CS-119(a) – ICC-C Mini-Projet
Séries 8 - 9

Bob a trouvé un post-it dans l'amphi après le dernier cours de programmation. Il ne contenait que le texte

```
ICC = Implémenter Candy Crush
```

Il a déduit que c'est le sujet du mini-projet de cette année ! Il s'est mis au travail et a écrit son code. Juste avant de faire tourner son projet, il a découvert que son ordinateur avait un virus, et que les définitions de certaines fonctions ont été effacées. Puisqu'il a prévu de partir en vacances ces prochaines semaines, c'est à vous de refaire ces fonctions !

La version ICC de Candy Crush a les règles suivantes.

— Le jeu se déroule sur un tableau **carré** de taille fixe.

— Au début, chaque case contient un chiffre entre 1 et 7. On garantit qu'il n'existe pas trois cases consécutives sur la même ligne ou sur la même colonne avec le même chiffre.

— Le joueur peut effectuer l'**action** suivante : échanger n'importe quelles deux cases adjacentes sur une ligne ou sur une colonne (pas sur une diagonale).

— Un groupe de **trois ou plusieurs cases consécutives** sur la même ligne ou sur la même colonne contenant le même chiffre s'appelle **une combinaison**. Une case peut contribuer à plusieurs combinaisons.

— Quand le joueur forme une ou plusieurs combinaisons, on enlève le contenu des cases qui forment ces combinaisons, et les chiffres qui se trouvent au dessus "tombent" pour remplir l'espace libre se trouvant plus bas. Si de nouvelles combinaisons sont formées par les chiffres qui tombent, alors on les enlève aussi et on répète ce processus jusqu'à ce qu'il n'y ait plus de combinaisons à enlever.

— Une combinaison englobe le plus grand nombre de cases possible - on n'a pas le droit d'enlever partiellement la combinaison retrouvée.

— Des cases libres se forment naturellement en haut du tableau (à partir de la première ligne). Ces cases sont aléatoirement remplis de chiffres. Si de nouvelles combinaisons sont formées par les nouveaux chiffres, alors elles sont enlevées aussi et on répète le même processus.

— Quand aucune action du joueur ne peut créer une nouvelle combinaison le jeu se termine.
— Le score est mis à jour à chaque étape et est égal au nombre de cases formant les combinaisons qu'on arrive à enlever, sans compter à double les cases faisant partie de plusieurs combinaisons.

Les fonctions que vous devez implémenter sont décrites plus bas. C'est **essentiel** d'utiliser exactement la déclaration indiquée afin de faciliter l'évaluation automatique. Chaque fonction sera soumise à une suite de tests pour vérifier que votre code fonctionne correctement. La note dépendra du nombre de tests réussis par vos fonctions.

Le temps maximal d'exécution par test est d'**au plus 300ms**. Même si votre code finit par trouver le bon résultat, s'il prend trop longtemps pour le produire, le test sera quand même marqué comme échoué.

Pour tester une fonction, vous pouvez écrire une fonction `main()` qui lit un tableau carré bidimensionnel et le passe à votre fonction, par exemple :

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

int fonction(int **board, int size)
{
    printf("Votre fonction ici\n");
    return 0;
}

int main()
{
    int **board, size;

    scanf("%d", &size);
    board = malloc(size * sizeof(int *));
    for (int i = 0; i < size; i++)
    {
        board[i] = (int *)malloc(size * sizeof(int));
        for (int j = 0; j < size; j++)
            scanf("%d", &board[i][j]);
    }
    fonction(board, size);
    // On peut afficher des résultats ici
    for (int i = 0; i < size; i++)
        free(board[i]); // Et ensuite on libère
    free(board);        // la mémoire!
}
```

Exo1 Détecter et marquer

Écrivez une fonction qui reçoit un tableau bidimensionnel carré rempli de chiffres pouvant aller de `1` à `7` et qui marque les combinaisons détectées en inversant le signe des chiffres qui forment ces combinaisons. Le tableau peut aussi contenir des `0`, ce qui signifie un chiffre manquant, ou un trou. **Les zéros ne contribuent pas aux combinaisons**, il ne faut pas y toucher. La fonction doit retourner **le nombre de cases** détectées comme faisant partie d'une combinaison.

Par exemple, si le tableau est

```
 7  3  3  4  1  2
 2  3  3  3  0  5
 1  5  3  1  0  7
 2  2  3  3  0  7
 2  5  6  3  2  7
 1  1  1  2  3  2
```

alors votre fonction doit le modifier de la manière suivante :

```
 7  3 -3  4  1  2
 2 -3 -3 -3  0  5
 1  5 -3  1  0 -7
 2  2 -3  3  0 -7
 2  5  6  3  2 -7
-1 -1 -1  2  3  2
```

et retourner `12`.

Dans cet exemple, il y a quatre combinaisons :

— Trois `3` successifs (ligne 2, colonne 2, horizontal),
— Quatre `3` successifs (ligne 1, colonne 3, vertical),
— Trois `7` successifs (ligne 3, colonne 6, vertical), et
— Trois `1` successifs (ligne 6, colonne 1, horizontal).

Le `3` de la ligne 2 / colonne 3 contribue à deux combinaisons, mais puisqu'on compte les cases qu'on vide, on va compter ce `3` une seule fois pour obtenir un score de `12`.

Pour assurer la compatibilité avec le reste du code, la fonction doit impérativement avoir la signature suivante :

```
1    int flip(int **board, int size);
```

Ici, `board` est un pointeur vers un tableau de pointeurs, et peut donc être utilisé comme un tableau bidimensionnel. Le paramètre `size` indique le nombre de colonnes et de lignes du tableau (dans l'exemple ci-dessus il serait égal à `6`).

Exo2 La chute

Après avoir marqué les combinaisons, on aimerait faire tomber les chiffres qui restent. Mais pas tous d'un coup ! Dans notre jeu on aimerait montrer l'animation de la chute des chiffres. Pour cette raison, on aimerait simuler une étape à la fois.

Écrivez une fonction qui reçoit un tableau 2d carré représentant la table de jeu, ainsi que sa taille. Pour chaque colonne, la fonction identifie la case la plus basse contenant un zéro et fait descendre d'une case tous les chiffres se trouvant au dessus de cette case. Les cases laissées vides tout en haut du tableau recevront la valeur `0`. On garantit que le tableau ne contient que des chiffres entre `1` et `7`, ou alors des `0`.

La fonction doit retourner `1` si le tableau a été modifié, c'est à dire si des chiffres ont changé de case, ou alors `0` si aucun chiffre ne peut descendre plus bas et que le tableau est resté inchangé.

Pour assurer la compatibilité avec le reste du code, la fonction doit impérativement avoir la signature suivante :

```
1    int collapse(int **board, int size);
```

Ici, `board` est un pointeur vers un tableau de pointeurs, et peut donc être utilisé comme un tableau bidimensionnel. Le paramètre `size` indique le nombre de colonnes et de lignes du tableau.

Par exemple, si on reçoit en paramètre le tableau suivant de taille `6 x 6`

```
7  3  0  4  1  2
2  0  0  0  4  5
1  5  0  1  2  0
2  2  0  3  1  0
2  5  6  3  2  0
0  0  0  2  3  2
```

un appel à la fonction `collapse` doit apporter les modifications suivantes :

```
0  0  0  0  1  0
7  3  0  4  4  2
2  0  0  1  2  5
1  5  0  3  1  0
2  2  0  3  2  0
2  5  6  2  3  2
```

et la fonction doit retourner `1`. Noter le fait que le `0` à la ligne 2 / colonne 2 descend avec les autres chiffres de la colonne.

Par contre si on reçoit le tableau suivant de taille `4 x 4`

```
7  0  0  0
2  3  0  0
1  5  0  1
5  3  0  3
```

la fonction doit retourner `0` et le tableau doit rester inchangé car aucun chiffre ne se trouve au dessus d'un zéro.

Exo3 Crush AI

La dernière fonction effacée par le virus est celle qui décide si le jeu est terminé ou pas. Écrivez donc une fonction qui cherche une action possible pour continuer le jeu. Si une telle action est trouvée, alors la fonction retourne `1`, sinon le jeu est terminé et la fonction retourne `0`.

Pour assurer la compatibilité avec le reste du code, la fonction doit impérativement avoir la signature suivante :

```
1  int suggest_move(int **board,
2                   int size,
3                   int *pr,
4                   int *pc,
5                   int *pswap);
```

`board` est un pointeur vers un tableau de pointeurs, et peut donc être utilisé comme un tableau bidimensionnel. Le paramètre `size` indique le nombre de colonnes et de lignes du tableau. Les pointeurs `pr` et `pc` recevront la position de la case où le joueur pourrait effectuer une action, et le pointeur `pswap` recevra l'action à effectuer :

- On affecte `1` à `*pswap` si on veut échanger le contenu de la case se trouvant à `*pr`, `*pc` avec la case du dessus, c'est à dire `*pr - 1`, `*pc`,
- On affecte `2` à `*pswap` si on veut échanger le contenu de la case se trouvant à `*pr`, `*pc` avec la case du dessous, c'est à dire `*pr + 1`, `*pc`,
- On affecte `3` à `*pswap` si on veut échanger le contenu de la case se trouvant à `*pr`, `*pc` avec la case de gauche, c'est à dire `*pr`, `*pc - 1`, ou
- On affecte `4` à `*pswap` si on veut échanger le contenu de la case se trouvant à `*pr`, `*pc` avec la case de droite, c'est à dire `*pr`, `*pc + 1`.

Une case est identifiée par la ligne et la colonne où elle se trouve, les deux valeurs pouvant aller de 0 à `size - 1`.

Par exemple, pour le tableau `5 x 5`

```
1 3 4 4 7
7 1 5 5 2
1 2 2 6 2
4 7 7 3 1
4 2 2 1 1
```

la fonction retourne `1` et n'importe laquelle des actions suivantes est juste :

- Échanger les cases se trouvant à la ligne 2 et aux colonnes 1 et 2 pour former une combinaison de trois `1` à la verticale, donc n'importe laquelle des actions suivantes :
  - `*pr = 1, *pc = 0, *pswap = 4` - échanger le contenu de la case à la ligne 2 / colonne 1 avec la case se trouvant à sa droite, ou
  - `*pr = 1, *pc = 1, *pswap = 3` - échanger le contenu de la case à la ligne 2 / colonne 2 avec la case se trouvant à sa gauche.
- Échanger les cases se trouvant à la ligne 3 et aux colonnes 4 et 5 pour former une combinaison de trois `2` à l'horizontale, donc n'importe laquelle des actions suivantes :
  - `*pr = 2, *pc = 3, *pswap = 4` - échanger le contenu de la case à la ligne 3 / colonne 4 avec la case se trouvant à sa droite, ou
  - `*pr = 2, *pc = 4, *pswap = 3` - échanger le contenu de la case à la ligne 3 / colonne 5 avec la case se trouvant à sa gauche.

Pour le tableau `5 x 5`

```
1 3 4 4 7
7 6 5 5 2
1 2 2 6 7
4 7 7 3 1
4 2 2 1 1
```

la fonction retourne `0` car aucune action ne peut créer une combinaison.