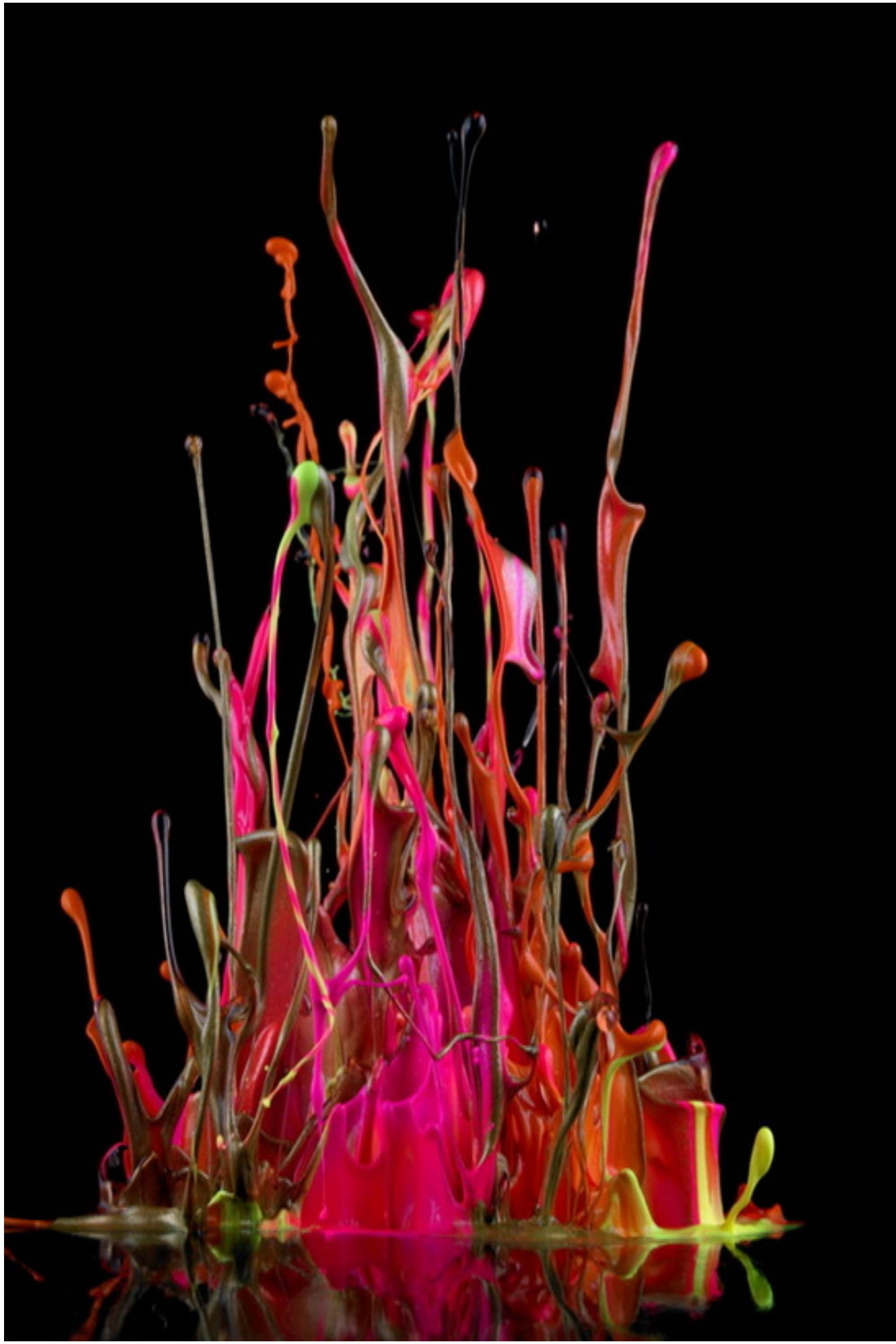


Photo Martin Klimas

Programmation pour Ingénieur

Gestion mémoire

ME 3e semestre

rev. 2025.1

Christophe Salzmann

Laboratoire d'Automatique

Pourquoi mon code *crash*-t-il?

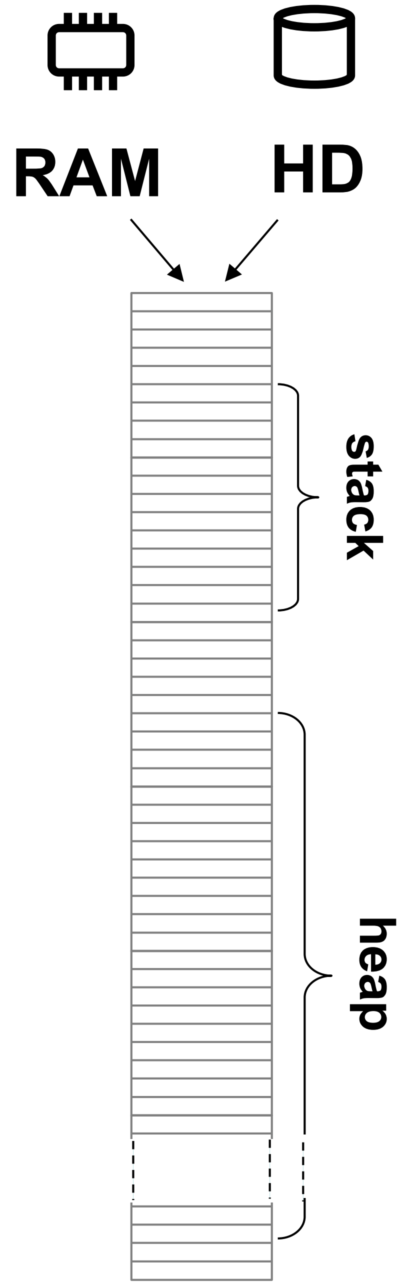
- Mon programme crash avant la première instruction
- Mon programme crash lorsque je crèe un tableau
- Mon programme crash lorsque j'accède à un tableau
- Mon programme crash lorsque j'accède au contenu d'un pointeur

Raisons possibles dans les slides qui suivent

Version très simplifiée d'une gestion mémoire moderne

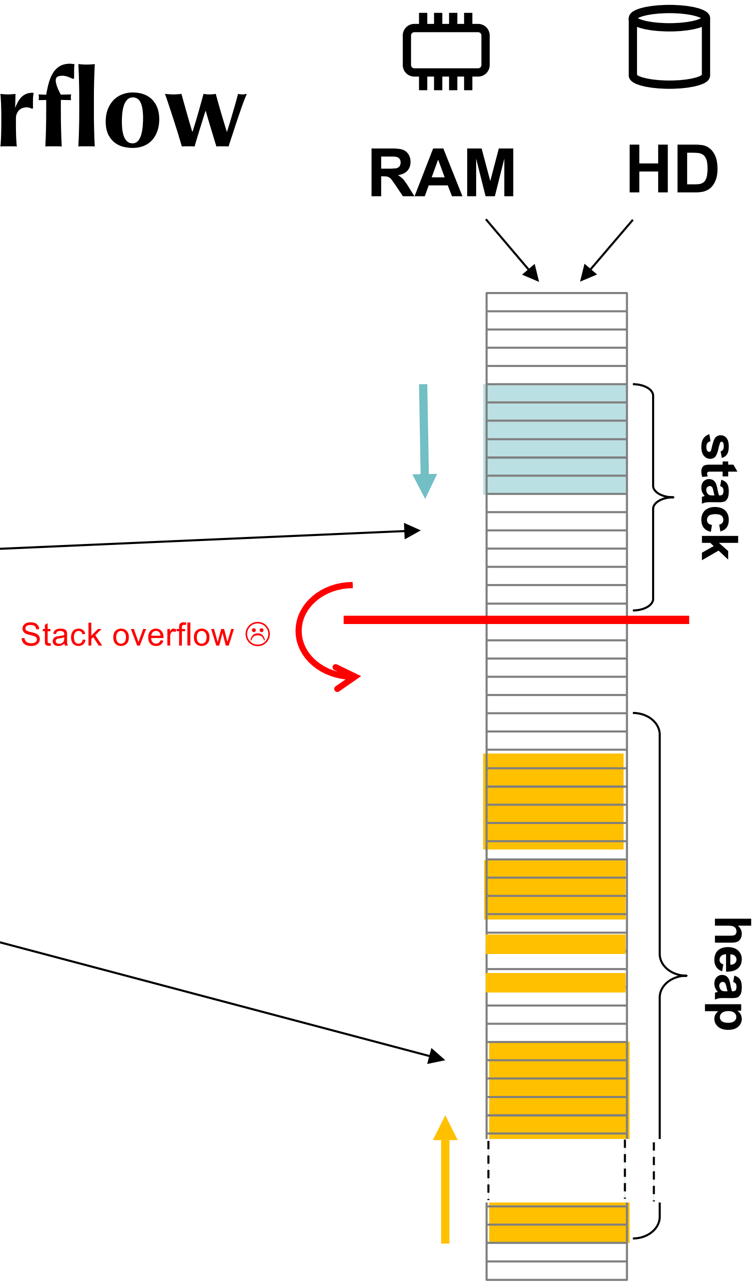
La gestion mémoire est très fortement influencée par l'OS, le langage, le compilateur, etc. Certains paramètres sont configurables

Mémoire – stack overflow



Mémoire – stack overflow

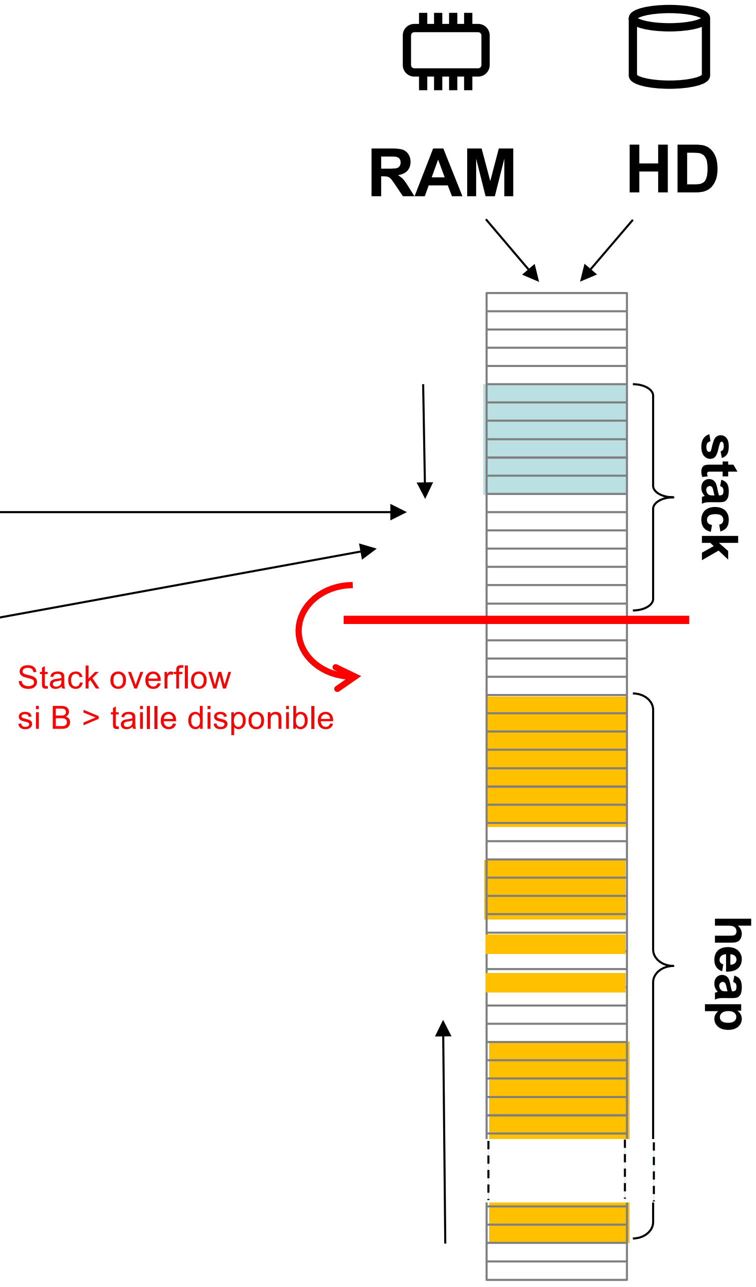
```c
int main(int argc, const char * argv[]) {

    int A[20];

    int B = 3;
    char C = 'a';

    A[9]=9;

   int* D=malloc(100000);

      if (D==NULL) {
          /* gestion error */}

   D[0]=0;
   ...

   free(D);

   return 0;
}
```



OSX stack size ~8 MB, heap size ~50 GB (very specific)

Mémoire – VLA

```
int main(int argc, const char * argv[]) {

    int n=0;

    printf("nb Elem in B? ");
    scanf("%d",&n);

    int B[n];

   // B[] is C99 Variable Length Array or VLA

    B[0]=0;

    return 0;
}
```



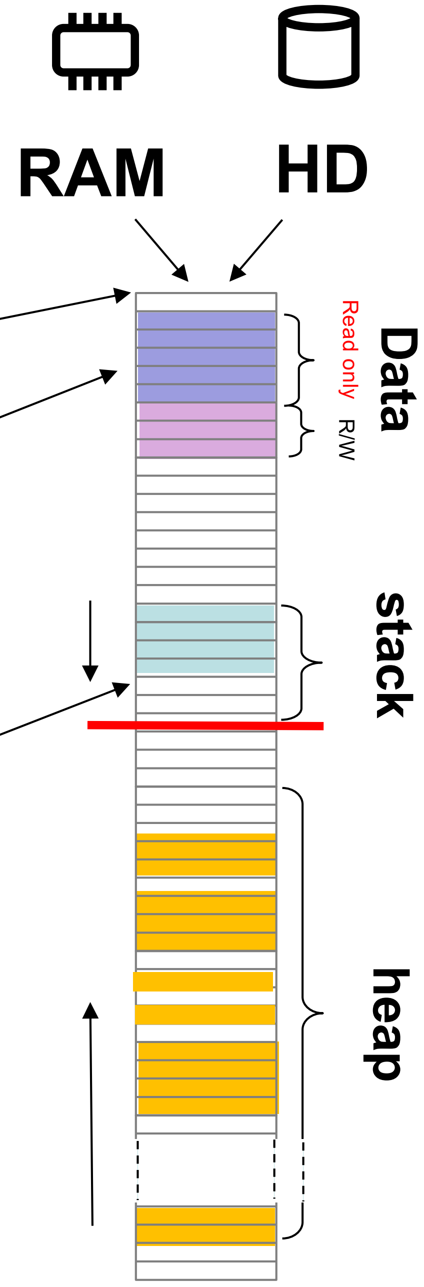
Mémoire – Segment Fault

```
int main(int argc, const char * argv[]) {

    int* p;
    *p = 10;

    char *str="HEllO\n";
    puts(str);
    *(str) = 'Y';

    char str2[]="HEllO\n";
    puts(str2);
    str2[0]= 'Y';

    return 0;
}
```

Segment fault
p is NULL

Segment fault
Read only

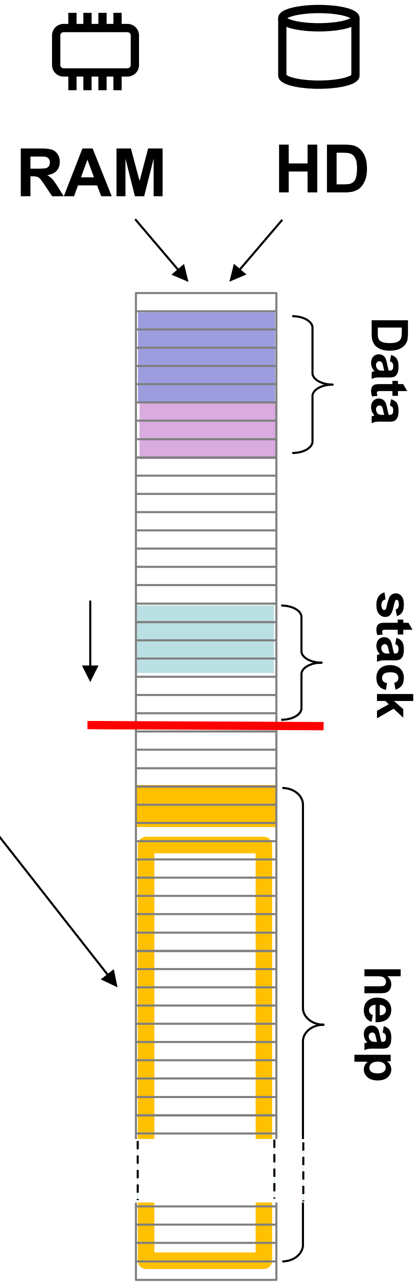
OK



Mémoire – Sz >SSD

```c
int main(int argc, const char * argv[]) {

    int* D=malloc(100000000000000); // ~100 TB!!!!

    if (D==NULL) {/* gestion error */} // OK ?¿?¿?

    for (long long i=0;i<100000000000000;i++)
        D[i]=(int)i;

   free(D);

   return 0;
}
```

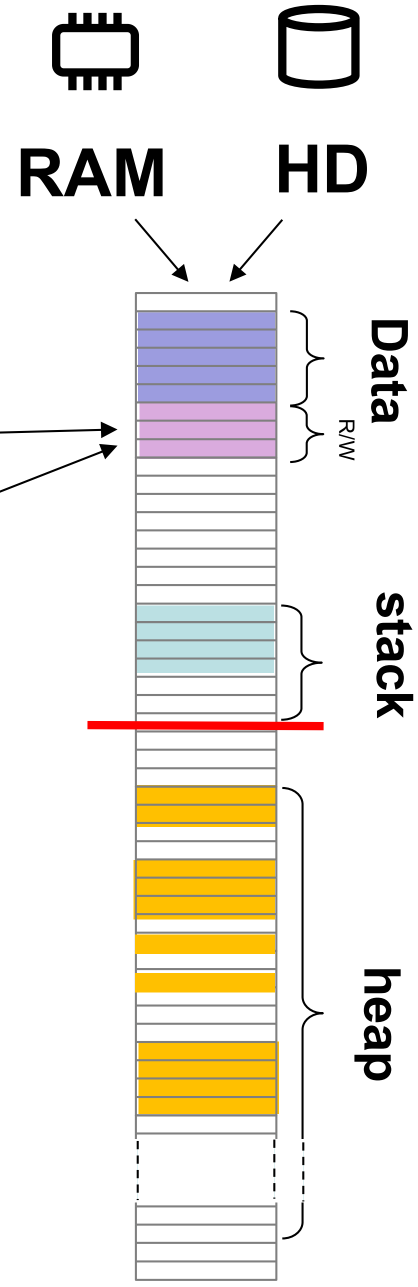
Crashes when i ~13772069888
~53GB



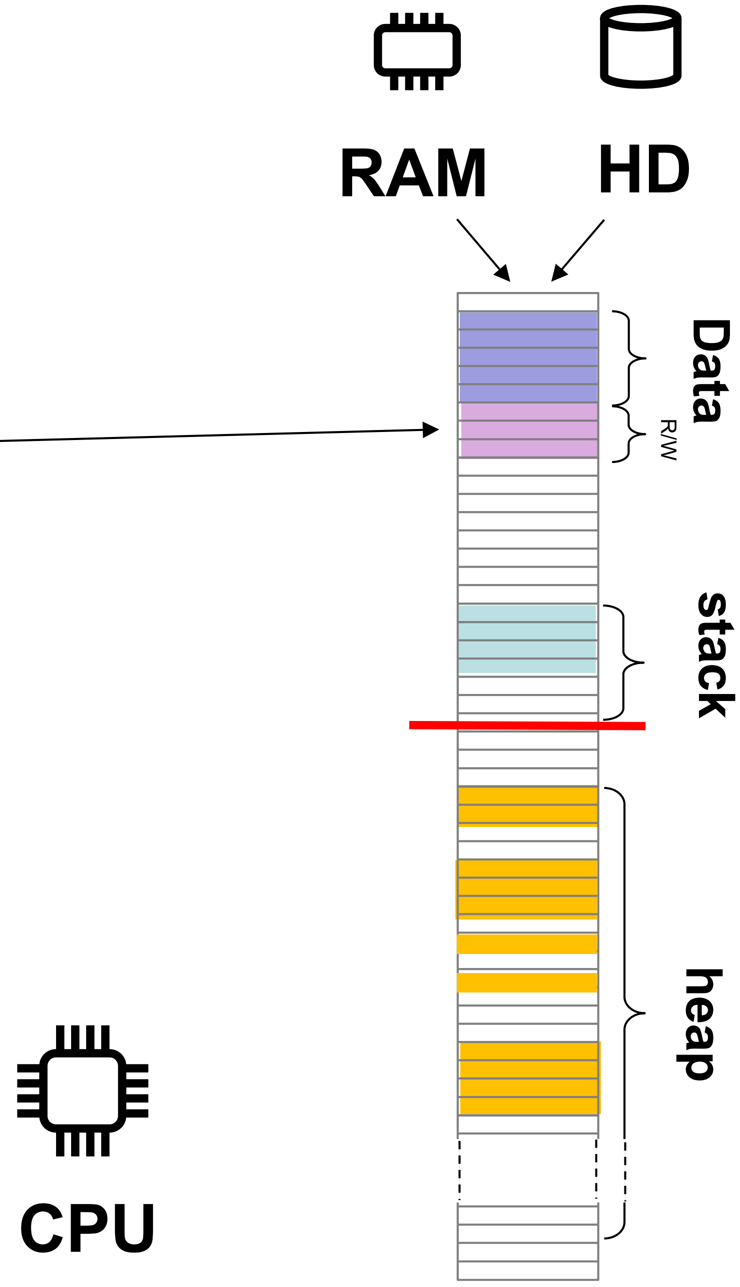
Overcommit refers to the practice of giving out virtual memory with no guarantee that physical storage for it exists

Mémoire – `static`

```c
int myGlobal = 5;

int main(int argc, const char * argv[]) {

    static int myStatic = 22;


    return 0;
}
```



Mémoire – register




```
int myGlobal = 5;

int main(int argc, const char * argv[]) {

   register int myStatic = 22;


   return 0;
}
```

En théorie dans un registre du CPU

Tableau de taille variable – `Pointeur` (C)

En C, il est possible de créer des tableaux dont la taille n'est connue que lors de l'exécution. Ces tableaux requiert une gestion de la mémoire précise sous peine de crash. La gestion de ces tableaux dynamiques font appel aux pointeurs.

```c
const unsigned int TailleInitiale = 10;

int *dynArray = (int*)malloc(TailleInitiale * sizeof(int));

for(int i=0; i<TailleInitiale; ++i){
        dynArray[i] = i;
        }
…

free(dynArray);
```

Il est primordial de s’assurer que pour chaque appel à `malloc()` il y a un appel à `free()`

Pointer, malloc, free

`malloc()` réserve la mémoire demandée (spécifiée en nombre de **bytes**) et retourne un pointeur de type **`void`** qui peut être *typecasté* sur l'espace mémoire créé ou **`NULL`** en cas d'erreur.

```
#include <stdlib.h>

const unsigned int N_elem = 10;

int *dynArray = (int*) malloc(N_elem * sizeof(int));

if (dynArray == NULL)  { /* Gestion Error */}

dynArray[0] = 0;         // array like access
*(dynArray + 1) = 1;     // ptr like access

free(dynArray);
```

Pointer, malloc, free

`free()` « libère » l'espace mémoire réservé. Il indique au gestionnaire de mémoire que cet espace est à nouveau disponible. Il **n'efface pas** le contenu de la mémoire.

```
#include <stdlib.h>

const unsigned int N_elem = 10;

int *dynArray = (int*) malloc(N_elem * sizeof(int));

if (dynArray == NULL)  { /* Error */}
dynArray[0] = 0;         // array like access

free(dynArray); // mem is free-ed NOT Erased

free(dynArray); // triggers an error since already free-ed

free(NULL);    // OK, does nothing
```

calloc

`calloc()` fonctionne comme **`malloc()`**, i.e. réserve la mémoire demandée (spécifiée avec le **nombre** d'éléments et la taille unitaire de l'élément spécifiée en **bytes**) et retourne un pointeur de type **`void`** qui peut être *typecasté* sur l'espace mémoire créé ou **NULL** en cas d'erreur.

```
#include <stdlib.h>

const unsigned int N_elem = 10;

int *dynArrayC = (int*) calloc(N_elem , sizeof(int));

// dynArray[0] .. dynArray[9] are set to '0'

if (dynArrayC == NULL)  { /* Gestion Error */}

free(dynArrayC);

dynArrayC = NULL; // safety measure, dynArrayC can't be used again
```