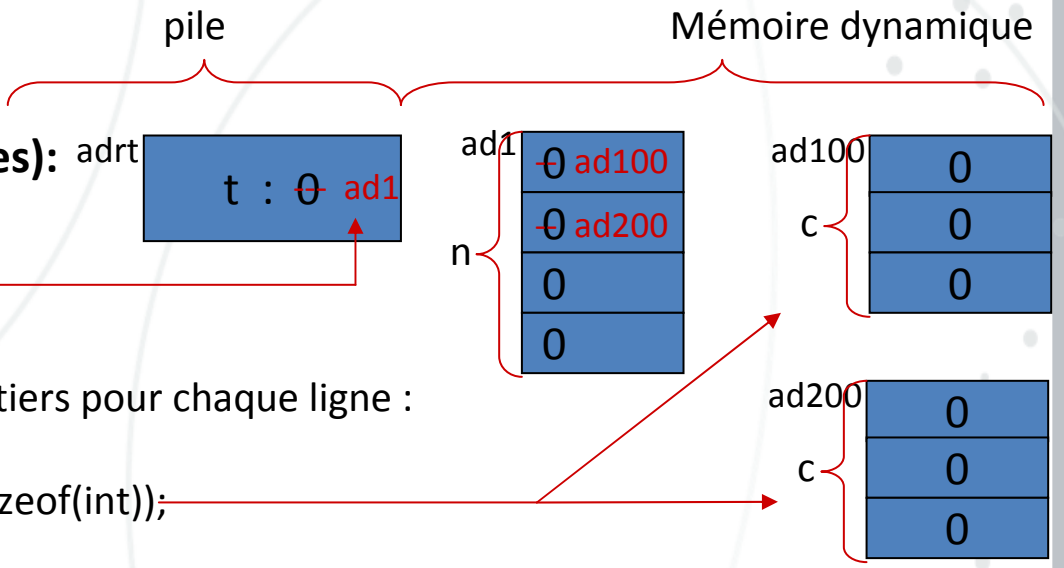
- **libération :** free(ch);
  ch=NULL;

Langage C : séance 5

Déclaration et Réservation d'un tableau multidimensionnel :

- **déclaration :**

```
int ** t=NULL;
```



- **Réservation (pour n lignes de c colonnes):**

```
// réservation des n lignes :
t=(int **)calloc(n,sizeof(int *));

// réservation de c colonnes d'entiers pour chaque ligne :
for (i=0;i<2;i++)
        t[i]=(int *)calloc(c, sizeof(int));
```

- **utilisation :**

```
for (i=0;i<n;i++)
   for (j=0;j<c;j++)
        t[i][j]=0;   // ou  scanf(''%d'',& t[i][j]);  ou  printf(''%d'',t[i][j]);
```

- **libération :**

```
for (i=0;i<2;i++)
        free(t[i]);
free(t);
t=NULL;
```