AP2
Travaux dirigés sur les pointeurs
Partie 2 : allocation dynamique

1 Allocation dynamiques de tableaux

Exercice 1.

(a) Le programme suivant alloue un tableau statique grâce à une fonction. Ce programme compile mais plante en cours d'exécution. Expliquer pourquoi.

```c
#include <stdio.h>
#define TAILLE 10

int* allocation()
{
   int t[TAILLE];
   int i;
   for(i = 0 ; i < TAILLE ; i++){
      t[i] = i;
   }
   return t;
}

int main()
{
   int* p;
   int i;
   p = allocation();
   for (i = 0; i < TAILLE;i++){
      printf("%i \n", p[i]);
   }
   return 0;
}
```

(b) Écrire un programme qui alloue dynamiquement un tableau d'entiers dont la taille sera demandée à l'utilisateur.

(c) On souhaite maintenant écrire une fonction qui réalise l'allocation dynamique d'un tableau. Pour chaque programme ci-dessous, schématiser l'état de la mémoire à chaque étape. Si un programme ne fonctionne pas correctement expliquer pourquoi.

Programme 1:

```c
#include <stdio.h>
#include <stdlib.h>

void allocation(int* t, int n)
{
   int i = 0;
   t = malloc(sizeof(int)*n);
   for(i = 0 ; i < n ; i++){
      t[i] = i;
   }
}

int main()
{
   int* tab;
   int i, taille = 4;
   allocation(tab , taille);
   for (i = 0; i < taille; i++){
      printf("tab[%i] = %i \n", i, tab[i]);
   }
   return 0;
}
```

Programme 2:

```c
#include <stdio.h>
#include <stdlib.h>

int* allocation(int n)
{
   int * t;
   int i = 0;
   t = malloc(sizeof(int)*n);
   for(i = 0 ; i < n ; i++){
      t[i] = i;
   }
   return t;
}

int main()
{
   int* tab;
   int i, taille = 4;
   tab = allocation(taille);
   for (i = 0; i < taille; i++){
      printf("tab[%i] = %i \n", i, tab[i]);
   }
   return 0;
}
```

Programme 3:

```c
#include <stdio.h>
#include <stdlib.h>

void allocation(int** p_tab, int n)
{
   int i = 0;
   *p_tab = malloc(sizeof(int)*n);
   for(i = 0 ; i < n ; i++){
      (*p_tab)[i] = i;
   }
}

int main()
{
   int* tab;
   int i, taille = 4;
   allocation(&tab, taille);
   for (i = 0; i < taille; i++){
      printf("tab[%i] = %i \n", i, tab[i]);
   }
   return 0;
}
```

Exercice 2.

(a) Écrire une fonction `saisie_tableau` qui :
   — demande le nombre de valeurs à stocker,
   — déclare et alloue un tableau d'entier correspondant,
   — demande à l'utilisateur la valeur de chaque entier.

(b) Écrire une fonction permettant d'ajouter une case à un tableau déjà alloué. Les anciennes valeurs seront conservées.
Principe:
   — on alloue un nouveau tableau à la bonne taille,
   — on copie les valeurs de l'ancien tableau dans le nouveau,
   — on ajoute la nouvelle valeur (à la fin),
   — on libère la mémoire allouée pour l'ancien tableau,
   — on modifie le pointeur de l'ancien tableau pour le faire pointer vers le nouveau.

(c) Écrire une fonction permettant de modifier un tableau `t` de manière à ce que l'élément d'indice `ind` dans le tableau soit supprimé.

(d) Écrire une fonction `saisie_entiers` permettant à l'utilisateur de saisir un nombre quelconque d'entiers non nuls. L'utilisateur indiquera que la saisie est finie en rentrant comme valeur `0`. La fonction retournera alors un tableau contenant les entiers saisis.

(e) Écrire une fonction permettant de diviser un tableau `t` en deux sous-tableaux `t1` et `t2`. à partir de l'indice `ind` passé en paramètre

Exercice 3 (Tableaux de tableaux).
On écrit un programme permettant de gérer les notes de différents groupes de TDs. Pour

chaque TD, les notes sont stockées dans un tableau de flottants alloué dynamiquement ayant pour taille le nombre d'élèves présents au devoir.

On souhaite pouvoir gérer un ensemble de `8` classes de TD, pour cela on crée un tableau dynamique de pointeurs sur flottant `ensemble_notes`, où chaque case contiendra l'adresse d'un tableau dynamiques contenant les notes d'une classe de TD. On créera aussi un tableau d'entiers `tab_nombre_notes` de taille `8` qui stockera le nombre de notes saisies pour chaque TD.

(a) Schématiser en mémoire la structure de données ainsi définie

(b) Écrire le programme principal qui déclare et alloue un tableau dynamique de 8 pointeurs sur flottant qui contiendra les adresses de tableau de notes des 8 groupes de TD. À ce stade, les tableaux de notes ne sont pas alloués.

(c) Écrire la fonction `saisir_note_TD` qui prend en entrée le tableau de pointeurs sur flottant `ensemble_notes`, le tableau d'entiers `tab_nombre_notes`, un numéro de TD et qui :

- demande à l'utilisateur le nombre de note à saisir pour ce TD,
- alloue le tableau correspondant,
- demande à l'utilisateur de saisir toutes les notes

Cette fonction renvoie le nombre de note saisies pour le TD concerné.

(d) Écrire une fonction d'affichage des notes de toutes les classes de TDs.

(e) Modifier le programme principal afin qu'il effectue la saisie et l'affichage des notes pour chaque TD.
Attention à ce que la mémoire allouée soit bien libérée après utilisation.

(f) Écrire la fonction `ajouter_note` qui prend en entrée le tableau de pointeurs sur flottant `ensemble_notes`, le tableau d'entiers `tab_nombre_de_notes`, un numéro de TD, une note et qui rajoute cette note à la liste des notes du TD concerné.

(g) Écrire une fonction `moyenne_globale` qui prend en entrée le tableau `ensemble_notes`, le tableau `tab_nombre_notes` et qui calcule la moyenne générale des notes pour tous les TDs.

(h) Écrire un fonction `meilleure_TD` qui prend en entrée le tableau `ensemble_notes`, le tableau `tab_nombre_notes` qui renvoie l'indice du TD ayant la meilleure moyenne.

Exercice 4 (Crible d'Eratosthène). Le *crible d'Ératosthène* est un algorithme calculant tous les nombres premiers inférieurs à un entier donné, qui procède comme suit : on place les nombres entiers dans un tableau, et pour chacun d'eux, on parcourt tout le tableau en remplaçant les multiples du nombre de l'itération courante par 0. À la fin de la procédure, les nombres non nuls restant sont les nombres premiers recherchés.

(a) Écrire une fonction `crible_eratosthene` qui prend en entrée un entier naturel `n` et qui renvoie un tableau de taille `n` où `tab[i]` vaut `i+1` si `i+1` est un nombre premier et `0` sinon.

(b) Proposer des améliorations pour réduire le nombre d'opérations nécessaires.
Indication : si d divise n, alors $d' = n/d$ divise aussi n ; observer que $d \leqslant \sqrt{n} \iff d' \geqslant \sqrt{n}$.

(c) Écrire un fonction `nombre_premier` qui prend en entrée un entier naturel `n` et qui renvoie un tableau contenant uniquement la liste des nombres premiers inférieurs à `n`.