I2010 : langage C (TP4)

Allocation dynamique de tableaux à plusieurs dimensions

1. Allocation dynamique d'une matrice : Puissance NxM

Vous connaissez tous le jeu Puissance 4[1]. Nous vous proposons de coder un jeu de puissance HxL. Ce jeu est en tout point identique au puissance 4 sauf qu'au lieu d'être joué sur un tableau de 6 cases de hauteur et de 7 cases de largeur, il se joue sur un tableau de H cases de hauteur et L cases de largeur.

Le programme vous demandera d'abord les valeurs H et L. Ensuite il vous demandera la colonne que joue le joueur X puis la colonne que joue le joueur O, puis celle du joueur X, et ainsi de suite.

A chaque coup, le programme affichera le tableau avec le caractère '.' pour les cases vides.

Dans un premier temps, nous ne vérifierons pas si l'un des joueurs a gagné. La partie se termine simplement lorsque l'un des joueurs entre la valeur 0.

Pour être « user friendly », nous commencerons à numéroter les colonnes à partir de 1 dans l'interface utilisateur (input/output) !

Vous allouerez dynamiquement la matrice en utilisant un tableau de tableaux (càd. des pointeurs de pointeurs). Allouez exactement la place nécessaire à la grille.

Exemple d'exécution :

```
Hauteur = ? 4
Largeur = ? 5

Colonne joueur X ? 3
. . . . .
. . . . .
. . . . .
. . x . .
Colonne joueur O ? 2
. . . . .
. . . . .
. . . . .
. o x . .
Colonne joueur X ? 3
. . . . .
. . . . .
. . x . .
. o x . .
Colonne joueur O ? 3
. . . . .
. . o . .
. . x . .
. o x . .
```

[1] https://fr.wikipedia.org/wiki/Puissance_4

2. Libération / réallocation de la mémoire

Reprenons le même jeu. Cette fois-ci les joueurs possèdent le pouvoir d'agrandir le tableau de jeu en rajoutant des colonnes. Il leur suffit pour cela de jouer un numéro de colonne plus grand que le nombre de colonnes courant.

Exemple :

```
. . . . .
. . o . .
. . x . .
. o x . .
Colonne joueur O ? 7
. . . . . . .
. . o . . . .
. . x . . . .
. o x . . . o
```

Les colonnes seront rajoutées à l'aide de la fonction *realloc.*

Si le joueur entre un nombre négatif, le nombre de colonnes est réduit. Par exemple, -3 ne conservera que les 3 premières colonnes.

```
. . . . . . .
. . o . . . .
. . x . . . .
. o x . . . o
Colonne joueur O ? -3
. . .
. . o
. . x
. o x
```

Il faudra bien entendu libérer la mémoire de ces colonnes et réajuster la taille du tableau.

Bonus

Détectez le joueur gagnant ! (en supposant que H et L sont supérieurs à 4).

Exercice de debugging

3. Utilisation de l'outil *valgrind* [2]

Reprenez vos solutions des exercices 2 et 3 du TP3 ainsi que la solution de l'exercice précédent et vérifiez-les avec l'outil *valgrind*. Si une fuite de mémoire (*memleak*) ou toute autre erreur est détectée, corrigez-la.

[2] Voir document « Valgrind_tuto.pdf » sur Moovin.

4. Gestion d'images (matrices dynamiques)

Il vous est demandé d'écrire un programme qui permet de manipuler des images. Les opérations à effectuer sur les images sont les suivantes :

1) Création d'une image de dimensions précisées par l'utilisateur avec valeurs aléatoires pour les pixels,
2) Création d'une image de dimensions précisées par l'utilisateur avec valeurs prédéfinies pour les pixels,
3) Affichage de l'image ;
4) Changement de la taille de l'image,
5) Affichage de l'histogramme de l'image,
6) Suppression de l'image,
7) Quitter le programme.

Pour effectuer ces différentes opérations, un menu est présenté à l'utilisateur qui choisit une opération. Une seule image sera présente en mémoire à la fois.

Opération 1 :

Lorsque l'utilisateur choisit de créer une nouvelle image, une matrice est réservée dynamiquement de la taille spécifiée par l'utilisateur. L'utilisateur introduira la taille sous le format (N,M) dans la ligne de commande (utilisez `scanf("(%d,%d)", &n, &m)`).

Lors d'une création d'une image, on demandera à l'utilisateur la profondeur de bits B souhaitée. Celle-ci définit le domaine de valeurs des pixels : $[0, 2^B-1]$. Pour les images en niveaux de gris, la profondeur de bits vaut généralement 8. Dans ce cas, chaque pixel de l'image est codé sur B=8 bits et leurs valeurs possibles sont comprises entre 0 et 255.

L'image est composée de valeurs aléatoires dans l'intervalle défini par la profondeur de bits B. Pour générer ces valeurs aléatoires, utilisez la fonction `rand()` de la librairie *stdlib*.

Lors de la création de l'image, utilisez une méthode indicée pour assigner les valeurs (et non l'arithmétique des pointeurs).

Opération 2 :

Il s'agit de la même opération que la première à la différence près que les pixels de chaque ligne de l'image auront des valeurs prédéfinies et non aléatoires. La valeur d'un pixel correspondra à son indice de ligne + 1. Assurez-vous que cette valeur respecte la profondeur de bits fixée, càd comprise entre 0 et 2^B.

Opération 3 :

Cette opération permet d'afficher l'image courante. Conventionnellement, le pixel (0,0) correspond au coin supérieur gauche de l'image.

Utilisez l'arithmétique des pointeurs pour parcourir et afficher l'image.

Opération 4 :

L’utilisateur peut demander de changer la taille de l’image (si une image existe déjà bien sûr).

Cela s'effectue avec le même format que précédemment (i.e. (N,M)). Si la nouvelle taille spécifiée est :

- Plus petite que la taille de l'image : celle-ci sera coupée (*cropped*) pour correspondre à la nouvelle taille. Celle-ci sera calculée à partir du coin supérieur gauche de l'image.
- Plus grande que la taille de l'image : une nouvelle image est créée de la bonne taille et les nouveaux pixels auront la couleur noire (càd la valeur 0). Utilisez la méthode indicée pour assigner les nouvelles valeurs des pixels.

Si l’utilisateur demande une nouvelle taille qui ne correspond ni à un découpage, ni à un agrandissement (par exemple en augmentant la hauteur mais en diminuant la largeur), un message d’erreur sera affiché.

Opération 5 :

Lorsque l'utilisateur demande l'affichage de l'histogramme de l'image, celui-ci est calculé et affiché à l’écran.

L’histogramme d’une image en niveaux de gris est la fonction qui, à chaque niveau de gris, associe le nombre de pixels de l’image possédant ce niveau de gris. Lorsque les pixels d’une image sont codés sur 1 byte, l’histogramme correspond à un tableau de 256 compteurs de pixels, un par nuance de gris.

On affichera l'histogramme (càd le tableau faisant correspondre un compteur de pixels pour chaque intensité de l’intervalle [0, 2^B-1]) et le nombre total de pixels présents dans l'image. Utilisez l'arithmétique des pointeurs pour parcourir et afficher l'histogramme.

Opération 6 :

L'utilisateur peut choisir de supprimer l'image sans quitter le programme. Dans ce cas, les mémoires allouées à l'image et à son histogramme (s'ils sont présents) seront libérées.

Opération 7 :

Lorsque l'utilisateur quitte le programme, toute la mémoire doit être libérée.

Si l'utilisateur demande une opération non possible (par exemple assigner une valeur de pixel alors qu'aucune image n'est présente), un message d'erreur s'affiche et le menu est affiché.

Testez votre programme et vérifiez la gestion de la mémoire grâce au débugger *valgrind*.

Bonus

Ajoutez une nouvelle opération : la rotation de l’image de 90° dans le sens horlogique autour du pixel (0,0). Enjoy ! 😈