Cours de programmation séquentielle

Allocation dynamique de mémoire: fractions et tableaux

1 Buts

— Utilisation de pointeurs.
— Allocation dynamique de mémoire.
— Utilisation de tableaux dynamiques.
— Passage d'arguments par référence.
— Réusinage de code.
— Utilisation de `make`.

2 Énoncé

Ce travail pratique consiste à réécrire deux travaux pratiques précédents en utilisant l'allocation dynamique de mémoire, afin de vous habituer, sur des algorithmes simples que vous maîtrisez à gérer la mémoire manuellement. C'est implicite dorénavant, mais vous devez compiler **tous** vos travaux pratiques à l'aide de `make`. Aucune exception ne sera acceptée. Vous devez également réfléchir à comment structurer vos fichiers pour avoir une compilation séparée.

3 Les fractions

Il s'agit de reprendre la librairie que vous avez écrite sur les fractions et à modifier toutes les fonctions afin de n'utiliser que des pointeurs et allouer/désallouer la mémoire manuellement.

Ainsi, par exemple, les signatures des fonctions de création de fraction ou d'addition de deux fractions seront données par:

— `fraction *fraction_create(int32_t n, int32_t d);`
— `fraction *fraction_add(fraction *lhs, fraction *rhs);`

Cette librairie offrira notamment comme fonctionnalités la saisie, l'affichage, l'addition, la soustraction, la multiplication et la division de fractions. Les fractions devront toujours être stockées sous forme irréductible. Il faudra également gérer la division par zéro qui produira une erreur.

Ensuite, il faudra écrire un programme utilisant la librairie et permettant de vérifier que votre code se comporte comme attendu.

3.1 Exercice plus supplémentaire du tout

Lorsque vous aurez fini cette première partie, ajoutez à votre programme la possibilité de passer un calcul en argument de votre programme à la ligne de commande et retourner le résultat dans le terminal.

3.2 Cahier des charges

La librairie sera constituée **au moins** de:

- Une `struct fraction` composée de 2 champs: le numérateur et le dénominateur;
- 4 fonctions utilitaires obligatoires et une fonction bonus:
  - Une fonction qui affiche une fraction à l'écran, en prenant en argument une fraction **par référence**.
  - Une fonction qui retourne un pointeur vers fraction irréductible et qui prend en argument une fraction **par référence**.
  - Une fonction qui prend une fraction en argument **par référence** et la met à une puissance entière (positive ou négative) et retourne **un pointeur** vers une fraction irréductible.
  - Une fonction qui lit une fraction à la ligne de commande en gérant les erreurs de saisie (voir plus bas) et retourne **un pointeur** vers la fraction;
- 5 fonctions de manipulation de fractions:
  - La fonction `fraction_add()` qui additionne deux fractions (les arguments sont passés **par référence**) et retourne **un pointeur** vers une fraction irréductible.
  - La fonction `fraction_sub()` qui soustrait deux fractions (les arguments sont passés **par référence**) et retourne **un pointeur** vers une fraction irréductible.
  - La fonction `fraction_mul()` qui multiplie deux fractions (les arguments sont passés **par référence**) et retourne **un pointeur** vers une fraction irréductible.
  - La fonction `fraction_neg()` qui prend une fraction en argument (l'argument est passé **par référence**) et retourne **un pointeur** vers le négatif d'une fraction irréductible.
  - La fonction `fraction_to_double()` qui retourne la valeur en à virgule flottante de la fraction (qui est passée par référence).

Une fois votre librairie de fraction écrite, vous **devez** l'utiliser pour évaluer le nombre `pi`. Les trois formules suivantes peuvent vous êtres utiles:

$$\sum_{n=1}^{\infty} \frac{1}{n^4} = 1 + \frac{1}{16} + \frac{1}{81} + ... = \frac{\pi^4}{90}, \tag{1}$$

$$\sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n^2} = 1 - \frac{1}{4} + \frac{1}{9} + ... = \frac{\pi^2}{12}, \tag{2}$$

$$\prod_{n=1}^{\infty} \left(\frac{2n}{2n-1}\right)\left(\frac{2n}{2n+1}\right) = \frac{2}{1} \cdot \frac{2}{3} \cdot \frac{4}{3} \cdot \frac{4}{5} \cdot ... = \frac{\pi}{2}. \tag{3}$$

Il est nécessaire d'utiliser les fractions directement et non leur représentation en nombre à virgule flottante.

Faites attention à la gestion de la mémoire!

3.3 Syntaxe pour la lecture de la ligne de commande

La syntaxe pour la lecture des fractions à la ligne de commande, ainsi que de l'opération à effectuer doit avoir la syntaxe suivante: Les fractions se représentent comme `num denum` (notez l'espace entre `num` et `denum`), puis un opérateur (`+`, `-`, `x`, `/`, `^`), puis une autre fraction également représentée comme `num denum`. Un nombre entier se représentera également sous la forme d'une fraction. La seule exception est le calcul du PGCD où on passe en argument deux nombres et `PGCD` et le PGCD s'affiche.

L'entrée des fractions sur la ligne de commande pourrait ressembler à ce qui suit.

```
> ./executable 3 10 + 8 15
5 6
> ./executable 3 10 x 8 15
4 25
> ./executable 3 1 x 8 11
24 11
> ./executable 3 10 x 8 1
12 5
> ./executable 3 10 ^ 2
9 100
> ./executable 15 10 PGCD
5
```

Afin de transformer les arguments de la ligne de commande en entier, la fonction `atoi()` pourrait vous être utile. Ainsi, la fonction `strcmp()` sert à comparer deux chaînes de caractères.

4 Les tableaux uni-dimensionnels

Une façon de déclarer un tableau dynamique d'entiers `tab` est la suivante:

```
int32_t size = 10;
int32_t *tab = malloc(size * sizeof(int32_t));
```

Écrire des fonctions réalisant les actions suivantes, ainsi qu'un programme testant leur bon fonctionnement.

1. Demander à l'utilisateur d'entrer la valeur de `size` pour la création du tableau, et retourner le tableau.
2. Remplir le tableau `tab` de valeurs aléatoires plus petites qu'un entier `val_max` qui est beaucoup plus petit que `size`.
3. Trouver le plus petit élément de `tab`.
4. Placer le plus grand élément de `tab` en fin de tableau en procédant à un échange de place.

5. Rechercher dans `tab` un élément entré au clavier par l'utilisateur.
6. Calculer la moyenne des éléments de `tab`.
7. Calculer la variance des éléments de `tab`. La variance `var` des éléments de `tab` est définie par la formule:

$$\text{var} = \frac{1}{\text{size}} \sum_{i=0}^{\text{size}-1} (\text{tab}[i] - \text{tab}_m)^2, \tag{4}$$

   où tab_m est désigne la moyenne des éléments de `tab` et $\text{tab}[i]$ est le i-ème élément de `tab`.
8. Trier les éléments de `tab` par ordre croissant.
9. Trouver l'élément médian du tableau `tab`. Après avoir trié le tableau `tab`, l'élément médian est défini comme étant la valeur:
   — `tab[(size-1)/2]` si `size` est impair
   — `(tab[(size-1)/2] + tab[size/2])/2.0` si `size` est pair.

Exemple:

TABLE 1: L'élément médian de ce tableau vaut: 3.

1	5	8	2	6	5	3	1	0	3

10. Ecrire une fonction pour chacun des points précédents.
11. Vérifier que le générateur de nombres aléatoires `rand()` est équitable. Pour cela, utiliser un tableau d'entiers `histo` pour comptabiliser le nombre de fois qu'une valeur est tirée. À noter que la
valeur tirée correspond à l'indice du tableau et que c'est son contenu qui est incrémenté de 1. Utiliser les notions précédentes (moyenne, écart-type, minimum/maximum) pour dire si le générateur est équitable.
12. Visualiser le du tableau `histo` en utilisant la bibltiothèque SDL 2.0 (https://wiki.libsdl.org/FrontPage). Un exemple d'utilisation de la librairie vous est fourni sur `Cyberlearn` (`gfx_example.tar.gz`). Vous pouvez également télécharger ce fichier en cliquant sur ce lien.

4.1 Exercice pas supplémentaire

Étendre les questions "11" et "12" à un tableau en deux dimensions d'entiers, `histo_2d`. Pour ce faire il s'agira de définir un tableau de tableaux,

```
int32_t height = 200;
int32_t width = 200;
int32_t **histo_2d;
// comment allouer un tableau à 2 dimensions avec malloc?
```

puis générer des couples de nombres aléatoires, `x` et `y`, entre `0, height-1` et `0, width-1` à l'aide de la fonction `rand()`, et pour chaque tirage de `x`, `y`, incrémenter de un la valeur tableau, `histo_2d[x][y]`. Finalement, afficher le tableau en niveaux de gris à l'aide de la librairie SDL2 pour vérifier si le générateur `rand()` ne possède pas de corrélation en deux dimensions (s'il produit des paires de nombres homogènes dans l'espace).