Mini-projet, L1 informatique
Labyrinthe

À partir d'une grille rectangulaire p × q on obtient un labyrinthe en ajoutant un certain nombre de murs pour séparer deux cases voisines. On convient en outre que ce labyrinthe est entouré de murs sur toute sa périphérie, de sorte qu'il n'est pas possible de sortir de la grille.

Utilisation de la Pile :

Dans ce projet nous utilisons la structure pile et les fonctions associées. Nous supposons donc que vous disposez de cet outil :

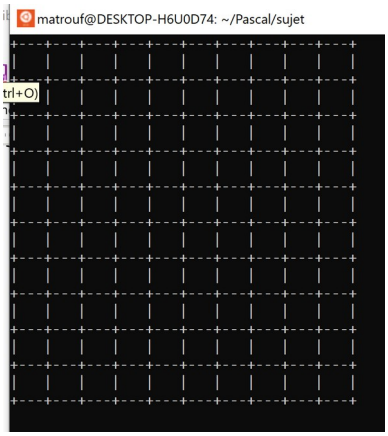
```
struct laby
{
	int p;
	int q;
	Case **tab;
};
struct couple
{
	int i;
	int j;
};
struct pile
{
	int nbe;
	int taille;
	couple *T;
};
void init(pile & p, int taille)
{
	p.taille=taille;
	p.nbe=0;
	p.T=new couple[taille];
}
bool empiler(pile & p, couple c)
{
	if((p.nbe)<p.taille)
	{
		p.T[p.nbe]=c;
		(p.nbe)++;
		return true;
	}
	else
	{
	    cout<<"la pile est pleine"<<endl ;
		exit(1) ;
	}
}
couple depiler(pile & p)
{
	if(p.nbe>0)
	{
		p.nbe--;
		return(p.T[p.nbe]);
	}
	else
	{
		cout<<"la pile est vide"<<endl;
		exit(1);
	}
}
bool vide(pile & P)
{
	if(P.nbe==0) return true;
	else return false;
}
```

I Dessiner et parcourir un labyrinthe :

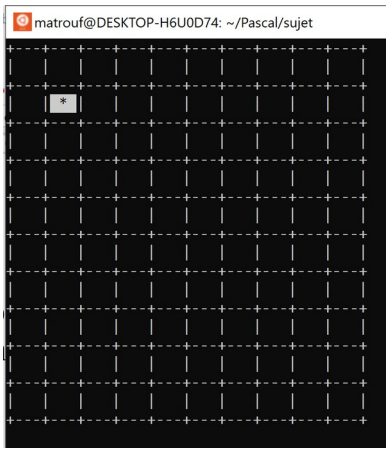
Pour l'affichage nous utiliserons la bibliothèque ***<ncureses.h>***, vous devez compiler en linkant avec la librairie ***ncurses***. Nous vous mettons à disposition les fonctions suivantes :

- void ShowLab(laby & L) ;
- void Mark_Case(int i, int j) ;
- void UMark_Case(int i, int j) ;
- int LireCommande()
- void clear()
- void mvprintw(int ligne, int colonne, const char *)

« ***void ShowLab(laby & L) ;*** » permet d'afficher un labyrinthe. Dans l'exemple suivant toutes les portes sont fermées :



« ***void Mark_Case(int i, int j) ;*** » permet de marquer la case i et j du labyrinthe. Par exemple sur le labyrinthe ci-dessus, ***Mark_Case(1, 1) ;*** ferait :



« ***void UMark_Case(int i, int j) ;*** » permet de dé-marquer la case i et j du labyrinthe.

« ***clear() »*** efface l'écran

« ***mvprintw »*** permet permet d'afficher une chaîne

La fonction « ***int LireCommande()*** » est une fonction qui retourne :
1) - 1 si on appuie sur la touche F1
2) 3 si on appuie sur la touche gauche
3) 9 si on appuie sur la touche droit
4) 12 si on appuie sur la touche haut
5) 6 si on appuie sur la touche bas

Afin que vous puissiez tester vos programme nous mettons à votre disposition un fichier contenant un labyrinthe sous format texte. Voici le format qu'on vous propose :

p q W N S E W N S E W N S E …..

p : le nombre de lignes
q : le nombre de colonnes
si W==1 alors la porte ouest est ouverte
si N==1 alors la porte Nord est ouverte
…
...

La fonction

laby * lire_Laby(laby & L) ;

Est donnée dans le fichier « votre_prog.cpp ». Ce fichier sera votre point de départ.

II-
II.1 Lire le « votre_prog.cpp », comprenez puis compilez avec :

>>g++ votre_prog.cpp labIO.cpp -o votre_prog -lncurses

Exécutez pour voir …

II.2 Écrire une fonction qui écrit un labyrinthe dans un fichier.

void ecrire_Laby(char *fn) ;

Testez la fonction.

III- Déplacement dans le lanyrinthe
Ecrire une fonction «***void deplacement(laby & L)*** » qui marque la case (0,0) et qui marque les cases les unes après les autres selon la commande de l'utilisateur. La commande de l'utilisateur est saisie grâce à la fonction « ***LireCommande()*** », décrite ci-dessus.

Une commande invalide conduit à l'arrêt de la partie : « échec »
Si l'utilisateur arrive à la destination, c'est-à-dire à la case (p-1,q-1) alors « gagner »

IV- La fonction création :
Nous allons maintenant nous intéresser à la génération « aléatoire » de labyrinthes. Il n'existe pas de solution canonique à ce problème , aussi allons nous nous restreindre à la création de labyrinthe dits parfaits, c'est-à-dire lorsque deux cases quelconques peuvent toujours être reliées par un unique chemin. La méthode que nous choisissons d'implémenter consiste à partir d'un labyrinthe où toutes

les portes sont fermées. Une case est choisie arbitrairement et marquée comme étant « visitée ». On détermine ensuite quelles sont les case voisines et non visitées, on en choisit une au hasard que l'on marque comme ayant été « visitée », et on ouvre les portes les séparant. On recommence ensuite avec la nouvelle case. Lorsqu'il n'y a plus de case voisine non visitée, on revient à la case précédente. Lorsqu'on revient à la case de départ et qu'il n'y a plus de possibilités, le labyrinthe est terminé. Écrire la fonction:

laby * My_Creation(int p, int q) ;

Montrez que ça marche grâce à la fonction Show_Laby.

Voici l'algorithme :
0- créer un labyrinthe dynamiquement : L
1- céer une pile
2- on empile (0,0)
3- tant que la pile n'est pas vide:
    c=dépiler()
    si il existe c' accessible depuis c et non visité:
        marquer c' visité
        ouvrir les portes entre c et c'
        re-empiler c
        emipler c'
4- retourner L

V Recherche d'un chemin dans le labyrinthe

Maintenant que le labyrinthe est créé, nous allons nous intéresser à la manière d'en sortir. Le principe de l'algorithme que nous vous proposons est le suivant : lorsqu'on arrive sur une case, celle-ci est marquée puis on se rend sur l'une de ses voisines non encore marquée. Lorsqu'il n'en existe pas, on revient sur nos pas jusqu'à la dernière intersection possédant une branche non encore explorée. Cette démarche porte le nom de parcours en profondeur car chaque route est explorée dans son entier avant d'en explorer une nouvelle.

Pour ce faire nous utiliserons, encore une fois, une pile dans la quelle on empile la case de départ en la marquant « explorée ». Ensuite on rentre dans une boucle dans laquelle on commence par dépiler une case appelée c (i,j). S'il existent encore des cases accessibles à partir de c non encore explorées alors on en choisi une, que nous appelons c'. On ré-empile c, on marque la case c' « explorée » et on l'empile et recommence au début de la boucle. Sinon (c'est-à-dire si toutes les cases voisines et accessibles à partir de c sont déjà explorées) on ne fait rien (on ne ré-empile pas c).
La boucle s'arrête lorsqu'on empile la case de destination. Dans ce mini-projet on considère la (0,0) comme étant la case de départ et la case (p-1,q-1) comme étant la case de destination.

Écrire la fonction :

Pile *explorer(laby & L) ;

Qui parcourt le labyrinthe à la recherche du chemin entre la case de départ ***Dep*** et la case de destination ***Dest***. La pile retournée contient le chemin qui mène de ***Dep*** à ***Dest***

VI. Fonction d'affichage du chemin trouvé par l'ordinateur.

Écrire une fonction qui réinitialise l'affichage et affiche le labyrinthe ainsi que le chemin qui mène de ***Dep*** à ***Dest :***

void affiche_chemin(Pile & P) ;

VII. Jeu

Enfin vous devez écrire une fonction jeu qui permet de :

- générer un labyrinthe.
- permettre au joueur de se déplacer (fonction déplacement) pour trouver un chemin.
- En cas d'abandon (touche F1) vous avez le choix entre :
    - afficher la solution (***affiche_chemin)***
    - modifier la fonction explorer pour qu'elle affiche les différentes tentatives d'exploration. Vous marquez les cases (***void Mark_Case(int i, int j);*** au fur et à mesure de leurs empilements et vous dé-marquez quand vous dépilez (***UMark_Case(int i, int j) ;***