TP3 : Plus de tableaux, liste chaînée

1 Un tableau qui s'agrandit à volonté

Dans cet exercice, vous réutiliserez les programmes réalisés lors du dernier TP permettant de construire un tableau dont la taille était stockée en début de zone mémoire (fonctions **mymalloc**, ...). Le but est de proposer des fonctionnalités supplémentaires permettant d'agrandir à volonté le tableau si on écrit une donnée hors de ce dernier.

1. Proposez une fonction *myread_uint32* permettant de lire la valeur dans une case d'un tableau d'entiers non signés 32 bits alloué avec **mymalloc**. Si la case demandée est hors du tableau, la fonction renvoie 0, sinon, elle renvoie le contenu de la case demandée.
2. Proposer une fonction *mywrite_uint32* permettant d'écrire une donnée dans une case d'un tableau d'entiers non signés 32 bits alloué avec **mymalloc**. Si la case demandée est hors du tableau, ce dernier est agrandi afin de contenir la case souhaitée, où l'on écrira ensuite la valeur donnée.

   Réfléchissez attentivement aux paramètres de cette fonction. Vous pouvez regarder la documentation des fonctions **realloc** et **memcpy** : il est possible de répondre à cette question en utilisant seulement l'une de ces deux fonctions.

2 Tableau 2d à l'aide d'un tableau 1d

Dans cet exercice, nous verrons qu'il est possible d'imiter le fonctionnement d'un tableau 2d à l'aide d'un tableau mono dimensionnel.

1. Copiez et coller ce code dans un nouveau programme :

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

#define hauteur 10000
#define largeur 10000


int main(int argc, const char * argv[]) {
    uint32_t i,j;
    uint32_t *tab;
    uint64_t s;

    srand(time(NULL));

    clock_t begin_prog = clock();

    tab = (uint32_t*) calloc(hauteur*largeur, sizeof(uint32_t));


    for (i=0; i<hauteur; i++)
        for(j=0; j<largeur; j++)
            tab[???] = rand();

    clock_t begin_somme = clock();

    s=0;
```

```
28      for (i=0; i<hauteur; i++)
29          for(j=0; j<largeur; j++)
30              s+=tab[???];
31
32      clock_t end_somme = clock();
33
34      double time_spent = (double)(end_somme-begin_somme)/CLOCKS_PER_SEC;
35      printf("Temps pour la somme : %lf\n", time_spent);
36
37
38      free(tab);
39
40      clock_t end_prog = clock();
41
42      time_spent = (double)(end_prog-begin_prog)/CLOCKS_PER_SEC;
43      printf("Temps pour le programme : %lf\n", time_spent);
44
45      return 0;
46
47  }
```

Ce code permet d'allouer un tableau 1d de **hauteur*largeur** cases, d'y stocker des nombres aléatoires, et de calculer leur somme en considérant le tableau comme un tableau 2d. A la fin, le programme affiche le temps passé à calculer la somme des éléments du tableau, ainsi que le temps passé dans le programme au total.

Afin de le faire marcher, il vous faut compléter les lignes 23 et 30. Une fois cette tâche effectuée, mesurez le temps d'exécution de votre programme.

2. Ecrivez un programme similaire utilisant un véritable tableau 2d dynamique afin de stocker les valeurs, et mesurez le temps d'exécution de votre programme : y a-t-il une grande différence avec la version précédente ?
3. Dans votre code de tableau dynamique 2d, que se passe-t-il si vous inversez les boucles imbriquées contrôlant le parcours du tableau (par exemple, au lieu de parcourir le tableau par la hauteur, puis par la largeur, vous le parcourez par la largeur, puis par la hauteur) ?
4. Que pouvez-vous en conclure concernant les tableaux 1d utilisés comme des tableaux 2d ?

3 Définir une bibliothèque de fonctions sur les listes

Dans ce premier exercice, vous allez réaliser l'ensemble des fonctions nécessaires à la gestion de listes chaînées.

1. Dans un premier temps, déclarez une structure *maillon* qui représentera les maillons de la liste, et une structure *liste* qui représentera la liste. Chaque maillon contiendra un entier non signé sur 32 bits :

```
1  typedef struct maillon
2  {
3      uint32_t value;
4      struct maillon *next;
5  } maillon;
6
7  typedef struct liste
8  {
9      uint32_t taille;
10     maillon *tete;
11     maillon *queue;
12 } liste;
```

2. Vous devrez maintenant écrire les fonctions suivantes, qui sont les fonctions standardes de gestion des listes :
   – **liste *Liste_new()** qui initialise une nouvelle structure de liste et renvoie un pointeur dessus.

- **maillon* Maillon_new(uint32_t v)** qui initialise un nouveau maillon, y copie la valeur passée en paramètre et renvoie le pointeur sur ce maillon.
- **void Maillon_free(maillon *m)** qui réalise la libération mémoire d'un maillon.
- **void Liste_free(liste *l)** qui réalise la libération mémoire d'une liste, et de chacun de ses maillons.
- **void Liste_pushTete(liste *l, uint32_t v)** qui rajoute, en tête de liste, un maillon contenant la valeur v.
- **void Liste_pushQueue(liste *l, uint32_t v)** qui rajoute, en queue de liste, un maillon contenant la valeur v.
- **uint32_t Liste_popTete(liste *l)** qui retire le maillon en tête de la liste et renvoie sa valeur.
- **void Liste_affiche(liste *l)** qui affiche le contenu de la liste.
- **_Bool Liste_isEmpty(liste *l)** qui renvoie un booléen à vrai si la liste est vide, et faux sinon.

3. Testez ces fonctions dans un main en créant une liste, y ajoutant des valeurs, puis affichez la liste. Enfin, sortez certaines valeurs de la liste, affichez-les, et ré-affichez la liste. Pour finir, libérez la liste de la mémoire, et utilisez valgrind afin de vérifier que vous n'avez pas oublié de libérer de la mémoire.

4. Une fois ce travail effectué, déplacez toutes les fonctions concernant la liste dans un fichier *myliste.c*, et toutes les déclarations de structures et les include nécessaires dans un fichier *myliste.h*. Rajoutez, dans le fichier header, tous les entêtes des fonctions du fichier source correspondant.

   Enfin, rajoutez dans le fichier *myliste.c*, ainsi que dans le fichier principal de votre programme, la ligne

   ```
   1 #include "myliste.h"
   ```

   Pour compiler votre programme c, vous rajouterez le nom du fichier source, contenant les codes gérant la liste, à la liste des fichiers à compiler :

   ```
   gcc -g -Wall main.c myliste.c -o prog
   ```

   Testez afin de vérifiez que votre programme a toujours le même comportement. Grâce à ce système, vous n'êtes plus obligé de placer tout votre code dans le même fichier, vous pouvez créer différents modules de codes.