Programmation en langage C

Plan

La taille des variables

```
int  maVariable = 0
```

1. Le programme demande au système d'exploitation (Windows, Linux, Mac OS…) la permission d'utiliser un peu de mémoire ;

2. le système d'exploitation répond au programme en lui indiquant où il peut stocker cette variable (il lui donne l'adresse qu'il lui a réservée) ;

3. lorsque la fonction est terminée, la variable est automatiquement supprimée de la mémoire.

La taille des variables

Type	Codage en mémoire
char	1 octet
int	2 ou 4 octets
float	4 octets
double	8 octets

- **L'opérateur sizeof() : vérifier quelle taille occupe chacun des types sur votre ordinateur.**

```
printf("char : %d octets\n", sizeof(char));

printf("int : %d octets\n", sizeof(int));

printf("long : %d octets\n", sizeof(long));

printf("double : %d octets\n", sizeof(double));
```

```
char : 1 octets
int : 4 octets
long : 4 octets
double : 8 octets
```

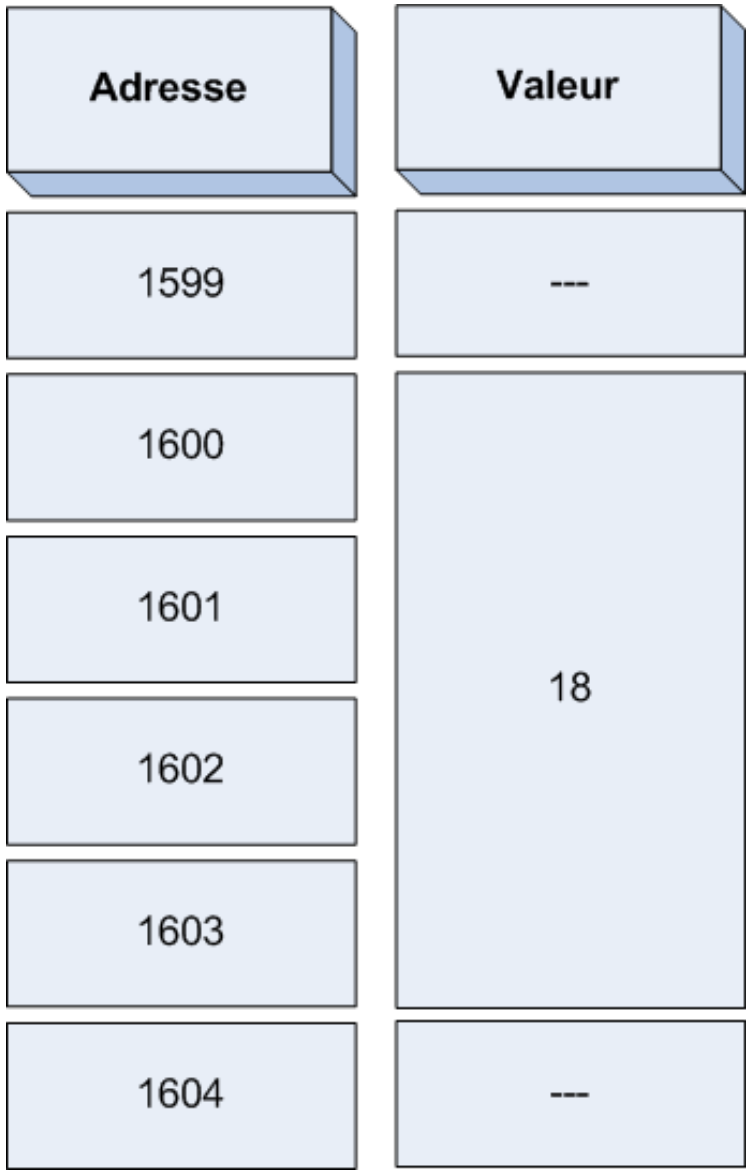
La taille des variables

```c
#include <stdio.h>

typedef struct Coordonnees Coordonnees;
struct Coordonnees
{
    int x;
    int y;
};

int main()
{
    printf("Coordonnees : %d octets\n", sizeof(Coordonnees));

    return 0;
}
```

```
Coordonnees : 8 octets
```

La taille des variables

int maVariable = 18



4

Allocation dynamique

L’allocation dynamique de la mémoire

1. **malloc (« Memory ALLOCation », c'est-à-dire « Allocation de mémoire ») :** demande au système d'exploitation la permission d'utiliser de la mémoire ;

2. **free (« Libérer ») :** permet d'indiquer au système d'exploitation que l'on n'a plus besoin de la mémoire qu'on avait demandée. La place en mémoire est libérée, un autre programme peut alors s'en servir au besoin.

3. **calloc() :** a le même rôle que malloc(). Elle permet d'allouer de la mémoire.

4. **realloc() :** permet de **ré-attribuer de la mémoire** à un pointeur

Il faut inclure la bibliothèque **<stdlib.h>**

4

Allocation dynamique

L’allocation dynamique de la mémoire

Etapes de l’allocation manuelle de mémoire :

1. **appeler malloc** pour demander de la mémoire ;

2. **vérifier la valeur retournée** par malloc pour savoir si le système d’exploitation a bien réussi à allouer la mémoire ;

3. une fois qu'on a fini d'utiliser la mémoire, on doit la **libérer avec free.** Si on ne le fait pas, on s'expose à des fuites de mémoire, c'est-à-dire que le programme risque de prendre beaucoup de mémoire alors qu'il n'a en réalité plus besoin de tout cet espace.

4

Allocation dynamique

L’allocation dynamique de la mémoire

malloc : demande d'allocation de mémoire

Prototype :

```
void* malloc(size_t   nombreOctetsNecessaires);
```

1. Paramètre d’entrée : le nombre d'octets à réserver

2. Type de renvoie : void -> la fonction ne retourne aucune valeur.
   void* -> pointeur sur vide (c’est un pointeur générique)
   => La fonction renvoie un pointeur indiquant l'adresse réservée

4

Allocation dynamique

L'allocation dynamique de la mémoire

malloc : demande d'allocation de mémoire

Exemple :

```
// On crée un pointeur sur int
int* memoireAllouee = NULL;

// La fonction malloc inscrit dans notre pointeur l'adresse qui a été reservée
memoireAllouee = malloc(sizeof(int));
```

- si l'allocation a marché, notre **pointeur contient une adresse** ;
- si l'allocation a échoué, notre **pointeur contient l'adresse NULL**.

L'allocation dynamique de la mémoire

malloc : demande d'allocation de mémoire

Exemple : tester le pointeur

```
int* memoireAllouee = NULL;
memoireAllouee = malloc(sizeof(int));

   // Si l'allocation a échoué
   if (memoireAllouee == NULL)
   {
      // On arrête immédiatement le programme
      exit(0);
   }

    // sinon, on continue le programme
```

L’allocation dynamique de la mémoire

free : libérer de la mémoire

Prototype : `void free(void* pointeur);`

Paramètre d’entrée : l'adresse mémoire à libérer

4 Allocation dynamique

L'allocation dynamique de la mémoire

Exemple :

```
int* memoireAllouee = NULL;
memoireAllouee = malloc(sizeof(int));
if (memoireAllouee == NULL)
   {
      exit(0);
   }

 // Utilisation de la mémoire
printf("Quel age avez-vous ? ");
scanf("%d", memoireAllouee);
printf("Vous avez %d ans\n", *memoireAllouee);

//libération de la mémoire
free(memoireAllouee);
```

Allocation dynamique d'une matrice

La fonction calloc()

La différence entre les fonctions calloc() et malloc(), c'est que **calloc() initialise à 0 tous les éléments de la zone mémoire**.

Prototype : `void* calloc(size_t num_elements, size_t size);`

Le premier argument : le nombre d'éléments qu'on souhaite pouvoir stocker en mémoire

Le deuxième argument : la taille des éléments que l'on obtient avec sizeof().

Allocation dynamique d'une matrice

La fonction calloc()

Exemple : allocation de 15 cases mémoires et initialisation de chacune à zéro

```
int  *tab = calloc(15, sizeof(int)); /* les 15 cases contiennent 0. */

if (tab == NULL) {
    printf("ERREUR : probleme de memoire ");
    exit(-1);
}

/* suite du programme */
```

Allocation dynamique d'une matrice

La fonction realloc()

- Permet de ré-attribuer de la mémoire à un pointeur mais pour une taille mémoire différente.
- S'utilise après l'utilisation de malloc() ou calloc().
- Peut être rappelée plusieurs fois de suite (dans une boucle).
- Renvoie l'adresse de la nouvelle zone mémoire allouée.

Prototype : `void * realloc(void *ptr, size_t size);`

Le premier argument : le pointeur sur lequel on désire effectuer l'opération,

Le deuxième argument : la nouvelle taille de l'espace mémoire qu'on veut allouer.

Allocation dynamique d'une matrice

La fonction realloc()

Exemple :

```
int  *tab = calloc(15, sizeof(int)); /* les 15 cases contiennent 0. */

if (tab == NULL) {
    printf("ERREUR : probleme de memoire ");
    exit(-1);
}

tab= realloc(tab, 20 * sizeof(int));

if (tab == NULL) {
    printf("ERREUR : probleme de memoire ");
    exit(-1);
}
free(tab);
```

4

Allocation dynamique

Allocation dynamique d'un tableau

Exemple : stocker dans un tableau les notes de tous les étudiants d'une classe

Méthode 1 :

```
double noteEtudiants[40];
```

```
double noteEtudiants[nbEtudiants];
```

4

Allocation dynamique

Allocation dynamique d'un tableau

Exemple : stocker dans un tableau les notes de tous les étudiants d'une classe

Méthode 2 : réserver **nbEtudiants * sizeof(double)** octets en mémoire

```
double *notes = malloc(nbEtudiants * sizeof(double));
```

- Permet de créer un tableau de type double qui a une taille correspondant exactement au nombre des étudiants

Résolution du programme dans l'ordre :

1. demander à l'utilisateur combien il y a d'étudiants;
2. créer un tableau de double ayant une taille égale au nombre d'étudiants (avec malloc) ;
3. demander la note de chaque étudiant un par un, et la stocker dans le tableau ;
4. afficher les notes
5. Libérer l'espace alloué par le tableau

Exemple : stocker dans un tableau les notes de tous les étudiants d'une classe

```
double * notes, *P;
int N = 0;

printf("Quel est le nombre des etudiants ?");
scanf("%d", &N);

notes = (double *) malloc(N * sizeof(double));
if(notes == NULL)
{
    exit(0);
}
for(P=notes; P<notes+N; P++)
{
    printf("Note %d =",P-notes);
    scanf("%d",P);
}
printf("LES NOTES :\n");
for(P=notes; P<notes+N; P++)
{
    printf("%d\n",*P);
}
free(notes);
```

4

Allocation dynamique

Allocation dynamique d’une matrice

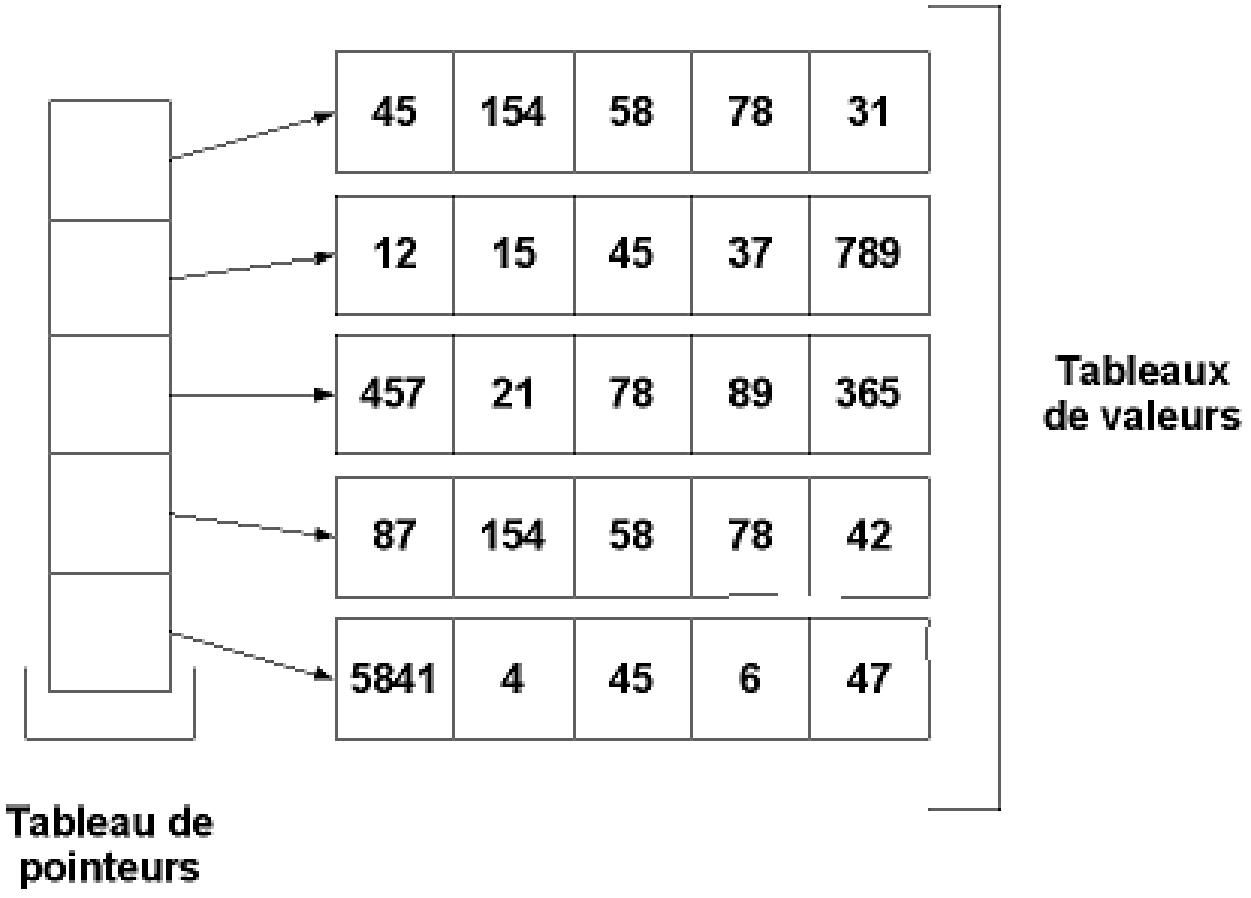
L'allocation dynamique d'un tableau à plusieurs dimensions, s'effectue en allouant les dimensions une par une.

Pour le cas d’un tableau à deux dimensions, il est nécessaire d’utiliser un double pointeur :

```
int    **ptr;     //2 étoiles pour 2 dimensions
```

Allocation dynamique d'une matrice

Un tableau à deux dimensions est un simple tableau de pointeurs, chacun de ces pointeurs va pointer sur un espace représentant un tableau à une dimension



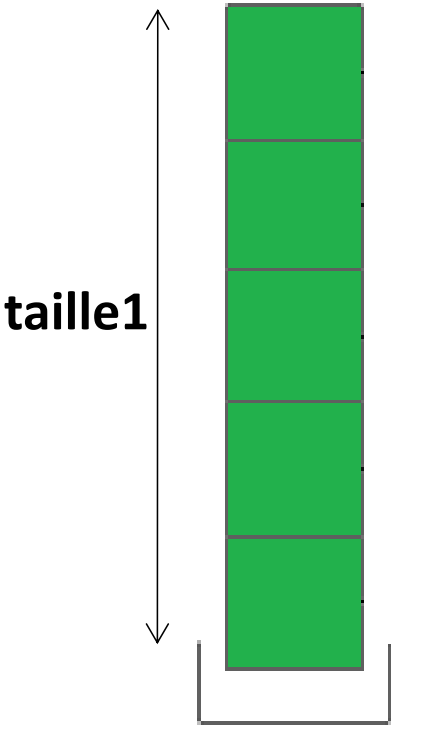
Allocation dynamique d'une matrice

Allocation

L'allocation doit se faire en deux étapes :

Etape 1 : Allocation de la première dimension (tableau de pointeurs)

```
int  **ptr;

//On alloue 'taille1' pointeurs
ptr = (int **) malloc(taille1 * sizeof(int*));
//ptr = (int **) malloc(taille1 * sizeof(*ptr));

if(ptr == NULL)
    //message d'erreur et quitter le programme
```



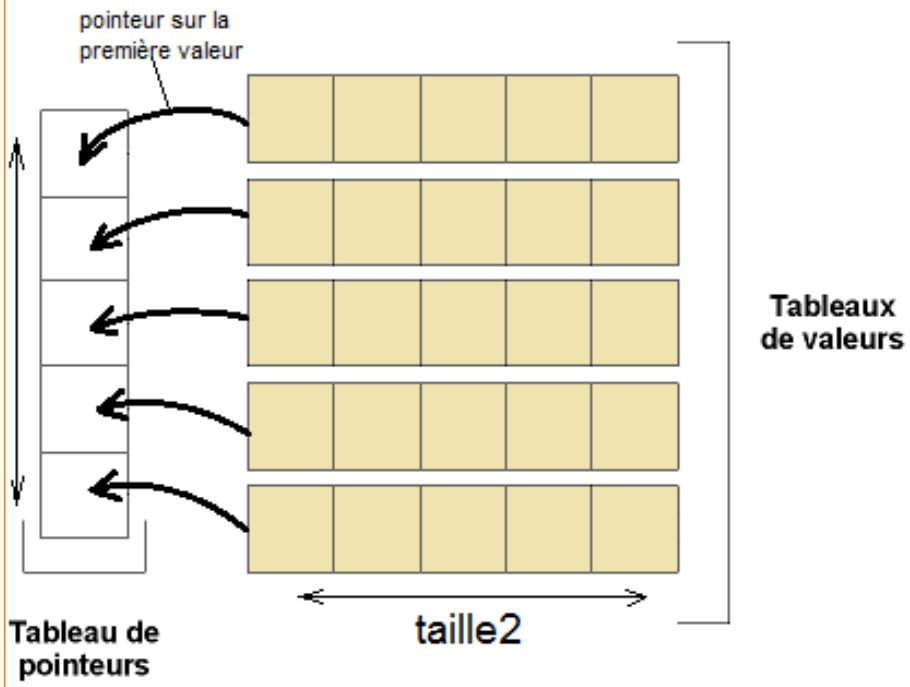
Allocation dynamique d'une matrice

Allocation

L'allocation doit se faire en deux étapes :

Etape 2 : Allocation de la dexième dimension (tableaux de valeurs)

```
int i;

//On alloue des tableaux de 'taille2' variables
for(i=0 ; i < taille1 ; i++){
   ptr[i] = (int *) malloc(taille2 * sizeof(int**) );
 //ptr[i] = (int *) malloc(taille2 * sizeof(*(ptr[i])));
 //ptr[i] = (int *) malloc(taille2 * sizeof(**ptr) );
    if(ptr[i] == NULL){
       //libérer la mémoire déjà allouée
       //message d'erreur et quitter le programme
    }
}
```



Allocation dynamique d'une matrice

Allocation

Allocation d'un tableau ptr[2][3] :

```
int i , taille1 = 2 , taille2 = 3;
int **ptr;

ptr = (int **)  malloc(taille1 * sizeof(*ptr));
if(ptr == NULL)
     exit(0);

for(i=0 ; i < taille1 ; i++){
    ptr[i] = (int *)  malloc(taille2 * sizeof(**ptr) );
    if(ptr[i] == NULL){
       //libérer la mémoire déjà allouée
       exit(0);
 }
}
```

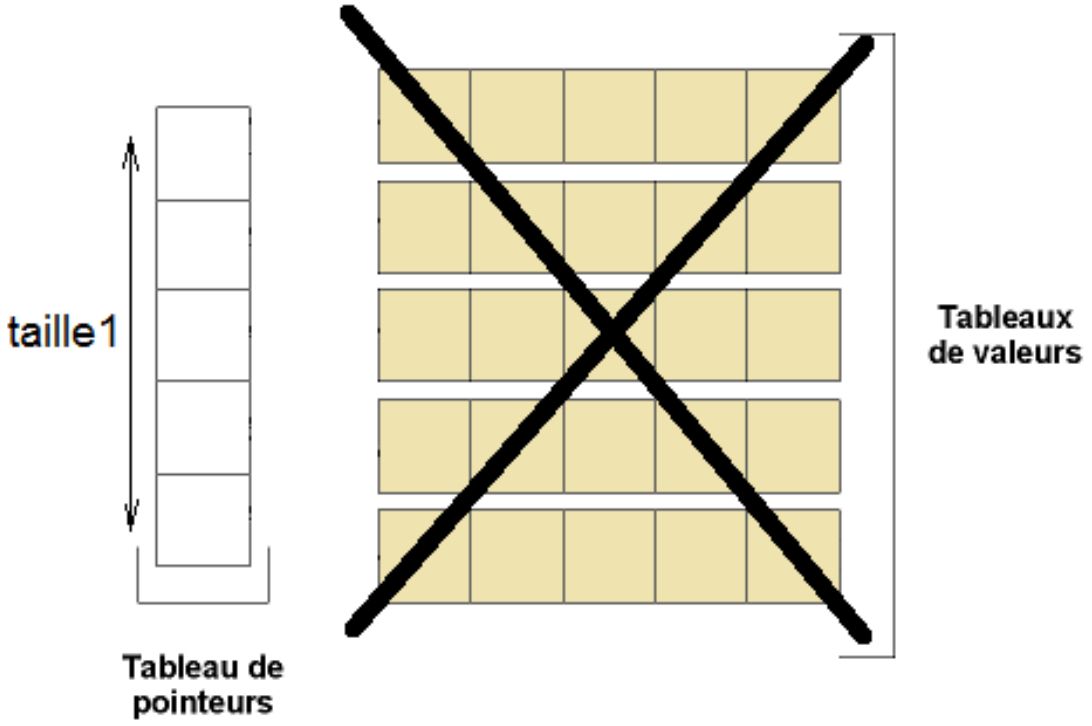
Allocation dynamique d'une matrice

Libération

La libération de mémoire se fait suivant l'ordre inverse à l'allocation

Etape 1 : Libération de la deuxième dimension (*ptr)

```
int i;

for(i=0 ; i < taille1 ; i++){
    free(ptr[i]);
}
```



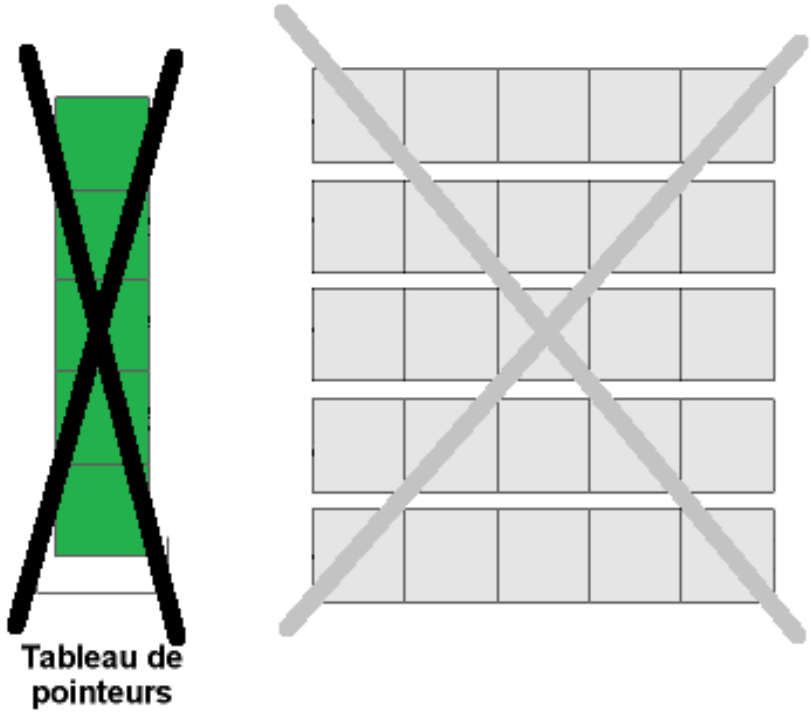
Allocation dynamique d'une matrice

Libération

La libération de mémoire se fait suivant l'ordre inverse à l'allocation

Etape 1 Libération de la première dimension (ptr)

```
int i;

for(i=0 ; i < taille1 ; i++){
    free(ptr[i]);
}

free(ptr);
```



Allocation dynamique d'une matrice

La libération n'est utilisable que si l'allocation s'est bien déroulé

```
ptr = (int **)  malloc(taille1 * sizeof(*ptr));
if(ptr == NULL)
     //pas de libération à ce niveau
     exit(0);
for(i=0 ; i < taille1 ; i++){
    ptr[i] = (int *)  malloc(taille2 * sizeof(**ptr) );
    if(ptr[i] == NULL){
       //On parcourt la boucle dans l'ordre inverse pour libérer ce qui a déjà été alloué
       for(i = i-1 ; i >= 0 ; i--)
              free(ptr[i]);
       //On libère la première dimension
       free(ptr);
       exit(0);
 }
}
```

Exemple : saisie et affichage d'une matrice allouée dynamiquement

```
int **A;
int N, M,i;
int **P1, *P2;
printf("Nombre de lignes : ");
scanf("%d", &N );
printf("Nombre de colonnes : ");
scanf("%d", &M );

A= (int **)  malloc(N* sizeof(*A));
for(i=0 ; i < N; i++)
    A[i] = (int *)  malloc(M * sizeof(**A) );

for(P1=A; P1<A+N; P1++)
    for (P2=*P1; P2<*P1+M; P2++)
    {
        printf("A[%d][%d] = ",P1-A,P2-*P1);
        scanf("%d",P2);
    }

for(P1=A; P1<A+N; P1++)
{
    for (P2=*P1; P2<*P1+M; P2++)
        printf("%d\t",*P2);
    printf("\n");
}
```