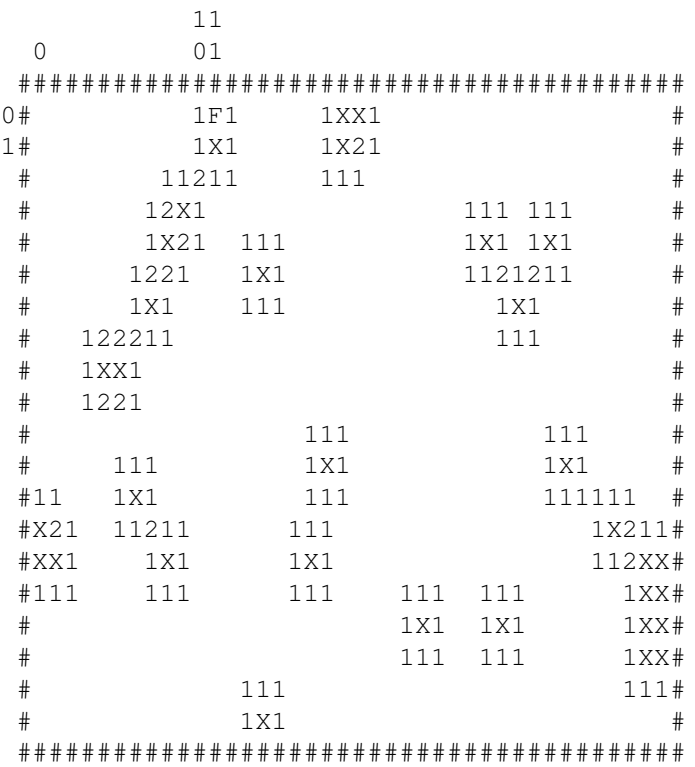
Programmation en C avancée

CC1: Jeudi 3/12/2020 de 9h30 à 12h30

Démineur

L'objectif est d'écrire un programme qui implante le jeu du démineur. Les règles sont simples : localiser des mines cachées dans une grille, avec pour seule indication le nombre de mines dans les cases adjacentes. Un exemple de grille est donné ci-dessous.



Sur une grille de démineur, une case peut être :

1. non dévoilée et non marquée (matérialisée par un 'X', ie. case $(1, 11)$),
2. non dévoilée et marquée d'un drapeau (matérialisée par un 'F', ie. case $(0, 11)$),
3. dévoilée et vide, c'est-à-dire, ne contenant pas de mines, ni de mines dans les cases adjacentes (matérialisée par un espace, ie. case $(0, 0)$),
4. dévoilée et indiquant le nombre de mines dans les cases adjacentes (ie. case $(0, 10)$).

Dès lors que l'utilisateur dévoile une case sur laquelle se trouve une mine, il a perdu.

Ce programme est à réaliser seul, sous environnement GNU/Linux. Il doit être rendu sur Moodle, sous forme d'un unique fichier `nom_prenom.c`. Seuls les documents disponibles sur Moodle sont autorisés. Aucun autre document (cours, exercices, ...), ni autre matériel (téléphone portable, ...) n'est autorisé.

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, et n'hésitez pas à commenter votre programme.

— Lisez tout le sujet avant de commencer.

1. Exercice préliminaire

Soit t un tableau d'entiers de taille n. Ce tableau contient k entiers positifs ou nuls, avec $k \in \{0, \cdots, n\}$, puis $n - k$ entiers égaux à -1, comme le tableau ci-dessous pour $(n, k) = (8, 5)$:

$$t = \{1, 3, 0, 2, 5, -1, -1, -1\}.$$

► **1.** Écrire une fonction `ajouter` qui ajoute un nouvel élément positif ou nul au tableau, après les k premiers éléments. Par exemple :

$$\texttt{ajouter}(t, 17) \quad \rightarrow \quad \{1, 3, 0, 2, 5, 17, -1, -1\}.$$

► **2.** Écrire une fonction `retirer` qui retire et retourne l'élément d'indice 0 du tableau. Par exemple :

$$\texttt{retirer}(t) \quad \rightarrow \quad 1, \{3, 0, 2, 5, 17, -1, -1, -1\}.$$

Remarque 1 *Vous remarquerez que ceci permet d'implanter une file, à l'aide d'un tableau. Si le tableau t ne contient que des éléments -1, la file est vide. Sinon, la fonction* `ajouter` *permet d'enfiler les éléments, et la fonction* `retirer` *permet de les défiler.*

2. Démineur en mode texte

Nous allons maintenant écrire un programme qui implante le démineur, en suivant les étapes suivantes.

► **1.** Définir une structure qui permet de représenter une case. Elle contiendra l'état de la case (non dévoilée et non marquée, non dévoilée et marquée d'un drapeau, ou dévoilée), le nombre de mines sur les cases adjacentes, et un booléen indiquant si la case contient elle-même une mine.

► **2.** Définir une structure qui permet de représenter une grille de démineur. Elle contiendra les dimensions de la grille, un tableau 2D de cases, le nombre total de mines, et le nombre de mines découvertes par l'utilisateur.

Remarque 2 *Pour le tableau 2D, vous utiliserez :*
- *soit des pointeurs et de l'allocation dynamique,*
- *soit un tableau 2D statique de grande taille (ie. 256×256), dans lequel vous stockerez la grille.*

► **3.** Écrire une fonction `creer` qui prend en paramètre les dimensions de la grille et le nombre total de mines, puis qui crée et renvoie une grille (ie. instance de la structure grille). La position des mines sera choisie de manière aléatoire, en utilisant la fonction `rand`, comme vu au TD2 (Exercice 5). Cette fonction mettra également à jour, par chaque case, le nombre de mines sur les cases adjacentes.

Remarque 3 *Vous pouvez manipuler cette grille en utilisant ou non un pointeur. Si vous utilisez un pointeur, il faudra au préalable allouer l'espace mémoire nécessaire.*

► **4.** Écrire une fonction `afficher` qui affiche une grille de démineur passée en paramètre, de la même manière que ce qui est présenté en Page 1. Pour chaque case, elle affichera 'X', 'F', un chiffre, ou un espace.

► **5.** Tester ces premiers éléments avec une fonction principale qui crée une grille et l'affiche.

Ensuite pour rappel, à chaque tour de jeu, l'utilisateur choisit une case. Si cette case est déjà dévoilée, l'action n'a aucun effet. Sinon (case non encore dévoilée), 4 cas de figure se présentent.

1. L'utilisateur pose ou enlève un drapeau sur cette case.
2. L'utilisateur dévoile cette case.
   (a) Cette case contient une mine, dans ce cas, l'utilisateur a perdu. La partie s'arrête.
   (b) Cette case possède au moins une case adjacente qui contient une mine, dans ce cas, la case est dévoilée et le nombre de mines adjacentes affiché.
   (c) Cette case est vide et toutes les cases adjacentes sont vides, également. La case est dévoilée et la même opération est répétée sur les cases adjacentes, et ce jusqu'à ce que la zone vide contenant cette case soit entièrement délimitée par des chiffres.

► **6.** Écrire une fonction `devoiler_case_vide`, qui permet de dévoiler la case vide de coordonnées (x, y) et répéter cette action sur les cases adjacentes (vides, également), jusqu'à créer une zone vide entièrement délimitée par des chiffres et contenant cette case (x, y).

Remarque 4 *Pour implanter cette fonction, 2 possibilités peuvent être envisagées.*

— *Soit vous définissez une fonction récursive, qui, appelée sur la case de coordonnées (x, y), la dévoile avant d'être appelée sur les adjacentes. Dans ce cas, pour éviter de recopier le tableau 2D de cases à chaque appel, vous favoriserez l'utilisation d'un pointeur en paramètre de la fonction.*

— *Soit vous définissez une fonction itérative, qui conservera une liste des cases à dévoiler. Plus particulièrement, elle pourra utiliser une file contenant au départ le numéro de la case à dévoiler. Ensuite, tant que cette file n'est pas vide, elle retirera (défilera) le premier élement, dévoilera la case correspondante, et ajoutera (enfilera) le numéro des cases adjacentes.*

Toute autre solution sera bien évidemment acceptée.

► **7.** Écrire enfin une fonction principale, qui permet de saisir le choix utilsateur, et d'effectuer l'action sur la case correspondante tant que l'utilisateur n'a pas perdu ou quitter la partie.

3. Démineur avec ncurses

La bibliothèque `ncurses` fournit une API pour le développement d'interfaces utilisateur, en utilisant les caractères et couleurs d'un mode semi-graphique. Elle prend notamment en charge la gestion des évènements clavier (pavé directionnel, ...).

Le listing ci-dessous montre un exemple simple de programme utilisant cette bibliothèque. Pour le compiler, il faut ajouter `-lncurses` à la ligne de compilation.

```
1  #include <stdio.h>
2  #include <ncurses.h>
3  #include <unistd.h>
4
5  int
6  main(void)
7  {
8    char action;
9
10   // ... initialisation de ncurses
11   initscr();
12   cbreak();
13   noecho();
14   keypad(stdscr, TRUE);
15   refresh();
16   curs_set(FALSE);
17   nodelay(stdscr, TRUE);
18
19   // ... boucle principale
20   while(action != 'Q') {
21     if((action = getch()) != ERR)
22       mvprintw(10, 4, "%c %d", action, action);
23     usleep(125000);
24   }
25
26   nodelay(stdscr, FALSE);
27   action = getch();
28
29   // ... arret de ncurses
30   endwin();
31
32   return 0;
33 }
```

Ici la fonction `mvprintw` (Ligne 22) est la fonction d'affichage : elle fonctionne de la même manière que `printf`, les deux premiers paramètres étant les coordonnées de l'affichage dans le terminal. Ce programme affiche donc l'action clavier en 10^{e} ligne et 4^{e} colonne, tant que celle-ci n'est pas le caractère 'Q'.

► **1.** Tester ce programme sur votre machine.

► **2.** Reprendre la fonction `afficher` pour afficher la grille de démineur dans le terminal en utilisant cette bibliothèque.

► **3.** Reprendre finalement la fonction principale, en utilisant cette bibliothèque. Le choix de la case pourra alors se faire en déplaçant un curseur sur la grille à l'aide du pavé directionnel.