Addendum :
Utilisation de tableaux à longueur variable

Attention : même si elle est possible dans les extensions de C, l'utilisation de tableaux à longueur variable (VLA) est **absolument déconseillée** par les recommandations de différentes normes (ANSSI[1] pour ne citer qu'elle). Dans le cadre de vos programmes (au moins en 1A), il vous est demandé de ne **jamais** y faire appel.

Raison 1

La première raison relève des risques de débordement de pile. La pile est un espace mémoire de taille modeste par rapport au tas (où sont faites les allocations dynamiques par la fonction `malloc`). Allouer un tableau de grande taille calculée à l'exécution pourra avoir le même effet qu'allouer un tableau de taille statique de grande taille : pas assez de place dans la pile. Dans ce cas, le programme s'interrompra abruptement.

Nous avons déjà[2] dit qu'allouer un tableau local de taille connue ne devait être fait que si cette taille était modeste. Par conséquent, il en va de même pour ceux de taille inconnue à la compilation et calculée à l'exécution. Or, il est très facile de ne pas se rendre compte que la taille que l'on va demander à l'exécution est déraisonnable car … on ne la voit pas explicitement en écrivant le code. Alors que, si l'on écrit explicitement une déclaration

```
int t[100000000]
```

il est manifeste que l'on est en train de faire quelque chose de fort discutable. Une telle allocation plantera effectivement à l'exécution, alors que la même quantité de mémoire demandée par un `int *t = malloc (100000000 * sizeof (int))` n'aurait pas posé de problème sur une machine moderne.

Ainsi, avec un VLA, un programme pourra planter alors qu'il aurait pu très bien fonctionner avec un `malloc`. Essayez :

```
int main () {
  int t[100000000] ;
  return 0 ;
}
```

puis

```
#include <stdlib.h>

int main () {
  int *t = malloc (100000000 * sizeof (int)) ; /* Faudrait faire un free. */
  return 0 ;
}
```

et appréciez la différence.

Avec un VLA, vous n'avez aucun moyen de détecter le manque de mémoire : ça plantera sans prévenir. Avec un `malloc`, il suffit de vérifier que le pointeur retourné n'est pas `NULL`.

---

1 https://cyber.gouv.fr/publications/regles-de-programmation-pour-le-developpement-securise-de-logiciels-en-langage-c

2 À vérifier que ça a bien été dit lors de la présentation de `malloc`.

Raison 2

La seconde raison est plus sournoise. Un VLA, comme toute variable locale, est détruit à la sortie de la fonction qui le crée (plus généralement, du bloc qui le déclare). Il est courant de devoir allouer dynamiquement de la mémoire (des tableaux) pour les faire retourner par la fonction qui les crée ou pour mémoriser ces tableaux (leur adresse de début) dans des structures de données devant survivre à la fonction (variables globales par exemple ou membre d'une structure de donnée elle-même retournée par la fonction).

Si l'on utilise un VLA, puisqu'il est détruit à la fin de l'appel de la fonction, tout traitement tentant d'y accéder ultérieurement accèdera en fait à une zone mémoire qui n'est plus valide. Le pire est que cette zone appartiendra toujours à l'espace mémoire de votre programme donc aucun accès illégal ne sera détecté par le système d'exploitation. Par contre, les appels de fonctions ultérieurs pourront avoir réutilisé cette zone pour leur propres variables locales (paramètres et autres informations). Donc votre programme détruira de manière non déterministe (au gré des futurs appels de fonctions) les informations stockées dans cet ancien tableau… sur lequel vous croyez encore pouvoir compter. C'est donc une source de bugs difficilement analysables et localisables.

Avec un `malloc`, la zone mémoire serait allouée ailleurs que sur la pile (dans le tas) et survivrait donc à la fin de la fonction (tant que personne n'aura fait de `free`). Il en résulte que le traitement sera parfaitement sûr.

Conclusion

Ainsi, afin d'éviter de devoir se demander (au risque d'oublier de temps en temps voire tout le temps) si l'on est dans un cas favorable aux VLA ou l'inverse, autant adopter un mode d'allocation qui fonctionne **dans tous les cas** : `malloc`. Et de bannir les VLA.

Comme indiqué précédemment, cela permet de surcroît de vérifier que l'espace mémoire demandé est bien disponible. Alors, certes, il faut vérifier le pointeur retourné. Mais au moins, ça évite (si on le fait bien) que notre programme ne plante sèchement. Et un programme bien fait vérifie toujours que les ressources qu'il demande lui ont bien été attribuées.