TP 3&4 : exploration de labyrinthes

On se propose dans l'exercice suivant de parcourir un labyrinthe que l'on aura préalablement représenté par un tableau à une dimension. L'algorithme de parcours repose sur l'utilisation d'une **pile**.

1. Construction et affichage de labyrinthes

Les labyrinthes que nous allons utiliser sont des labyrinthes rectangulaires, composés de cases franchissables ou infranchissables. Les cases du bord seront toujours infranchissables sauf en haut à gauche (case de coordonnées **(0,1)**) et en bas à droite (dépend de la longueur et largeur du labyrinthe) : ce seront les cses de départ et d'arrivée de notre labyrinthe.

1.1 Affichage du labyrinthe

Ecrivez une fonction qui dessine le labyrinthe en mode texte. Pour ce faire, on définit au préalable quelques macros en C.

```
#define FERME 0
#define OUVERT 1
```

Vous pouvez tester cette fonction à l'aide de l'exemple suivant :

```
#define LIGNES 5
#define COLONNES 6


int creation_laby(int* laby, float proba){
    for (int i=0; i<LIGNES; i++){
        for (int j=0; j<COLONNES; j++){
             if ((rand()%10 <= proba) || (i==0 && j==1)
                     || (i==1 && j==1) || (i==LIGNES-1 && j==COLONNES-2)){
                *(laby +(i*COLONNES)+j) = OUVERT;
             }
             else {
                *(laby +(i*COLONNES)+j) = FERME;
             }
             // fermer les contours
             if((i==0     &&     j!=1)     ||     (j==0)     ||     (j==COLONNES-1)
                       || (i==LIGNES-1 && j!=COLONNES-2)){
                *(laby +(i*COLONNES)+j) = FERME;
            }
        }
    }
    return(1);
}

int main(int argc, char* argv[]){
      int *laby = (int*) calloc(sizeof(int), LIGNES*COLONNES); // tout sur une ligne
      creation_laby(laby, proba);
      …
      return(1);
}
```

L'affichage devra correspondre à :

```
X XXXX
X    X
X    X
X    X
XXXX X
```

Dans la suite de l'exercice, on conviendra que la longueur sera de 70 lignes et la largeur de 30 colonnes. De plus, ce labyrinthe sera représenté par une variable globale dans le programme construit.

1.2 Création du labyrinthe

Ecrivez une fonction qui prend en entrée la probabilité qu'une case soit franchissable et affecte aléatoirement les cases de notre labyrinthe de taille longueur/largeur. (On rappelle que la fonction **`rand`** permet d'obtenir un entier aléatoire).
Le programme devra proposer des labyrinthes à l'utilisateur jusqu'à ce que celui-ci accepte l'un d'entre eux (N'oubliez pas de changer la graine de votre générateur d'entiers aléatoires avec la fonction **`srand()`** dans la fonction **`main()`**).

2. Une pile

Pour parcourir le labyrinthe, nous allons utiliser une pile composée elle-même de « *pas* ». Un pas correspond à **4 paires de coordonnées** : les coordonnées (**x, y**) du point de départ du pas et les coordonnées (**z, t**) de son point d'arrivée.
La pile contiendra les pas élémentaires entre deux cases contigües à explorer ; les coordonnées de départ nous permettront d'afficher aisément les pas.
Définissez la ou les structures à utiliser par le programme.

2.1 Structure de la pile

Implantez une pile globale à l'aide d'un tableau et écrivez les fonctions classiques de manipulation de pile (empiler - *push*, dépiler - *pop*, etc.) [cf. cours]

2.2 Algorithme de parcours de labyrinthe

Pour parcourir notre labyrinthe (« *en profondeur d'abord* »), nous allons utiliser une variable représentant notre pile et qui contiendra les pas à explorer. Au début du parcours, elle ne contient que le seul pas allant de (0,1) à (1,1). On ajoute ensuite à la pile tous les pas possibles (c'est-à-dire les déplacements pour lesquels la case d'arrivée est franchissable et pas déjà explorée) à partir du point d'arrivée du pas considéré (Par commodité, **on ne se déplacera pas en diagonale**).

Il s'agit en réalité d'explorer un chemin jusqu'à ce que l'on ne puisse plus avancer dans le labyrinthe. Ce cas arrive lorsque l'on a soit parcouru tout le labyrinthe et trouvé la sortie, soit lorsque l'on est arrivé sur un pas qui n'a pas de successeurs. Dans ce cas, il faut rebrousser chemin dans notre exploration et partir d'un pas qui n'a pas été visité.

Testez cet algorithme sur l'exemple donné plus haut et déduisez-en l'usage de la pile.

2.3 Implantation de l'algorithme de parcours

Implantez une fonction **`parcourir()`** qui parcourt le labyrinthe selon la méthode décrite ci-dessus. Attention, n'oubliez pas que tous les labyrinthes n'ont pas forcément une sortie !

Vous pourrez afficher à chaque étape l'état de l'exploration du labyrinthe (en ajoutant un **o** aux cases explorées)

Remarque : Notez que le parcours obtenu n'est sans doute pas **optimal** !
Pour ce faire, il nous faudrait utiliser non pas une pile mais une autre structure (la file) afin d'explorer le labyrinthe en largeur d'abord.