Programmation C

Fabien Calcado
fabien.calcado@isty.uvsq.fr

Mise à disposition de mon cours de 1ère année dans le cadre des rappels / compléments de C

- **Rappels – 3 modes d'allocation**
  - Avantages / inconvénients
- **Fonctions liées à la gestion dynamique de la mémoire**
  - Malloc, calloc, realloc, free
  - Ex. d'allocation dynamique : variable, tableau 1D et 2D

- **Rappels – 3 modes d'allocation (cf. cours n°1)**
  - Allocation statique (variable globale)
    - **Zone mémoire réservée avant l'exécution du programme**
      - » Accessible au chargement du programme
      - » Située dans des segments spécifiques du binaire selon l'utilisation
    - **Avantages**
      - » Quantité mémoire constante et connue avant l'exécution
      - » Pas de surcoût durant l'exécution pour la rendre utilisable
      - » Accessible et modifiable de n'importe où
    - **Inconvénients**
      - » Rigide : taille doit être connue à la compilation (et non modifiable)
      - » Accessible et modifiable de n'importe où…
        - ➔risques d'erreurs difficiles à corriger
    - **Utilisation**
      - » A privilégier pour des constantes/valeurs connues à la compilation

- **Rappels – 3 modes d'allocation (cf. cours n°1)**
  - Allocation automatique (variable locale)
    - **Zone mémoire réservée / libérée pendant l'exécution**
      - » Située dans un environnement spécifique (pile) à un bloc / fonction
        - » réservation à l'entrée et libération (automatique) à la sortie
    - **Avantages**
      - » Quantité mémoire connue avant l'exécution par une analyse du code
      - » Libération implicite pour le programmeur ➔ pas de fuite mémoire
    - **Inconvénients**
      - » Rigide : taille doit être connue à la compilation (et non modifiable)
        - » Utilisation + ou - excessive de mémoire par rapport aux besoins
    - **Utilisation**
      - » A privilégier lorsque les besoins mémoires sont connus à l'avance
      - » Prendre garde à la quantité de mémoire utilisée pour certaines implémentations (Ex. : récursivité)

Rappels – 3 modes d'allocation (cf. cours n°1)

- Allocation dynamique
  - **Zone mémoire réservée / libérée pendant l'exécution**
    - » Située dans un environnement spécifique (tas)
  - **Avantages**
    - » Flexible : taille définie à l'exécution et peut être modifiée par la suite si nécessaire (création d'une nouvelle zone)
      - » Utilisation mémoire en adéquation avec les besoins
    - » Zone reste accessible / modifiable par l'intermédiaire de son adresse
  - **Inconvénients**
    - » Programmeur a la responsabilité de la libération ➔ /!\ fuites mémoires
  - **Utilisation**
    - » Lorsque la taille n'est pas connue à la compilation
      - » Ex. : taille d'un tableau donnée par l'utilisateur
    - » Lorsqu'on souhaite utiliser une zone en dehors d'un bloc / fonction
      - » Ex. : création d'un tableau dans une fonction, utilisée en dehors de celle-ci

Fonctions de la bibliothèque standard

- Allouer une zone mémoire dynamiquement
  - **malloc( )**
    - » Allocation d'une zone mémoire sans initialisation
  - **calloc( )**
    - » Allocation d'une zone mémoire initialisée à zéro
  - **realloc( )**
    - » Changer la taille d'une zone précédemment allouée
- Désallouer une zone mémoire dynamique
  - **free( )**
    - » Libérer une zone mémoire précédemment allouée (dynamiquement)

Fonction « malloc »

- Prototype

```
void * malloc( size_t  size )
```

Adresse du bloc alloué (pointeur universel) — Taille du bloc à allouer (en octets)

- Permet de réserver un bloc mémoire
  - **Aucune initialisation ➔ contenu indéterminé**
  - **Renvoie l'adresse du bloc alloué**
    - » Valeur de type « pointeur universel » (indépendant d'un type particulier)
    - » Valeur renvoyée « NULL » si l'allocation échoue

Fonction « calloc »

- Prototype

```
void * calloc(size_t nmemb, size_t size) ;
```

Adresse du bloc alloué (pointeur universel) — Nombre d'éléments — Taille d'un élément (en octets)

- Permet de réserver un bloc mémoire de taille « nmemb*size »
  - **Zone remplie avec des zéros (au niveau binaire)**
    - » Attention à la représentation de la « valeur zéro » suivant les types
      - ➔ utile que pour les entiers et chaînes de caractères
  - **Renvoie l'adresse du bloc alloué**
    - » Valeur de type « pointeur universel » (indépendant d'un type particulier)
    - » Valeur renvoyée « NULL » si l'allocation échoue

Gestion dynamique de la mémoire

Fonction « realloc »

– Prototype

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

Adresse du bloc alloué (pointeur universel)

Nouvelle taille du bloc (en octets)

– Permet de modifier la taille du bloc mémoire pointé par « ptr »

- **Conserve le contenu de l'ancienne zone mémoire allouée dynamiquement**
  - » Si la nouvelle taille est supérieur à l'ancienne, le surplus n'est pas initialisé
  - » Si « ptr » est NULL l'appel est identique à « malloc(size) »
- **Renvoie l'adresse du bloc alloué**
  - » Valeur de type « pointeur universel » (indépendant d'un type particulier)
  - » Valeur renvoyée « NULL » si l'allocation échoue

Gestion dynamique de la mémoire

Fonction « free »

– Prototype

```
void  free(void *ptr)
```

Adresse de la zone à libérer

– Permet de libérer la zone mémoire pointée par « ptr »

- **La zone doit avoir été allouée par un appel à malloc, calloc ou realloc**
  - » Dans le cas contraire ou si « ptr » a déjà été libéré : comportement indéterminé
- **Si « ptr » est NULL : pas de tentative de libération**

Gestion dynamique de la mémoire

Allocation / libération d'une variable

```
int* p = NULL;
int a = 42;

p = (int*)malloc(sizeof(int));

if (p == NULL){
    printf("Échec malloc\n");
    return EXIT_FAILURE;
}

*p = a;
printf(" a=%d ; &a=%8p         \n",a ,&a);
printf("*p=%d ; &p=%8p ; p=%8p\n",*p ,&p ,p);

free(p);
p = NULL;
```

```
 a=42 ; &a=0028FF08
*p=42 ; &p=0028FF0C ; p=004D1708
```


Gestion dynamique de la mémoire

Allocation / libération d'un tableau 1D

```
char* tab1D = NULL;
unsigned int t_utile = 0, i = 0;

do
{
    printf("\n\nTaille du tableau :");
    scanf(" %d", &t_utile);
}while(t_utile <= 0);

tab1D = (char*)malloc(t_utile*sizeof(char));

if (tab1D == NULL){
    printf("Échec malloc\n");
    return EXIT_FAILURE;
}

for(i=0;i<t_utile;i++)
    printf("%c ",tab1D[i]);

free(tab1D);
tab1D = NULL;
```

```
Taille du tableau :15
▶ 4 ┤   ↕ ┤   n d o w s
```

Allocation / libération d'un tableau 1D (calloc / realloc)

```
int* tab1D = NULL, *tmp = NULL;
unsigned int t_utile = 0, i = 0;

do
{
    printf("\nTaille du tableau :");
    scanf(" %d", &t_utile);
}while(t_utile <= 0);

tab1D = (int*)calloc(t_utile,sizeof(int));
if (tab1D == NULL){
    printf("Echec calloc\n");
    return EXIT_FAILURE;
}

for(i=0;i<t_utile;i++)
    printf("%d ",tab1D[i]);
```

```
Taille du tableau :4
0 0 0 0
```

(suite prochaine diapo)

Allocation / libération d'un tableau 1D (calloc / realloc)

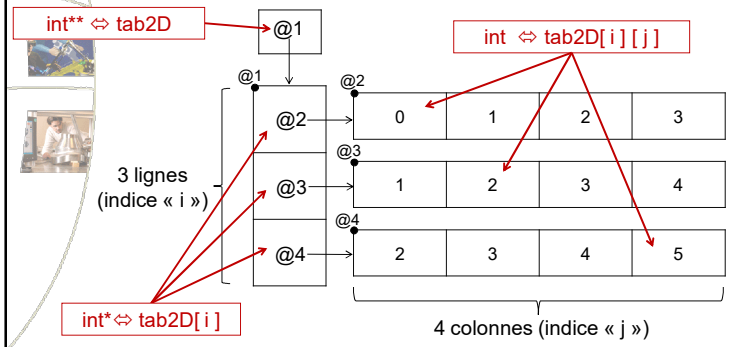
```
do
{
    printf("\nNouvelle taille du tableau :");
    scanf(" %d", &t_utile);
}while(t_utile <= 0);

tmp = realloc(tab1D, t_utile);
if (tab1D == NULL){
    free(tab1D)
    printf("Echec realloc\n");
    return EXIT_FAILURE;
}
tab1D = tmp;

for(i=0;i<t_utile;i++)
    printf("%d ",tab1D[i]);

free(tab1D);
tab1D = NULL;
```

L'utilisation de « tmp » est impératif sinon la libération n'aurait pas pu se faire en cas d'échec ➔ tab1D aurait été NULL…

```
Taille du tableau :4
0 0 0 0
Nouvelle taille du tableau :8
0 0 0 0 2088064460 134257516 2 1252
```


Représentation mémoire d'un tableau 2D

– Exemple sur un tableau 2D d'entiers



Etapes pour allouer un tableau 2D dynamique

– Créer un pointeur sur le tableau 2D
- **Type pointeur de pointeur : stocke l'adresse de l'adresse d'un tableau (➔ adresse d'un tableau de pointeur)**

– Créer un tableau dont chacune des cases est de type pointeur
- **Taille du tableau : nombre de lignes du tableau 2D**
- **Contenu de la case « i » = adresse de la première case du tableau représentant la ligne « i »**

– Créer un ensemble de tableaux contenant les valeurs du tableau 2D
- **1 tableau pour chacune des lignes**
- **Taille de chaque tableau : nombre de colonnes**
- **Chaque case contient la valeur à la position « i , j » du tableau 2D**

Gestion dynamique de la mémoire

Allocation / libération d'un tableau 2D

```
int** tab2D = NULL;
unsigned int dim1 = 0, dim2 = 0, i = 0, j = 0;

printf("dim1 & dim2 :");
scanf(" %d %d",&dim1 ,&dim2);

tab2D = (int**)malloc(dim1*sizeof(int*));
for(i=0;i<dim1;i++)
    tab2D[i] = (int*)malloc(dim2*sizeof(int));

for(i=0;i<dim1;i++){
    for(j=0;j<dim2;j++){
        tab2D[i][j]=i+j;
        printf(" %d", tab2D[i][j]);
    }
    printf("\n");
}
```

```
dim1 & dim2 :3 4
 0 1 2 3
 1 2 3 4
 2 3 4 5
```

```
for(i=0;i<dim1;i++)
    free(tab2D[i]);
free(tab2D);
tab2D = NULL;
```

→ La libération doit se faire dans l'ordre inverse de l'allocation

Gestion dynamique de la mémoire

Utiliser une zone mémoire en dehors d'une fonction

```
int main(void)
{
    unsigned int taille = 0;
    int* tab1D = NULL;

    printf("Taille de votre tableau : ");
    scanf(" %d", &taille);

    tab1D = allouer_tab1D_int(taille);
    remplir_tab1D_int(tab1D, taille);
    afficher_tab1D_int(tab1D, taille);
    liberer_tab1D_int(&tab1D);

    return 0;
}
```

```
int* allouer_tab1D_int(unsigned int t)
{
    int* tab = (int*)malloc(t*sizeof(int));
    return tab;
}
```

Env. du main

taille	tab1D
0	NULL

Gestion dynamique de la mémoire

Utiliser une zone mémoire en dehors d'une fonction

```
int main(void)
{
    unsigned int taille = 0;
    int* tab1D = NULL;

    printf("Taille de votre tableau : ");
    scanf(" %d", &taille);

    tab1D = allouer_tab1D_int(taille);
    remplir_tab1D_int(tab1D, taille);
    afficher_tab1D_int(tab1D, taille);
    liberer_tab1D_int(&tab1D);

    return 0;
}
```

```
int* allouer_tab1D_int(unsigned int t)
{
    int* tab = (int*)malloc(t*sizeof(int));
    return tab;
}
```

Env. du main

taille	tab1D
5	NULL

Gestion dynamique de la mémoire

Utiliser une zone mémoire en dehors d'une fonction

```
int main(void)
{
    unsigned int taille = 0;
    int* tab1D = NULL;

    printf("Taille de votre tableau : ");
    scanf(" %d", &taille);

    tab1D = allouer_tab1D_int(taille);
    remplir_tab1D_int(tab1D, taille);
    afficher_tab1D_int(tab1D, taille);
    liberer_tab1D_int(&tab1D);

    return 0;
}
```

```
int* allouer_tab1D_int(unsigned int t)
{
    int* tab = (int*)malloc(t*sizeof(int));
    return tab;
}
```

Env. du main

taille	tab1D
5	NULL

Env. allouer_tab1D

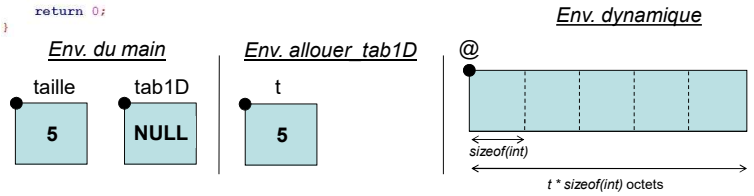
t
5

Gestion dynamique de la mémoire

Utiliser une zone mémoire en dehors d'une fonction

```
int main(void)
{
    unsigned int taille = 0;
    int* tab1D = NULL;

    printf("Taille de votre tableau : ");
    scanf(" %d", &taille);

    tab1D = allouer_tab1D_int(taille);
    remplir_tab1D_int(tab1D, taille);
    afficher_tab1D_int(tab1D, taille);
    liberer_tab1D_int(&tab1D);

    return 0;
}
```

```
int* allouer_tab1D_int(unsigned int t)
{
    int* tab = (int*)malloc(t*sizeof(int));
    return tab;
}
```

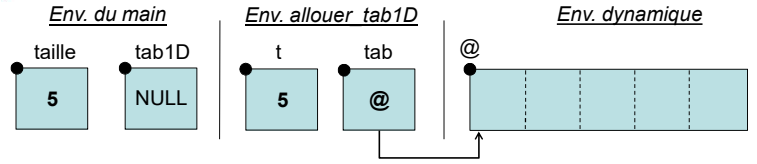


Gestion dynamique de la mémoire

Utiliser une zone mémoire en dehors d'une fonction

```
int main(void)
{
    unsigned int taille = 0;
    int* tab1D = NULL;

    printf("Taille de votre tableau : ");
    scanf(" %d", &taille);

    tab1D = allouer_tab1D_int(taille);
    remplir_tab1D_int(tab1D, taille);
    afficher_tab1D_int(tab1D, taille);
    liberer_tab1D_int(&tab1D);

    return 0;
}
```

```
int* allouer_tab1D_int(unsigned int t)
{
    int* tab = (int*)malloc(t*sizeof(int));
    return tab;
}
```



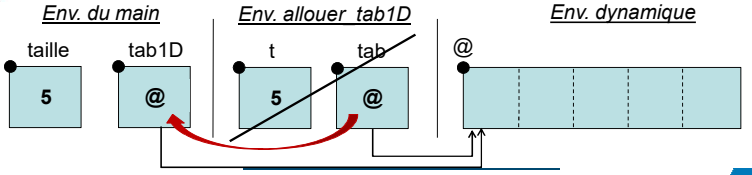
Gestion dynamique de la mémoire

Utiliser une zone mémoire en dehors d'une fonction

```
int main(void)
{
    unsigned int taille = 0;
    int* tab1D = NULL;

    printf("Taille de votre tableau : ");
    scanf(" %d", &taille);

    tab1D = allouer_tab1D_int(taille);
    remplir_tab1D_int(tab1D, taille);
    afficher_tab1D_int(tab1D, taille);
    liberer_tab1D_int(&tab1D);

    return 0;
}
```

```
int* allouer_tab1D_int(unsigned int t)
{
    int* tab = (int*)malloc(t*sizeof(int));
    return tab;
}
```

Env. dynamique n'est pas libéré en sortant de la fonction

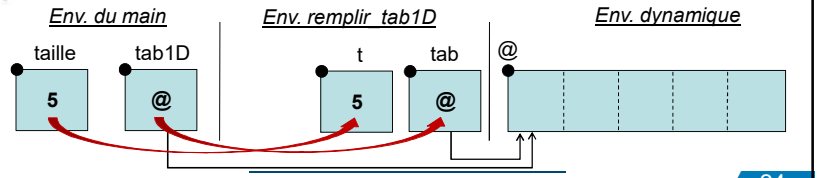


Gestion dynamique de la mémoire

Utiliser une zone mémoire en dehors d'une fonction

```
int main(void)
{
    unsigned int taille = 0;
    int* tab1D = NULL;

    printf("Taille de votre tableau : ");
    scanf(" %d", &taille);

    tab1D = allouer_tab1D_int(taille);
    remplir_tab1D_int(tab1D, taille);
    afficher_tab1D_int(tab1D, taille);
    liberer_tab1D_int(&tab1D);

    return 0;
}
```

```
void remplir_tab1D_int(int* tab, unsigned int t)
{
    unsigned int i = 0;

    for(i=0;i<t;i++)
        tab[i] = i;

    return;
}
```



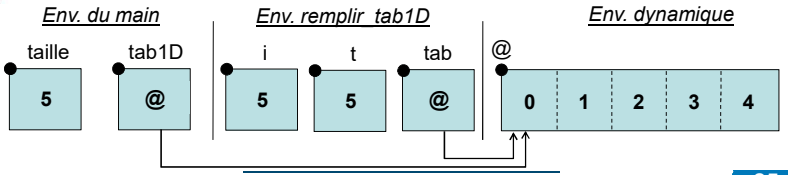
Gestion dynamique de la mémoire

Utiliser une zone mémoire en dehors d'une fonction

```
int main(void)
{
    unsigned int taille = 0;
    int* tab1D = NULL;

    printf("Taille de votre tableau : ");
    scanf(" %d", &taille);

    tab1D = allouer_tab1D_int(taille);
    remplir_tab1D_int(tab1D, taille);
    afficher_tab1D_int(tab1D, taille);
    liberer_tab1D_int(&tab1D);

    return 0;
}
```

```
void remplir_tab1D_int(int* tab, unsigned int t)
{
    unsigned int i = 0;

    for(i=0;i<t;i++)
        tab[i] = i;

    return;
}
```

*tab[i] ⇔ *(tab+i)*

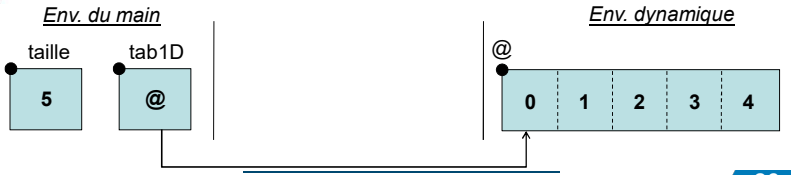


Gestion dynamique de la mémoire

Utiliser une zone mémoire en dehors d'une fonction

```
int main(void)
{
    unsigned int taille = 0;
    int* tab1D = NULL;

    printf("Taille de votre tableau : ");
    scanf(" %d", &taille);

    tab1D = allouer_tab1D_int(taille);
    remplir_tab1D_int(tab1D, taille);
    afficher_tab1D_int(tab1D, taille);
    liberer_tab1D_int(&tab1D);

    return 0;
}
```

```
void remplir_tab1D_int(int* tab, unsigned int t)
{
    unsigned int i = 0;

    for(i=0;i<t;i++)
        tab[i] = i;

    return;
}
```

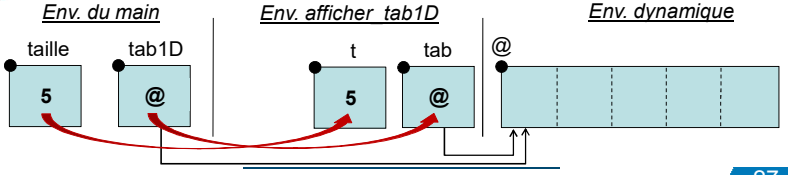


Gestion dynamique de la mémoire

Utiliser une zone mémoire en dehors d'une fonction

```
int main(void)
{
    unsigned int taille = 0;
    int* tab1D = NULL;

    printf("Taille de votre tableau : ");
    scanf(" %d", &taille);

    tab1D = allouer_tab1D_int(taille);
    remplir_tab1D_int(tab1D, taille);
    afficher_tab1D_int(tab1D, taille);
    liberer_tab1D_int(&tab1D);

    return 0;
}
```

```
void afficher_tab1D_int(int* tab, unsigned int t)
{
    unsigned int i = 0;

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

    return;
}
```



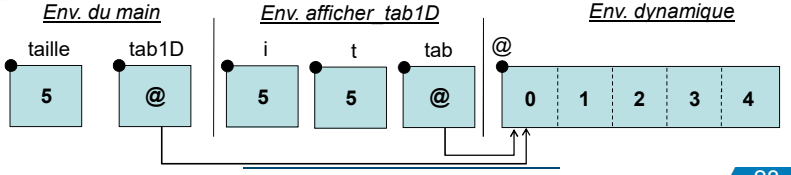
Gestion dynamique de la mémoire

Utiliser une zone mémoire en dehors d'une fonction

```
int main(void)
{
    unsigned int taille = 0;
    int* tab1D = NULL;

    printf("Taille de votre tableau : ");
    scanf(" %d", &taille);

    tab1D = allouer_tab1D_int(taille);
    remplir_tab1D_int(tab1D, taille);
    afficher_tab1D_int(tab1D, taille);
    liberer_tab1D_int(&tab1D);

    return 0;
}
```

```
void afficher_tab1D_int(int* tab, unsigned int t)
{
    unsigned int i = 0;

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

    return;
}
```

```
Taille de votre tableau : 5
tab[0]=0
tab[1]=1
tab[2]=2
tab[3]=3
tab[4]=4
```



Gestion dynamique de la mémoire

Utiliser une zone mémoire en dehors d'une fonction

```
int main(void)
{
    unsigned int taille = 0;
    int* tab1D = NULL;

    printf("Taille de votre tableau : ");
    scanf(" %d", &taille);

    tab1D = allouer_tab1D_int(taille);
    remplir_tab1D_int(tab1D, taille);
    afficher_tab1D_int(tab1D, taille);
    liberer_tab1D_int(&tab1D);

    return 0;
}
```

```
void afficher_tab1D_int(int* tab, unsigned int t)
{
    unsigned int i = 0;

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

    return;
}
```

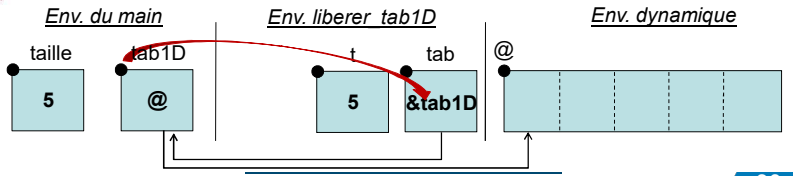
```
Taille de votre tableau : 5
tab[0]=0
tab[1]=1
tab[2]=2
tab[3]=3
tab[4]=4
```

Gestion dynamique de la mémoire

Utiliser une zone mémoire en dehors d'une fonction

```
int main(void)
{
    unsigned int taille = 0;
    int* tab1D = NULL;

    printf("Taille de votre tableau : ");
    scanf(" %d", &taille);

    tab1D = allouer_tab1D_int(taille);
    remplir_tab1D_int(tab1D, taille);
    afficher_tab1D_int(tab1D, taille);
    liberer_tab1D_int(&tab1D);

    return 0;
}
```

```
void liberer_tab1D_int(int** tab)
{
    free(*tab);
    *tab = NULL;

    return;
}
```



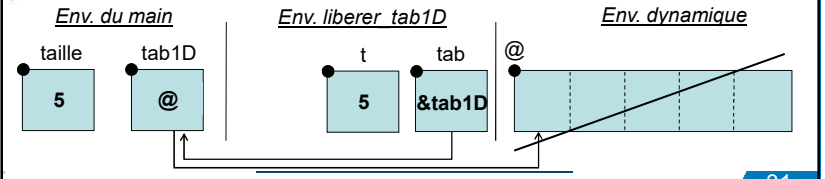
Gestion dynamique de la mémoire

Utiliser une zone mémoire en dehors d'une fonction

```
int main(void)
{
    unsigned int taille = 0;
    int* tab1D = NULL;

    printf("Taille de votre tableau : ");
    scanf(" %d", &taille);

    tab1D = allouer_tab1D_int(taille);
    remplir_tab1D_int(tab1D, taille);
    afficher_tab1D_int(tab1D, taille);
    liberer_tab1D_int(&tab1D);

    return 0;
}
```

```
void liberer_tab1D_int(int** tab)
{
    free(*tab);
    *tab = NULL;

    return;
}
```

*Libère la zone à l'adresse « *tab » ➔ « @ »*



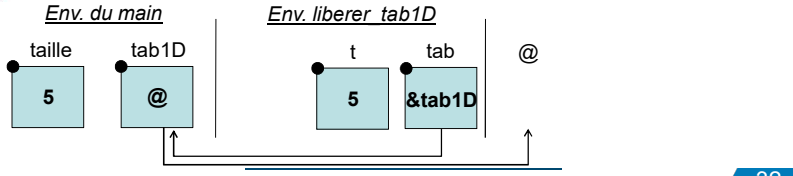
Gestion dynamique de la mémoire

Utiliser une zone mémoire en dehors d'une fonction

```
int main(void)
{
    unsigned int taille = 0;
    int* tab1D = NULL;

    printf("Taille de votre tableau : ");
    scanf(" %d", &taille);

    tab1D = allouer_tab1D_int(taille);
    remplir_tab1D_int(tab1D, taille);
    afficher_tab1D_int(tab1D, taille);
    liberer_tab1D_int(&tab1D);

    return 0;
}
```

```
void liberer_tab1D_int(int** tab)
{
    free(*tab);
    *tab = NULL;

    return;
}
```

Le pointeur « tab1D » pointe toujours sur « @ »
« tab1D » doit devenir non valide ➔ NULL



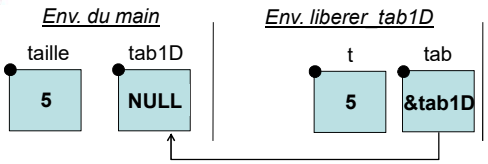
Gestion dynamique de la mémoire

Utiliser une zone mémoire en dehors d'une fonction

```
int main(void)
{
    unsigned int taille = 0;
    int* tab1D = NULL;

    printf("Taille de votre tableau : ");
    scanf(" %d", &taille);

    tab1D = allouer_tab1D_int(taille);
    remplir_tab1D_int(tab1D, taille);
    afficher_tab1D_int(tab1D, taille);
    liberer_tab1D_int(&tab1D);

    return 0;
}
```

```
void liberer_tab1D_int(int** tab)
{
    free(*tab);
    *tab = NULL;

    return;
}
```

On modifie « tab1D » par l'intermédiaire de « tab »
(par adressage indirect)

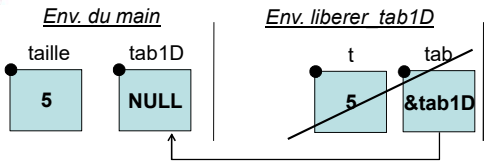


Gestion dynamique de la mémoire

Utiliser une zone mémoire en dehors d'une fonction

```
int main(void)
{
    unsigned int taille = 0;
    int* tab1D = NULL;

    printf("Taille de votre tableau : ");
    scanf(" %d", &taille);

    tab1D = allouer_tab1D_int(taille);
    remplir_tab1D_int(tab1D, taille);
    afficher_tab1D_int(tab1D, taille);
    liberer_tab1D_int(&tab1D);

    return 0;
}
```

```
void liberer_tab1D_int(int** tab)
{
    free(*tab);
    *tab = NULL;

    return;
}
```



Gestion dynamique de la mémoire

Utiliser une zone mémoire en dehors d'une fonction

```
int main(void)
{
    unsigned int taille = 0;
    int* tab1D = NULL;

    printf("Taille de votre tableau : ");
    scanf(" %d", &taille);

    tab1D = allouer_tab1D_int(taille);
    remplir_tab1D_int(tab1D, taille);
    afficher_tab1D_int(tab1D, taille);
    liberer_tab1D_int(&tab1D);

    return 0;
}
```