Programmation en C et structures de données

guillaume.revy@univ-perp.fr

CC2, Jeudi 21/12/2023 de 14h à 18h

Ce programme est à réaliser seul, sous environnement GNU/Linux. Il doit être laissé dans le répertoire ProgC-CC2-2023/ à la racine du compte examen, sous forme d'un unique fichier `<nom_prenom.c>`. Seuls les documents disponibles dans ce réportoire sont autorisés. Tout autre document (cours, exercices, ...), ou autre matériel (téléphone portable, ...) est interdit.

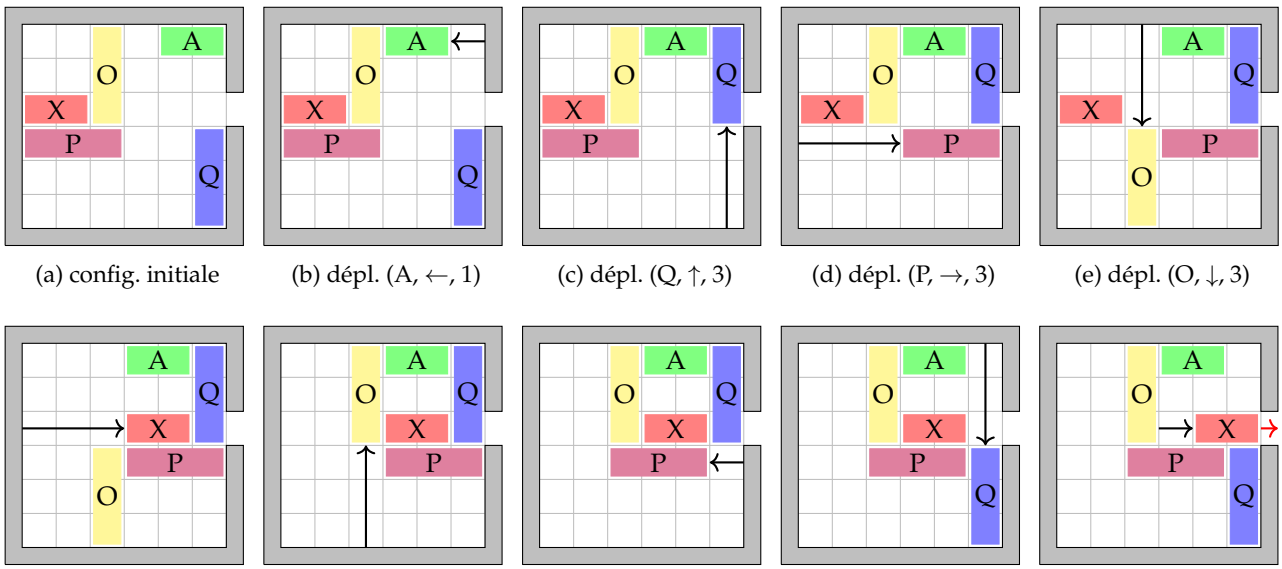
Conseils :

- Il est à noter que la qualité primera sur la quantité : un programme court, clair, bien conçu et qui compile sera mieux noté qu'un gros programme mal conçu, incompréhensible, voire qui ne compile pas. Utilisez des noms de variables explicites, indentez votre programme, et n'hésitez pas à le commenter.
- Lisez tout le sujet avant de commencer.
- Veuillez respecter les noms de fonctions et structures proposées dans le sujet.
- Le sujet est assez court : une attention particulière devra être apportée à l'allocation dynamique et la libération de la mémoire ... Pensez à utiliser `valgrind`!

Un programme qui ne compile pas sera considéré comme faux, et il entrainera la note de 0**.**

Le Rush Hour est un jeu qui se joue sur une grille de taille 6×6. Cette grille représente une aire de stationnement sur laquelle un ensemble de véhicules est positionné. Ceci est illustré en Figure 1a. Le but est d'extraire le véhicule rouge par l'unique sortie situé à droite du plateau. Pour ce faire, le joueur doit déplacer les véhicules :

- vers la gauche ou vers la droite pour les véhicules horizontaux, comme sur la Figure 1b,
- et vers le haut ou vers le bas pour les véhicules verticaux, comme sur la Figure 1c.



(a) config. initiale (b) dépl. (A, ←, 1) (c) dépl. (Q, ↑, 3) (d) dépl. (P, →, 3) (e) dépl. (O, ↓, 3)

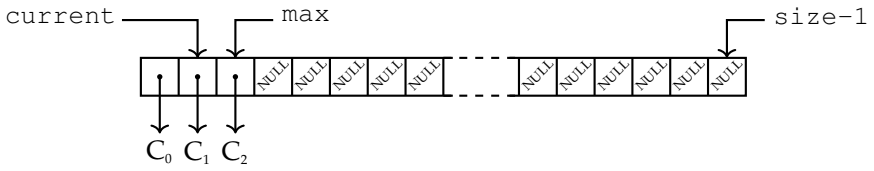
(f) dépl. (X, →, 3) (g) dépl. (O, ↑, 3) (h) dépl. (P, ←, 1) (i) dépl. (Q, ↓, 3) (j) dépl. (X, →, 1)

FIGURE 1 – Exemple d'une partie *beginner* de Rush Hour.

L'objectif du TP est d'écrire un programme qui, étant donnée une configuration initiale, affiche la suite de mouvements permettant de sortir le véhicule rouge.

1. Modélisation des véhicules et configurations

Dans ce TP, une partie de Rush Hour est vue comme une liste de configurations, représentant un ensemble de configurations atteignables depuis la configuration initiale (notée C_0), dont la configuration courante. Étant donné que nous ne connaissons pas à l'avance le nombre de configurations que nous allons manipuler, cette liste est représentée pour un tableau dynamique de configurations, que nous réallouons au besoin. Pour que tout le monde parte sur de bonnes bases, la structure `rush_hour` est illustrée ci-dessous de manière graphique.



En plus du tableau de configurations et des paramètres `current` (indice dans le tableau de la configuration courante), `size` (taille du tableau) et `max` (indice de la dernière configuration stockée), la structure contient également le nombre total de véhicules dans la partie (qui est bien évidemment le même pour toutes les configurations manipulées).
Dans cette structure, une configuration représente la position de chaque véhicule, toutes stockées dans un tableau. *Il est a noté qu'au maximum 16 véhicules peuvent être utilisés dans une partie de Rush Hour. Donc pour simplifier, nous utilisons un tableau statique de taille 16 pour stoker la position de tous les véhicules d'une configuration donnée.* En particulier, une configuration C est caractérisée par :

- un tableau de au plus 16 véhicules,
- l'indice `prec` du prédécesseur de C dans la liste des configurations, c'est-à-dire, l'indice de la configuration C_{prec} dont est issue la configuration C,
- le mouvement (véhicule, direction, longueur) permettant le passage de C_{prec} à C.

Prenons l'exemple de la partie de la Figure 1. La configuration initiale C_0 est d'indice 0 dans la liste (Figure 1a). La configuration C de la Figure 1b est obtenue à partir d'un mouvement sur la configuration initiale : l'indice `prec` du prédécesseur de C est 0, et le mouvement est (2, $\leftarrow$, 1) où 2 est l'indice du véhicule A dans le tableau de véhicules et 1 la longueur du déplacement.
Enfin, dans une configuration, un véhicule est représentée par :

- les coordonnées (ligne, colonne) de la case la plus à gauche (pour les véhicules horizontaux) ou la case la plus haute (pour les véhicules verticaux),
- la longueur, l'orientation, et un nom.

Par exemple, sur la Figure 1a, le véhicule X est de coordonnées (2,0) d'orientation horizontale et de longeur 2, alors que le véhicule Q est de coordonnées (3,5), d'orientation verticale et de longeur 3.

► **1.** Définir les structures `vehicule` et `configuration`.

► **2.** Définir enfin la structure `rush_hour`.

► **3.** Écrire la fonction `increase_size` qui prend en paramètre une structure `rush_hour` et un entier n, puis qui augmente de n éléments la taille du tableau de configurations.

2. Lecture et affichage

La configuration initiale C_0 est sockée dans un fichier, que vous devez lire. Ce fichier contient un certain nombre de lignes, chaque ligne correspondant à une voiture, la première ligne étant la position du véhicule rouge (celui à sortir). Plus précisément, une ligne contient :

- les coordonnées (ligne, colonne) de la case la plus à gauche (pour les véhicules horizontaux) ou la case la plus haute (pour les véhicules verticaux),
- la longueur (en nombre de cases), l'orientation du véhicule ('h' pour horizontale et 'v' pour verticale), et un nom (sous forme d'un caractère).

Par exemple, le fichier de la configuration initiale en Figure 1a est donné ci-dessous.

```
2 0 2 h X
3 0 3 h P
0 4 2 h A
0 2 3 v O
3 5 3 v Q
```

D'autres fichiers sont disponibles dans le répertoire ProgC-CC2-2023.

► **1.** Écrire la fonction `allocate` qui, étant un nom de fichier, alloue la mémoire nécessaire à une partie de Rush Hour, lit le contenu du fichier pour construire la configuration initiale, l'ajoute à l'indice 0 du tableau de configurations, initialise `current` et `max`, et renvoie la structure correspondante.

► **2.** Écrire la fonction `deallocate` qui libère la totalité de la mémoire préalablement allouée pour une partie de Rush Hour passée en paramètre.

► **3.** Écrire enfin la fonction `print` qui affiche la configuration courante d'une partie de Rush Hour passée en paramètre.

Par exemple, pour la configuration initiale de la Figure 1a, l'affichage pourra être le suivant.

```
+------+
|..O.AA|
|..O...|
|XXO...
|PPP..Q|
|.....Q|
|.....Q|
+------+
```

3. Résolution automatique

La dernière étape consiste en la résolution automatique de la partie de Rush Hour. L'algorithme est le suivant.

1. Nous initialisons une partie de Rush Hour à partir d'un fichier dont le nom est donné en ligne de commande. La configuration initiale est alors en indice 0, qui est la configuration courante.
2. Nous déterminons toutes les configurations atteignables depuis la configuration courante en au plus un mouvement de véhicule. Toutes ces configurations sont ajoutées à la suite dans le tableau des configurations, en réallouant le tableau si nécessaire.
3. Nous passons à la configuration courante suivante : si la configuration courante est d'indice i, nous passons à la configuration courante d'indice $i + 1$.
4. Et nous répétons les étapes 2 et 3 tant qu'aucune configuration gagnante n'est trouvée.

Une configuration gagnante est une configuration où le véhicule rouge est sur le point de sortir, c'est-à-dire, que sa case la plus à gauche est en coordonnées $(2, 4)$. Cet algorithme consiste en le *parcours en largeur d'abord* du graphe des configurations possibles. Ainsi dès lors qu'une configuration gagnante est trouvée, elle minimise le nombre de mouvements. *Mais attention, quand vous ajoutez une configuration dans le tableau des configurations atteignables, vérifiez si elle n'y est pas déjà, sinon, vous allez peut-être tourner en rond !*

► **1.** Écrire la fonction `move` qui, étant donnés une partie de Rush Hour (structure `rush_hour`), l'indice k du véhicule à déplacer, la direction d et la longueur ℓ du déplacement, essaie d'effectuer ce déplacement : si ce déplacement est possible, elle retourne le configuration obtenue, sinon, elle renvoie NULL.

► **2.** Écrire la fonction `generate_configurations` qui, étant donnée une partie de Rush Hour, génère l'ensemble des configurations atteignables depuis la configuration courante en au plus un mouvement de véhicule.

► **3.** Écrire la fonction `print_moves` qui, une fois une configuration gagnante trouvée, affiche l'ensemble des mouvements nécessaires pour l'atteindre depuis la configuration initiale.

► **4.** Écrire la fonction `rush_hour` qui, étant donnée une partie de Rush Hour, génère les configurations atteignables depuis la configuration initiale en utilisant l'algorithme précdédent, jusqu'à obtenir une configuration gagnante. Si la configuration initiale n'est pas résolvable, elle l'indique, sinon, elle affiche l'ensemble des mouvements utilisés pour la résoudre.

► **5.** Écrire enfin la fonction `main` qui lit le nom de fichier contenant la configuration initiale en ligne de commande, puis qui tente de la résoudre.

Pour écrire ces fonctions, vous pouvez avoir besoin des fonctions :
— `is_on_cell` qui teste si un véhicule est sur la case de coordonnées (x, y) de l'aire de stationnement,
— et `equals` qui teste si deux configurations sont identiques, c'est-à-dire, si chaque véhicule est au même endroit sur les deux configurations.

Saurez-vous sortir à temps votre véhicule de l'aire de stationnement pour partir en vacances ?