Langage C

Allocation dynamique de mémoire

#include<stdio.h>

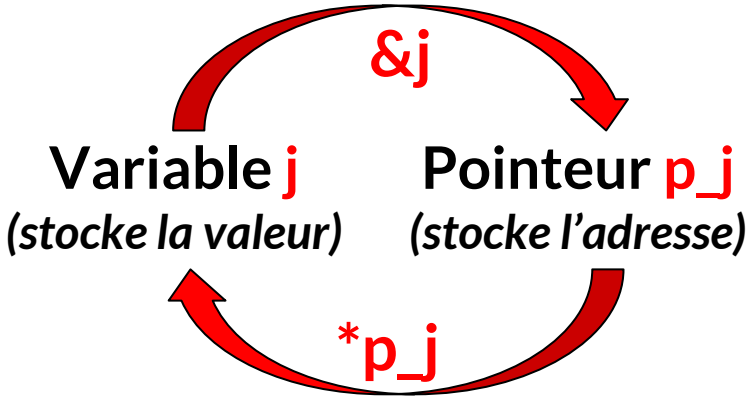
INSTITUT d'OPTIQUE GRADUATE SCHOOL ParisTech | LEnsE Laboratoire d'Enseignement Expérimental

http://www.twenty19.com

Sylvie LEBRUN / Charles BOURASSIN / Julien VILLEMEJANE

Les pointeurs

Pointeur : variable qui peut stocker l'adresse d'une autre variable



- Permet de manipuler des variables par leur adresse (unique dans la mémoire)

- Passage d'arguments par adresse

```
void Mafonction(int* p1, int* p2, ... );
```

=> Nombre illimité d'arguments de sortie

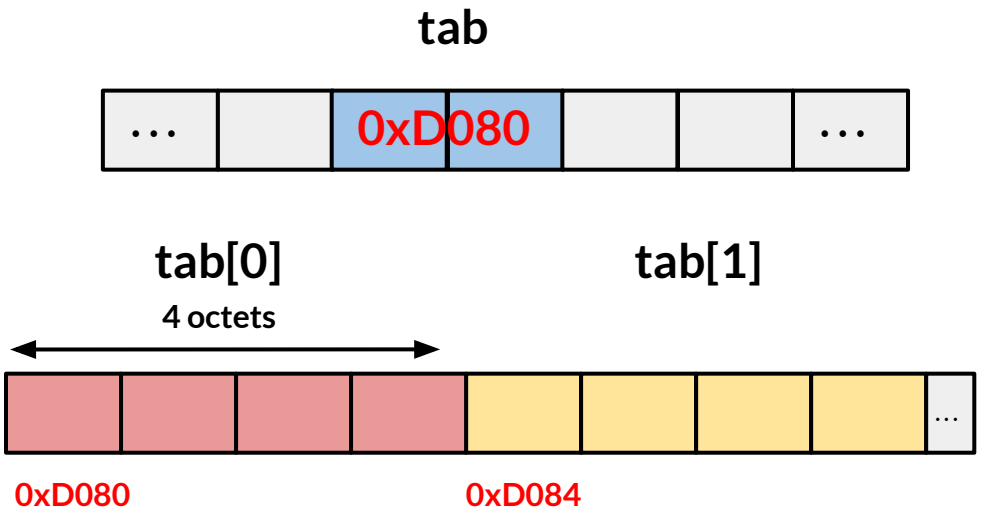
- Tableau = pointeur + bloc mémoire

tab[4] ⇔ *(tab + 4)

&tab[2] ⇔ tab + 2

Allocation statique de mémoire

```
int tab[5];
```

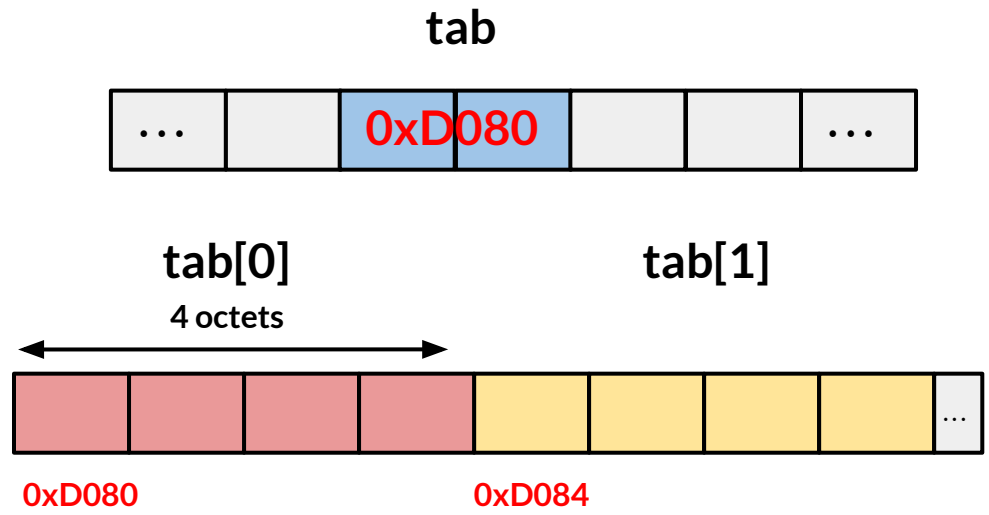


1. Définition d'un pointeur tab
2. Allocation d'un bloc mémoire de 5 x4octets contigües

tab contient l'adresse du 1er élément du tableau

Allocation statique de mémoire

```
int tab[5];
```



Rappel:

```
int dim ;
...
scanf("%d",&dim);
int tab[dim];
```

Ca ne marche pas

Il faut connaître la taille du tableau à la compilation

Problématique dans beaucoup de cas
(ex. Ouvrir une image PGM de taille inconnue…)

Allocation statique de mémoire

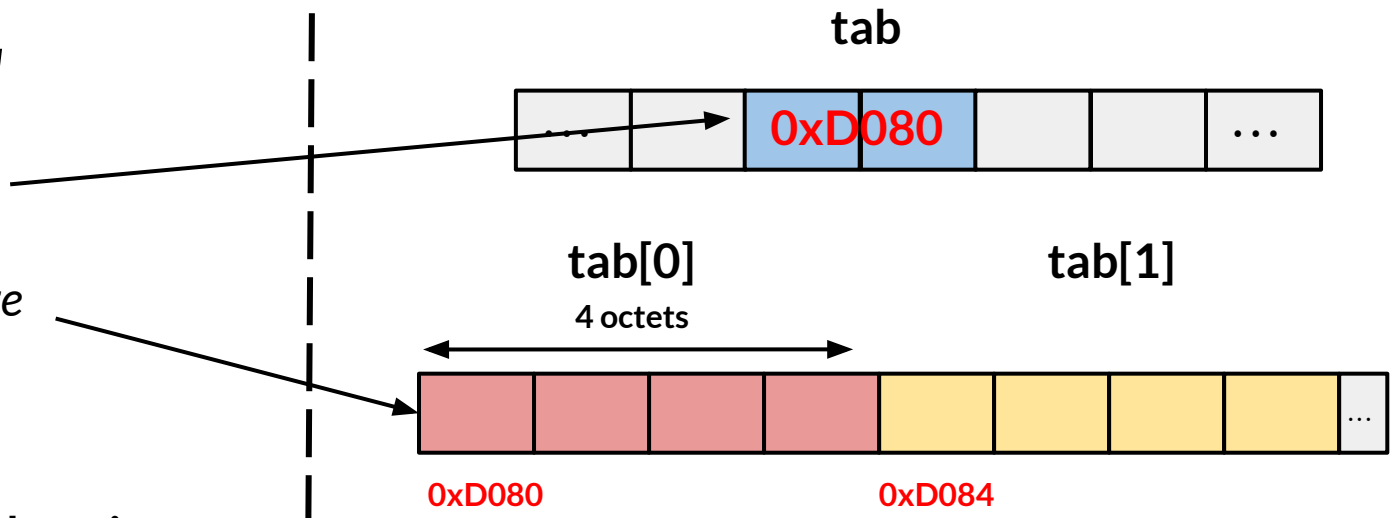
Un tableau

=

un pointeur

+

un bloc mémoire



On pourrait déclarer le pointeur (int* tab;)

Puis allouer le bloc mémoire quand on en a besoin

Il faut connaître la taille du tableau à la compilation

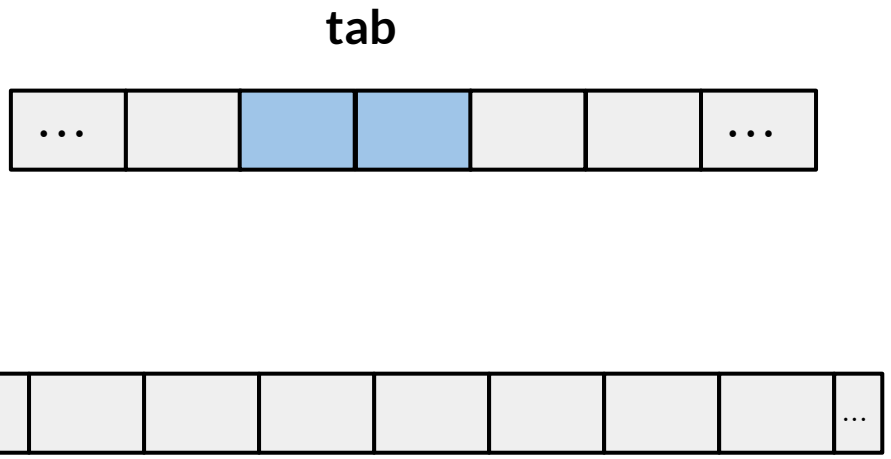
Problématique dans beaucoup de cas
(ex. Ouvrir une image PGM de taille inconnue…)

Allocation dynamique de mémoire

*On **ne** connaît **pas** la taille à la compilation*

```
int dim;

int* tab;

...

scanf("%d",&dim);

tab = malloc (dim*sizeof(int));

...

free(tab);
```

On ne connaît pas sa taille



Allocation d'un espace mémoire pour stocker l'adresse d'un int

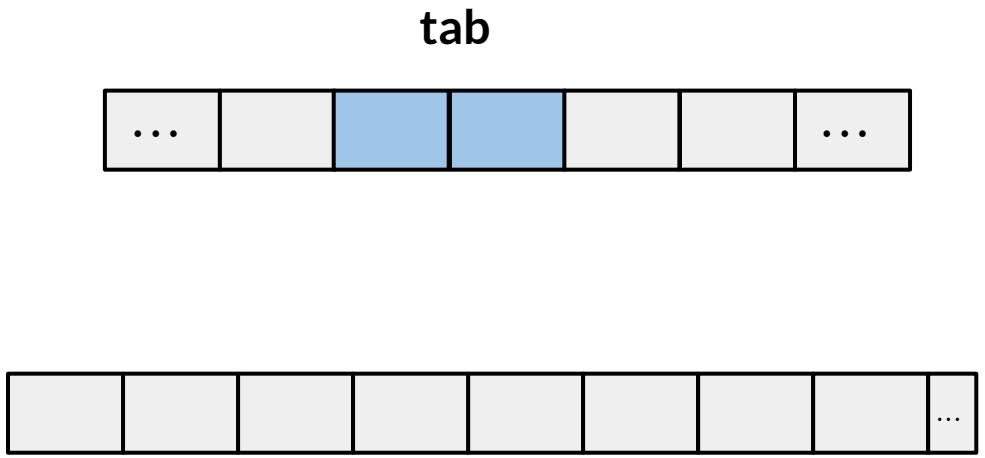
Allocation dynamique de mémoire

*On **ne** connait **pas** la taille à la compilation*

```
int dim;

int* tab;

...

scanf("%d",&dim);

tab = malloc (dim*sizeof(int));



...

free(tab);
```

dim = 5;

On connait la taille de tab



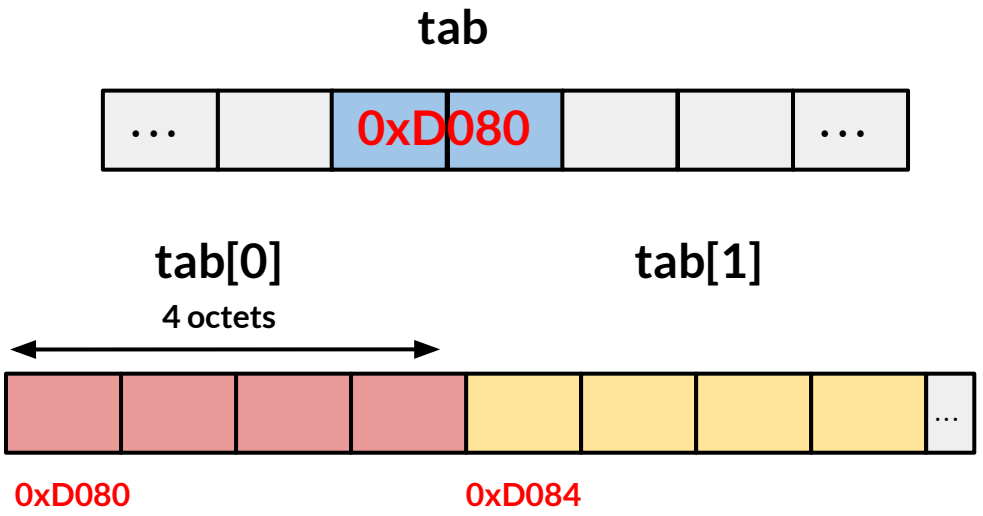
Allocation dynamique de mémoire

*On **ne** connait **pas** la taille à la compilation*

```
int dim;

int* tab;

...

scanf("%d",&dim);

tab = malloc (dim*sizeof(int));

...

free(tab);
```

dans <stdlib.h>

Nombre d'octets à allouer

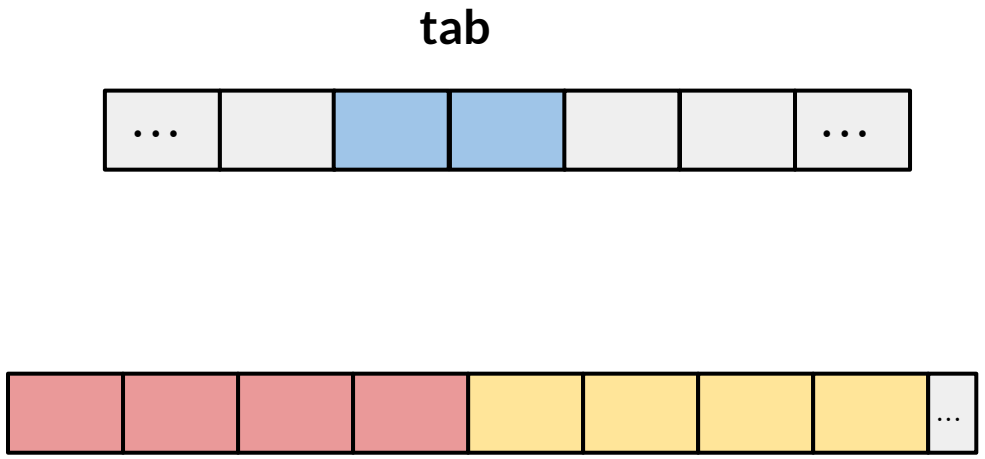


1. Allocation d'un bloc mémoire de 20 octets pour stocker 5 int
2. Ecriture de l'adresse de la 1ère case du bloc dans tab

Allocation dynamique de mémoire

*On **ne** connait **pas** la taille à la compilation*

```
int dim;

int* tab;

...

scanf("%d",&dim);

tab = malloc (dim*sizeof(int));



...

free(tab);
```



Libération de la mémoire allouée pour tab

Allocation dynamique de mémoire

	Nombre d'octets
char	1
int	2 ou 4
double	8

Un **int** *peut occuper 2 ou 4 octets selon les machines*
sizeof( ) *améliore la portabilité des programmes*

- **Si l'allocation échoue, malloc renvoie le pointeur NULL** *(adresse #0000 qui ne correspond en fait à aucune zone de mémoire accessible)*

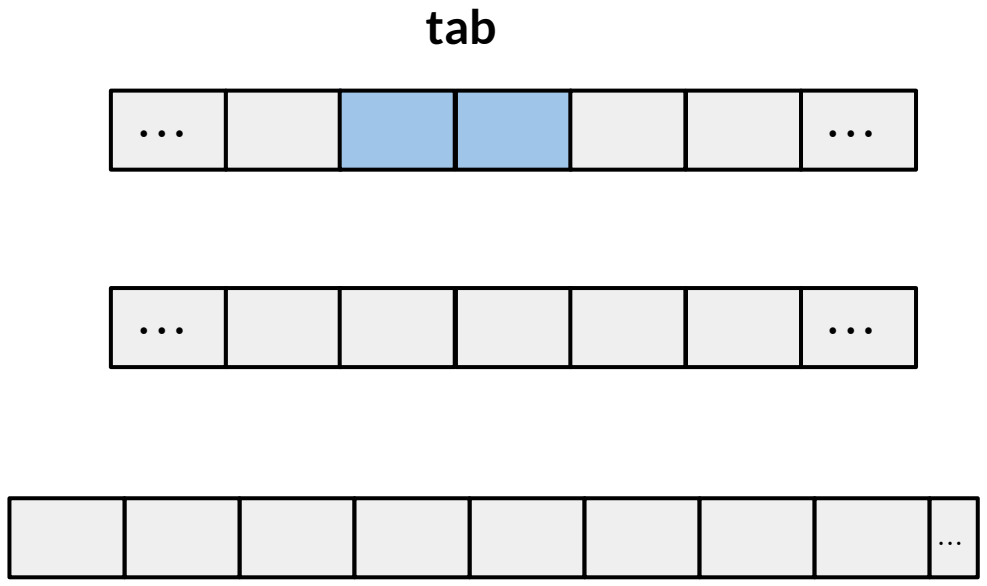
```
tab = malloc(dim*sizeof(double));

if ( tab != NULL) { ... }
else  printf("Allocation impossible\n");
```

- **Bonne pratique : libérer la mémoire allouée avec free(tab);** *(Essentiel lorsqu'on manipule de grands tableaux ou qu'on a peu de mémoire)*

Allocation dynamique dans les fonctions

Mauvaise méthode

```
void Mafonction(int* T, int dim);

void main(){
        int* tab;
        …
        dim = 5;
        Mafonction(tab, dim);
        printf("%d",tab[0]);
}

void Mafonction(int* T, int dim){
        int i;
        T=malloc(dim*sizeof(int));
        for (i=0;i<dim;i++) T[i]=0;
}
```

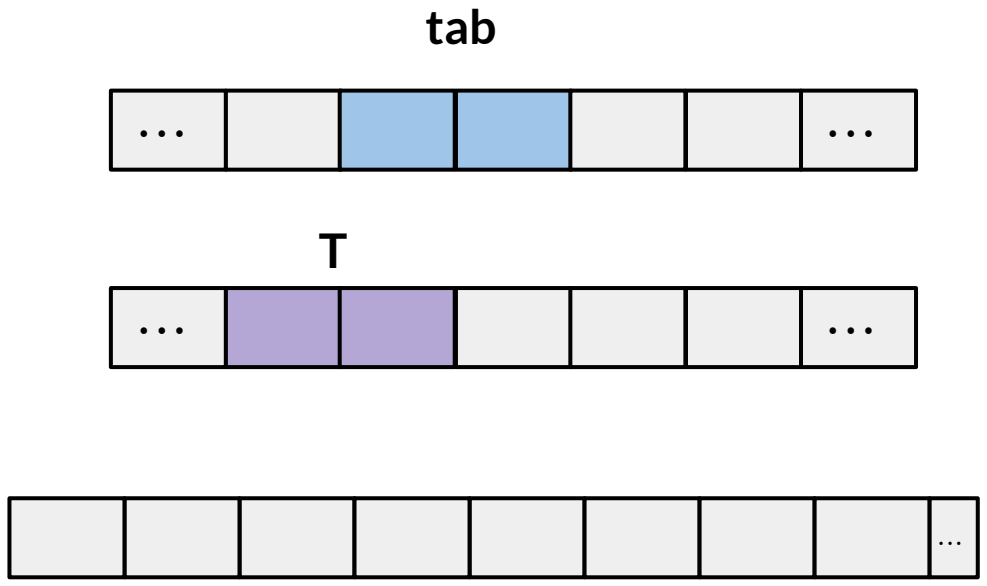


Allocation d'un espace mémoire pour stocker l'adresse d'un int

Allocation dynamique dans les fonctions

Mauvaise méthode

```
void Mafonction(int* T, int dim);

void main(){
        int* tab;
        …
        dim = 5;
        Mafonction(tab, dim);
        printf("%d",tab[0]);
}

void Mafonction(int* T, int dim){
        int i;
        T=malloc(dim*sizeof(int));
        for (i=0;i<dim;i++) T[i]=0;
}
```

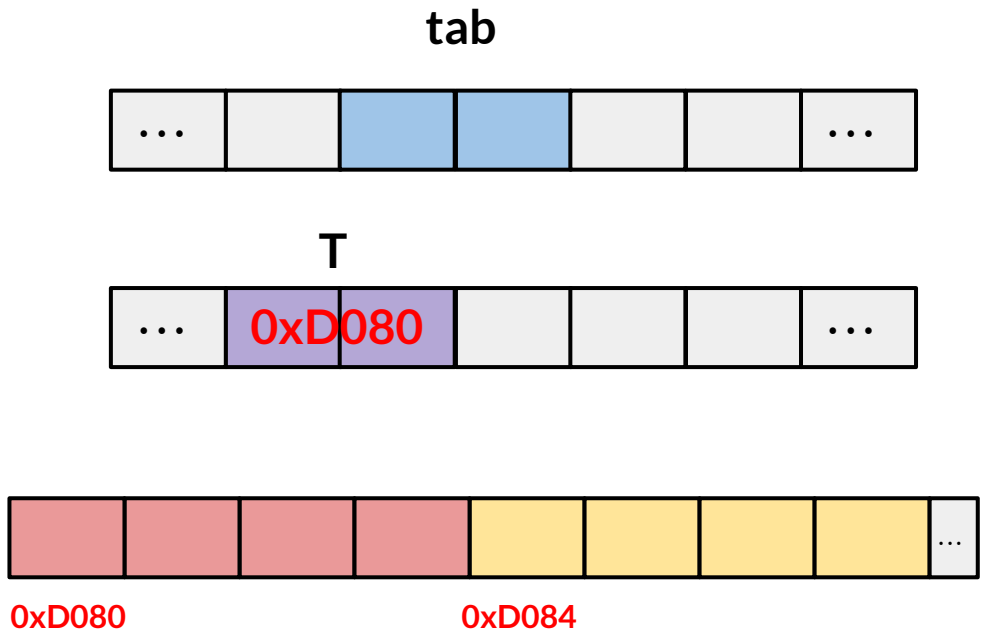


1. Création d'un pointeur T (variable locale)
2. Copie du contenu de tab dans T (mais il n'y a rien dans tab…)

Allocation dynamique dans les fonctions

Mauvaise méthode

```
void Mafonction(int* T, int dim);

void main(){
        int* tab;
        …
        dim = 5;
        Mafonction(tab, dim);
        printf("%d",tab[0]);
}

void Mafonction(int* T, int dim){
        int i;
        T=malloc(dim*sizeof(int));
        for (i=0;i<dim;i++) T[i]=0;
}
```

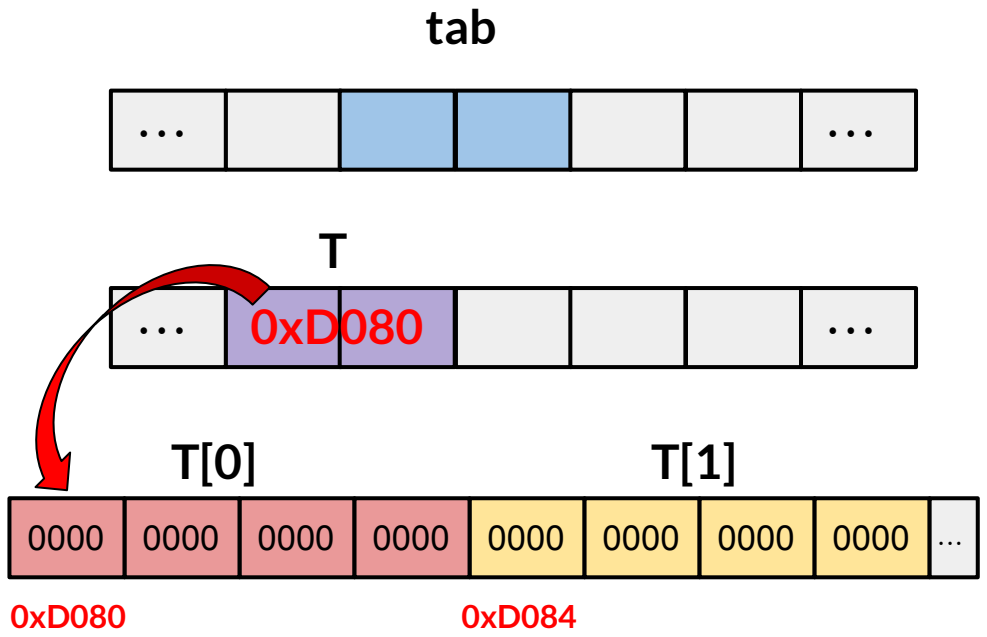


1. Allocation d'un bloc mémoire de 20 octets pour stocker 5 int
2. Ecriture de l'adresse de la 1ère case du bloc dans T

Allocation dynamique dans les fonctions

Mauvaise méthode

```
void Mafonction(int* T, int dim);

void main(){
        int* tab;
        ...
        dim = 5;
        Mafonction(tab, dim);
        printf("%d",tab[0]);
}

void Mafonction(int* T, int dim){
        int i;
        T=malloc(dim*sizeof(int));
        for (i=0;i<dim;i++) T[i]=0;
}
```

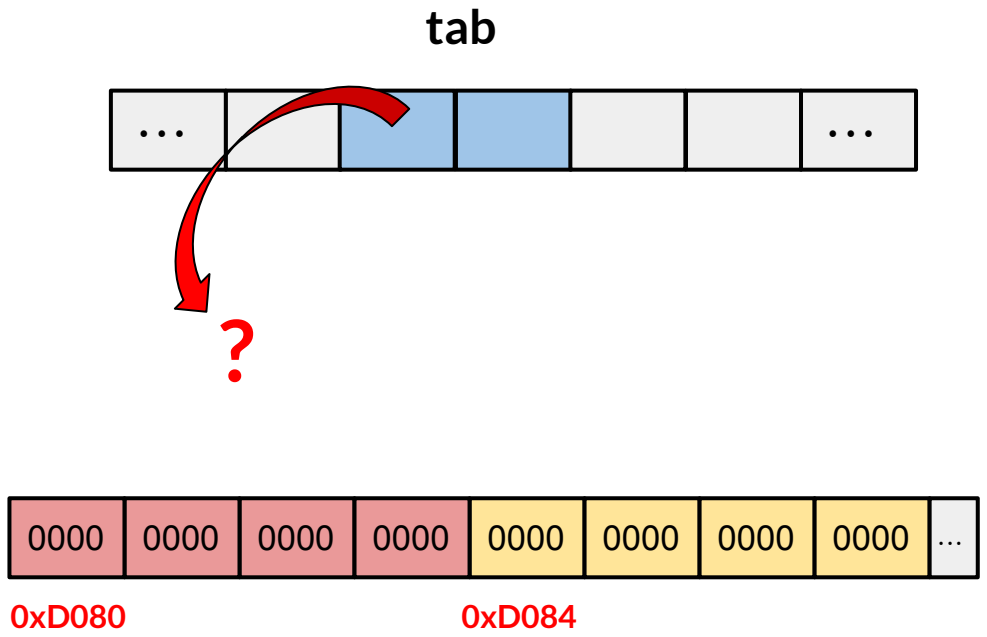


Lecture de l'adresse stockée dans T (0xD080) et écriture des 0 à partir de cette adresse

Allocation dynamique dans les fonctions

Mauvaise méthode

```
void Mafonction(int* T, int dim);

void main(){
        int* tab;
        ...
        dim = 5;
        Mafonction(tab, dim);
        printf("%d",tab[0]);
}

void Mafonction(int* T, int dim){
        int i;
        T=malloc(dim*sizeof(int));
        for (i=0;i<dim;i++) T[i]=0;
}
```



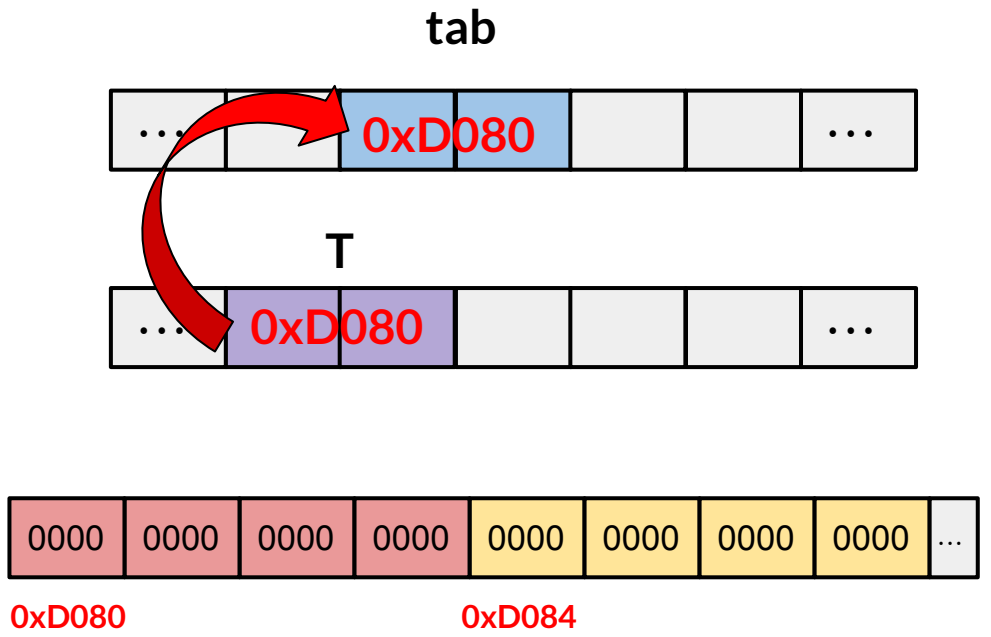
1. T (variable locale) est détruit à la fin de la fonction
2. tab ne contient aucune adresse

tab[0] ➡ Plantage à l'exécution

Allocation dynamique dans les fonctions

Bonne méthode 1

```
int*  Mafonction(int dim);

void main(){
        int* tab;
        …
        dim = 5;
        tab = Mafonction(dim);
        printf("%d",tab[0]);
}

int* Mafonction(int dim){
        int i; int* T;
        T=malloc(dim*sizeof(int));
        for (i=0;i<dim;i++) T[i]=0;
        return T;}
```



Solution : Passer T en argument de sortie

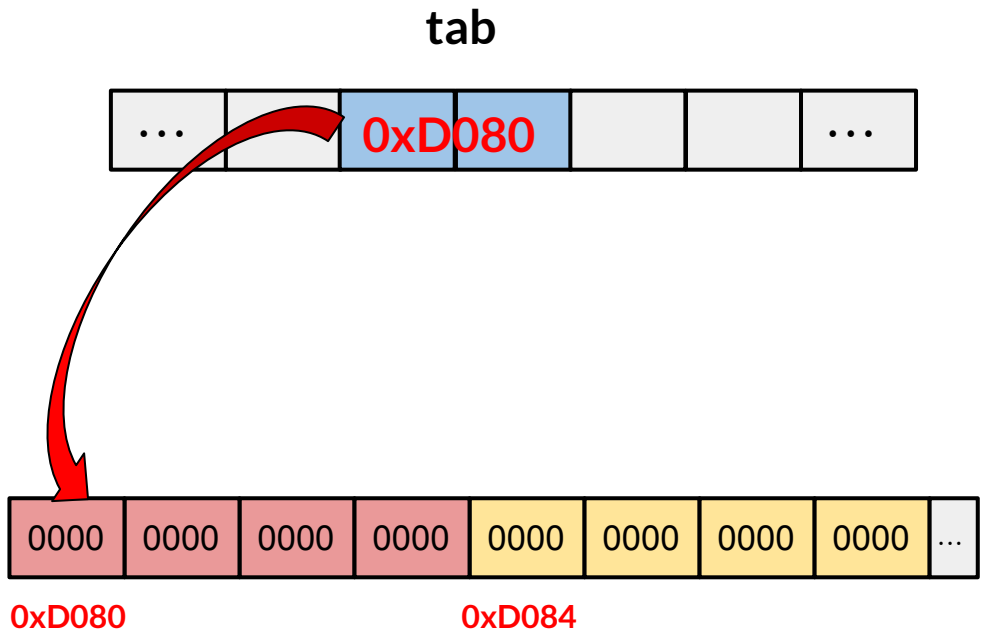
Copie de l'adresse contenue dans T (0xD080) dans tab

Allocation dynamique dans les fonctions

Bonne méthode 1

```
int*  Mafonction(int dim);

void main(){
        int* tab;
        …
        dim = 5;
        tab = Mafonction(dim);
        printf("%d",tab[0]);
}

int* Mafonction(int dim){
        int i; int* T;
        T=malloc(dim*sizeof(int));
        for (i=0;i<dim;i++) T[i]=0;
        return T;}
```

tab[0] = 0

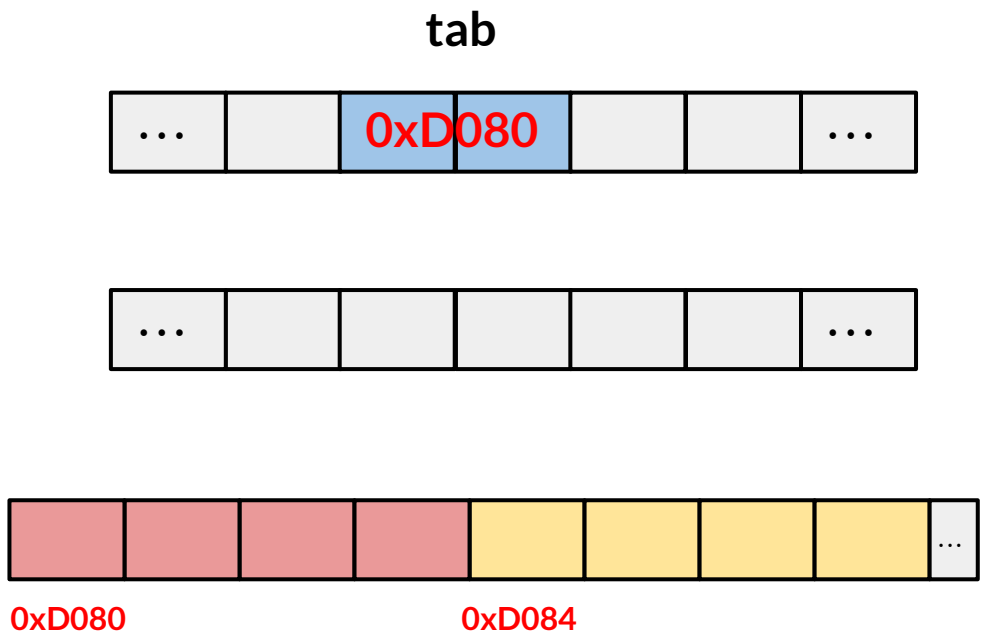


printf("%d",tab[0]) affiche 0

Allocation dynamique dans les fonctions

Bonne méthode 2 (malloc dans le main)

```
void Mafonction(int* T, int dim);

void main(){
        int* tab;
        ...
        dim = 5;
        tab=malloc(dim*sizeof(int));
        Mafonction(tab,dim);
        printf("%d",tab[0]);
}

void Mafonction(int* T, int dim){
        int i;
        for (i=0;i<dim;i++) T[i]=0;
}
```

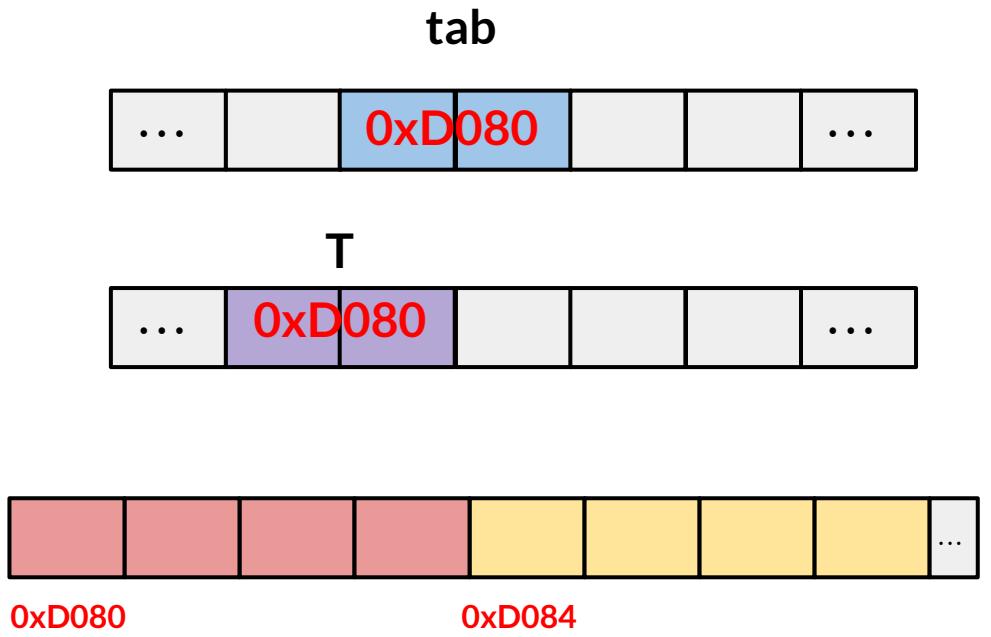


1. Allocation d'un bloc mémoire de 20 octets pour stocker 5 int
2. Ecriture de l'adresse de la 1ère case du bloc dans tab

Allocation dynamique dans les fonctions

Bonne méthode 2 (malloc dans le main)

```
void Mafonction(int* T, int dim);

void main(){
        int* tab;
        ...
        dim = 5;
        tab=malloc(dim*sizeof(int));
        Mafonction(tab,dim);
        printf("%d",tab[0]);
}

void Mafonction(int* T, int dim){
        int i;
        for (i=0;i<dim;i++) T[i]=0;
}
```

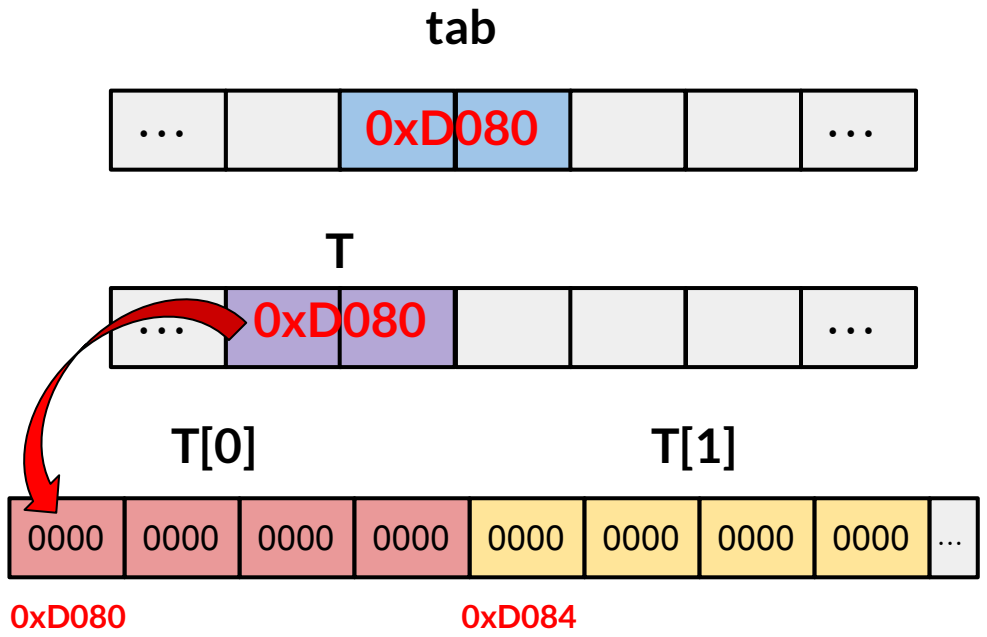


Création d'un pointeur T (variable locale) et écriture de l'adresse stockée dans tab

Allocation dynamique dans les fonctions

Bonne méthode 2 (malloc dans le main)

```
void Mafonction(int* T, int dim);

void main(){
        int* tab;
        ...
        dim = 5;
        tab=malloc(dim*sizeof(int));
        Mafonction(tab,dim);
        printf("%d",tab[0]);
}

void Mafonction(int* T, int dim){
        int i;
        for (i=0;i<dim;i++) T[i]=0;
}
```



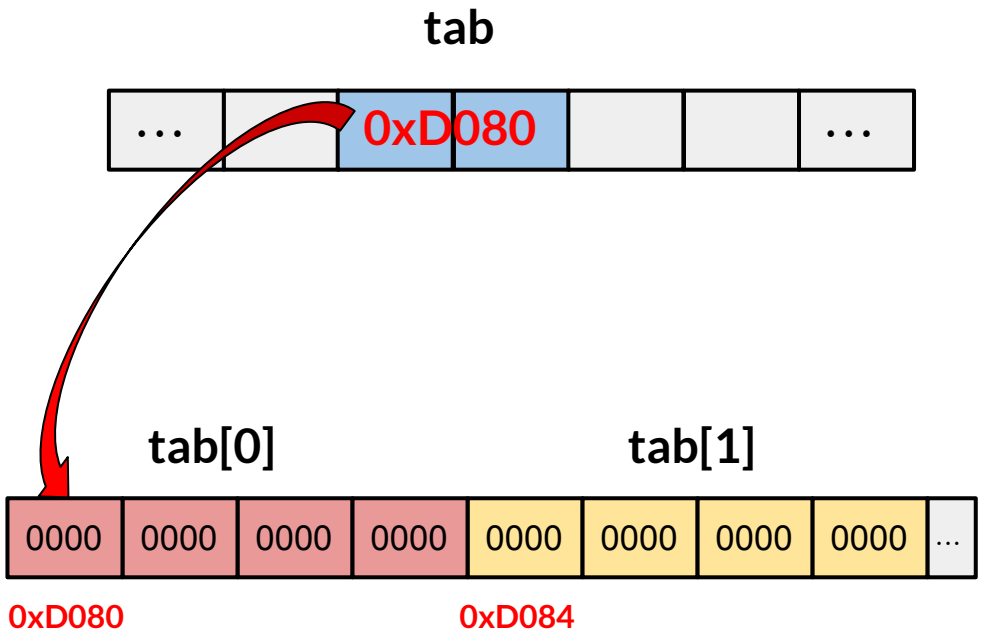
Lecture de l'adresse stockée dans T (0xD080) et écriture des 0 à partir de cette adresse

Allocation dynamique dans les fonctions

Bonne méthode 2 (malloc dans le main)

```
void Mafonction(int* T, int dim);

void main(){
        int* tab;
        ...
        dim = 5;
        tab=malloc(dim*sizeof(int));
        Mafonction(tab,dim);
        printf("%d",tab[0]);
}

void Mafonction(int* T, int dim){
        int i;
        for (i=0;i<dim;i++) T[i]=0;
}
```

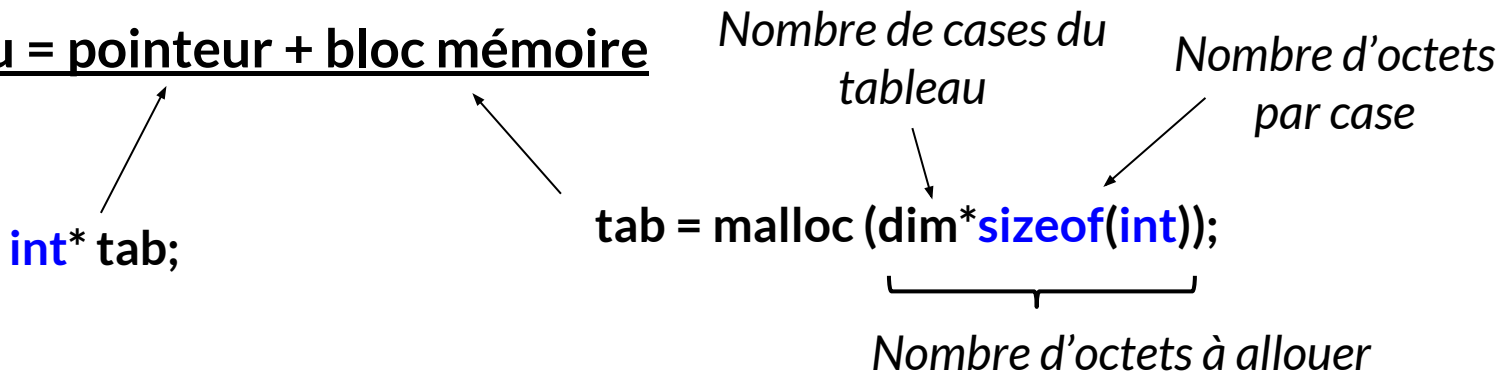
tab[0] = 0



1. T n'existe plus (variable locale)
2. Lecture de la valeur stockée à l'adresse tab

Allocation dynamique : plus besoin de connaître la taille d'un tableau à la compilation

- **Tableau = pointeur + bloc mémoire**



- *Si l'allocation échoue,* **tab = NULL**
- **free(tab);** *libère la mémoire allouée pour* **tab**

Allocation dynamique : plus besoin de connaître la taille d'un tableau à la compilation

- **Allocation dans les fonctions:**

Soit **tab = malloc(...)** *dans la fonction et* **return tab;**

Soit **tab = malloc(...)** *dans le* **main()** *et* **tab** *en argument d'entrée de la fonction*

- **Equivalence des notations:**

```
void Mafonction(int* tab, ...)
```

⟷

```
void Mafonction(int *tab, ...)
```

⟷

```
void Mafonction(int tab[], ...)
```