TP2 : Tableaux et pointeurs

1 Des matrices de génériques

Dans cet exercice, vous effectuerez des allocations dynamiques de tableaux 2d d'éléments génériques.

1. Proposez une fonction *allocationGenerique2d* prenant en paramètre, entre autre, une largeur et une hauteur, et qui réalise l'allocation mémoire d'une matrice 2d de *generique*. Cette fonction devra être de type void : quel paramètre supplémentaire votre fonction devra-t-elle prendre ?
2. On souhaite maintenant créer une structure de données contenant des matrices de *generique*. Pour ce faire, commencez par définir le nouveau type de données suivant, permettant de stocker des matrices de *generique* :

```
typedef struct
{
    generique **data;
    uint64_t hauteur;
    uint64_t largeur;
} matriceg;
```

   Proposez une fonction *creerMatriceGenerique* prenant en paramètre une hauteur et une largeur, et qui construira une *matriceg* qui sera retournée en sortie. Cette fonction appellera *allocationGenerique2d* afin de réaliser l'allocation mémoire du champ *data*.
3. Créez une fonction *remplirMatriceGenerique* qui prend en paramètre une *matriceg* ainsi que les différentes valeurs, sous forme de chaines de caractères, à placer dans la matrice. La fonction devra donc placer ces valeurs dans la *matriceg*.
   Voici un exemple de code pour créer et remplir une matrice 2x2 :

```
matriceg r = creerMatriceGenerique(2, 2);
remplirMatriceGenerique(r, "32", "34.56", "-3", "-9.76");
```

   Pour vous aider, voici un petit code avec une fonction contenant un nombre variable de paramètres :

```
#include <stdarg.h>

void afficher(int num_arg, ...)
{
    uint32_t i;
    va_list ap;

    va_start(ap, num_arg);
    for(i = 0; i < num_arg; i++)
    {
        printf("%s\n", va_arg(ap, char *));
    }
    va_end(ap);
}

int main(int argc, char **argv)
{
    afficher(3, "bonjour", "ca va", "oui, moi aussi");
    return 0;
}
```

La fonction affichera à l'écran :

```
bonjour
ca va
oui, moi aussi
```

4. Proposez une fonction *afficheMatriceGenerique* qui prend en paramètre une *matriceg* et affiche tous ses éléments : sa hauteur, sa largeur, et chaque élément en faisant appel à la fonction *afficheGenerique* (on aura donc un élément de la matrice par ligne).
5. Réalisez une fonction réalisant la destruction mémoire des données d'une matriceg. Testez votre code et utilisez, si possible, valgrind pour vérifier que toutes les allocations mémoires effectuées dans votre programme ont été libérées.
6. Ecrivez une fonction *addMatriceGenerique* permettant d'additionner deux matrices de *generique*. Votre fonction devra effectuer les contrôles nécéssaires sur les tailles des matrices au préalable.

2 A la recherche de la taille

La plupart des fonctions prenant en paramètre un tableau dynamique doivent prendre aussi sa taille en paramètre. Ceci est dû au fait qu'il est soit-disant impossible de retrouver la taille d'un tableau alloué dynamiquement...

Cependant, certaines fonctions de C connaissent la taille d'une zone mémoire allouée dynamiquement, sans qu'il soit nécessaire de leur transmettre explicitement l'information. Par exemple, la fonction *free* connait la taille de la zone mémoire à libérer, sans qu'il soit nécessaire de la lui transmettre. Nous verrons dans cet exercice comment ces fonctions s'y prennent, et nous tenterons de mettre en place un mécanisme similaire.

1. Cet question pourrait ne pas fonctionner si vous utilisez un autre compilateur que gcc. Même en utilisant ce dernier, il n'y a aucune garantie que cela fonctionne...

   Considérez le code suivant :

```c
int main(int argc, char **argv)
{
    uint32_t *g;
    g=(uint32_t*)malloc(6833);

    printf("%p\n", g);
    t = (uint32_t*)g;
    printf("%u\n", *t);
    return 0;
}
```

   On affiche l'adresse du tableau g, puis le contenu du pointeur t. Si vous exécutez ce programme, il n'affichera rien de vraiment intéressant.

   Modifiez le programme afin qu'il n'affiche pas le contenu de t, mais le contenu du uint32_t situé juste avant t : que constatez-vous. Si vous modifiez la taille de la zone mémoire, cette valeur change-t-elle ?

   Certaines implémentations de la fonction malloc stockent la taille de la zone mémoire (et d'autres informations peut-être) avant le tableau... La fonction free lit cette information pour décider de la taille de la zone mémoire à libérer. Cependant, ce n'est pas exactement la taille du tableau qui est stockée, mais une autre information...

   Nous allons faire nos propres fonctions d'allocation mémoire et de libération mémoire utilisant ce même principe.

2. Créez une fonction *mymalloc* qui fait la même chose que malloc : elle prend en paramètre une taille en octet, et réalise une allocation mémoire de cette taille et renvoie le pointeur associé.

   Cependant, votre fonction devra effectuer une allocation de 8 octets de plus, afin de stocker, en tout début de la zone mémoire, la taille de la zone mémoire allouée (vous la coderez sur un entier 64 bits non signé). Vous renverrez non pas le pointeur renvoyé par votre appel de malloc, mais l'adresse de la zone mémoire située après ces 8 premiers octets où est stockée la taille.

   Attention, quel est le type du pointeur renvoyé ?

Conseil : pour vérifier que votre fonction a marché, affichez dans la fonction *mymalloc* l'adresse renvoyée par malloc, et dans le main, l'adresse renvoyée par votre fonction : cette dernière doit valoir 8 octets de plus que la première adresse.

3. Créez une fonction *mymemsize* qui prend en paramètre le pointeur renvoyé par la fonction *mymalloc*, et renvoie la taille de la zone mémoire allouée. Attention, quel doit être le type du pointeur passé en paramètre ?
4. Créez une fonction *mytabsize* qui prend en paramètre le pointeur renvoyé par la fonction *mymalloc*, et renvoie le nombre de cases du tableau de la zone mémoire alloué. Attention, cette fonction a besoin d'un paramètre supplémentaire : lequel ?
5. Créer une fonction *myfree* qui libère la zone mémoire.
6. Faisons un test... Dans le main, créez un tableau de 10 uint32_t (avec la fonction *mymalloc*), et remplissez chacune de ses cases avec la valeur 2*indice, où indice est l'indice de la case.

   Ecrivez une fonction test qui prend comme **unique paramètre** un tableau de uint32_t et affiche tous ses éléments, et testez la avec le tableau créé dans le main.
7. Testez que toute la mémoire est bien libérée par vos fonctions avec valgrind.