C - Fonction supression pointeurs, double pointeur

En C il est courant d'utiliser une fonction pour supprimer des pointeurs que nous avons initialisé et utilisé.
Cependant lors du rendu de notre premier TP en deuxième année (où le niveau avait donc augmenté d'un cran), notre cher M. Palermo nous à fait remarqué quelque chose que personne n'a fait : Passer un double pointeur dans la fonction de suppression !

Cela ressemble au code suivant :

```c
void destroy_pointer(int** ptr) {
    free(*ptr);
    *ptr = NULL;
}
```

Mais pourquoi faire cela ?
Selon Saint Palermo, ne pas faire cela représenterais une potentielle faille de sécurité. Mais au lieu de bêtement faire ce qu'on nous dis, comprenons ce qu'il se passe.

Tout d'abord, pourquoi ne pourrions nous pas simplement écrire le code suivant ? (C'est même ce que ChatGPT fait en général) :

```c
void destroy_pointer(int* ptr) {
    free(ptr);
    ptr = NULL;
}
```

Le problème se situe dans la manière qu'a le Language C de passer des paramètres,
dans cette deuxième version, nous récupérons effectivement l'adresse pointé via ce paramètre, et pouvons **free** les données située à celle-ci.
CEPENDANT, lorsque nous affectons la valeur *NULL* à ce pointeur, nous pouvons constater que cela ne prend pas effet dans la fonction appelante, ce qui n'est pas le comportement attendu (et correspond à ce qu'on appelle un **Dangling Pointers**, une potentielle faille de sécurité).

Nous pouvons le constater en exécutant le code suivant :

```c
void destroy_pointer(int* ptr) {
    free(ptr);
    ptr = NULL;
}

int* number = (int*)malloc(sizeof(int));
destroy_pointer(number);

if (number) {
    printf("Pointeur non NULL !\n"); // CE TEXTE S'AFFICHE, ET C'EST PAS
NORMAL
} else {
    printf("Pointeur NULL\n");
}
```

Ce qu'il se passe, c'est que dès lors que nous appelons une fonction en C, le paramètre passé est une **COPIE** de la variable de base et deviennent des variables locales.
Cela explique donc qu'il ne se passe rien lors de l'affectation à *NULL* car nous modifions une variable locale, qui disparaîtra à la fin de l'execution de la fonction.

Comme nous le savons déjà, si nous voulons qu'une modification faite dans une fonction en C persiste après la fin de la fonction, nous devons utiliser un pointeur et modifier quelque chose pointé par celui ci. Mais dans notre cas pour modifier un pointeur, nous devons effectivement passer : ***Le pointeur d'un pointeur***

Voici donc la version légèrement modifié du code qui implémente cette idée :

```c
void destroy_pointer(int** ptr) {
    free(*ptr);
    *ptr = NULL;
}

int* number = (int*)malloc(sizeof(int));
destroy_pointer(&number);

if (number) {
    printf("Pointeur non NULL !\n");
} else {
    printf("Pointeur NULL\n"); // CE TEXTE S'AFFICHE, NICKEL !
}
```