IN 102 - Examen
20 octobre 2020
Goran Frehse

Énoncé

ENSTA ParisTech

— Le sujet comporte **4** pages et l'examen dure **2** heures.
— Les documents (polys, transparents, TDs, livres . . .) sont autorisés.
— Sont **absolument interdits :** le web, le courrier électronique, les messageries, les répertoires des camarades, le téléphone (même pour avoir l'heure puisque vous l'avez sur votre ordinateur).
— Votre travail sera en partie évalué par un mécanisme automatique. Vous devez respecter les règles de **nommage** des fichiers et des fonctions qui vous sont données. Le programme ne doit **afficher uniquement ce qui est explicitement demandé**, en respectant exactement le format (espaces, virgules, etc.).
— Dans les indications de format, le symbole ␣ représente un espace, ↵ représente un retour à la ligne.
— **En cas de non respect, la note peut être diminuée d'un point**.

Soumission :

— **À la fin de l'examen**, vous devrez créer une **archive** contenant **tous** les fichiers **sources** que vous avez écrits (`.c`, `.h`). Le nom de cette archive devra avoir la structure suivante (avec votre propre nom !) :
`nom_prenom.zip` ou `.tgz` (selon l'outil d'archivage que vous utilisez).

Pour créer l'archive, vous pouvez utiliser, par exemple :

```
zip -r nom_prenom.zip mondossierdexamen
tar czvf nom_prenom.tgz mondossierdexamen
```

En cas de non respect, la note peut être diminuée d'un point.

— Vous devrez **copier** cette archive dans le répertoire de rendu se trouvant à `~frehse/in102rendus/` avec la commande :

```
cp -vi nom_prenom.zip ~frehse/in102rendus/
```

Si la copie a réussie, vous verrez alors le message

```
nom_prenom.zip -> ~frehse/in102rendus/nom_prenom.zip
```

Questions : Jeu de démineur

Ecrivez un programme pour jouer une variante du jeu démineur. Sur une grille de N $\times$ M cases (N verticales, M horizontales), le joueur tentera de trouver K mines placées de façon aléatoire. Pour tester, on supposera $N = 3$, $M = 4$, $K = 3$, mais N, M, K doivent rester variables dans le programme. Au début, toutes les cases sont inexplorés, donc on les affiche avec un " ?". Pour le joueur, les mines sont donc "cachées". Par contre, le temps d'écrire et débogguer le programme on affichera même les mines cachées, en les indiquant avec " !". (Avant de jouer pour du vrai on remplacerait le " !" avec " ?", mais pour cet examen on garde le " !".) Une fois une mine est détectée, on l'affiche avec "*". Une case exploré mais vide est affiché avec ".".

Les cases de la grille sont numérotées ligne par ligne, par exemple pour une grille 3 x 4 :

0	1	2	3
4	5	6	7
8	9	10	11

A chaque tour, le joueur désigne une case avec le numéro correspondant. Si le joueur tombe sur une mine, il perd. Sinon, on lui dévoile les "environs" de la case, c'est à dire les 4 cases adjacentes : à gauche, à droite, dessus et dessous. Une mine cachée sur une de ces cases devient alors détectée.

La partie se termine si le joueur a sélectionné une case avec mine (perdu) ou s'il a detecté toutes les mines (gagné). Après chaque tour, le programme affichera le nombre de mines détectées ainsi que le nombre de mines qui restent à trouver. A la fin de la partie, il affichera soit "Gagné en X tours.", où X est le nombre de tours joués, ou "Perdu en X tours.".

Attention : Suivre soigneusement les étapes indiquées. Chaque étape rapporte des points, même si vous n'arrivez pas jusqu'au bout. Coder exactement les fonctions demandées. *Si vous ne trouvez pas la solution d'une sous-question, n'hésitez pas à continuer avec la sous-question suivante.*

Nommage : Le fichier source de ce programme devra s'appeler `demin.c`.

Format de sortie : Afficher la grille actuelle, suivie par un retour à la ligne, suivi par le nombre de mines détectées ainsi que le nombre de mines qui restent à trouver, un retour à la ligne, puis "numéro␣:␣", puis faire entrer le numéro. A la fin de la partie, afficher la grille actuelle, ensuite soit "Gagné en X tours.", où X est le nombre de tours joués, ou "Perdu en X tours.", suivi par un retour à la ligne. La grille est affichée avec les symboles " ?" (non explorée sans mine), " !" (non explorée avec mine), "." (explorée sans mine), "*" (mine détectée).

Ex. tests :

```
./demin
??!?
???!
?!??
0␣trouvées,␣3␣restantes
numéro␣:␣0
..!?
.??!
?!??
0␣trouvées,␣3␣restantes
numéro␣:␣5
..!?
...!
?*??
1␣trouvées,␣2␣restantes
numéro␣:␣6
..*?
...*
?*.?
3␣trouvées,␣0␣restantes
Gagné␣en␣3␣tours.
```

```
./demin
????
??!?
!??!
0␣trouvées,␣3␣restantes
numéro␣:␣5
?.??
..*?
!.?!
1␣trouvées,␣2␣restantes
numéro␣:␣11
?.??
..*?
!.?!
1␣trouvées,␣2␣restantes
Perdu␣en␣2␣tours.
```

Q1 La grille sera représentée par un tableau d'enum : INCONNU_SANS représentant une case non explorée sans mine, INCONNU_AVEC représentant une case non explorée avec mine, DEMINE une case déminée, MINE une case avec mine détectée.

Créer une fonction `int* creer_grille(int N, int M)` qui crée d'abord un tableau dont les cases sont inexplorées sans mine et qui ensuite retourne un pointeur vers ce tableau.

Q2 Écrivez une fonction `int afficher(int* grille, int N, int M)` qui prend en argument le tableau et affiche les cases sous forme de grille.

En dessous de la grille, elle affiche `X trouvées, Y restantes` où X est le nombre de mines trouvées et Y le nombre de mines encore inconnues. La fonction donne en valeur de retour le nombre de mines trouvées.

Voir les tests ci-dessus pour un exemple d'affichage.

Q3 Testez les fonctions précédentes dans une fonction `test_affichage()`. **À l'intérieur de cette fonction**, exécutez toutes les actions suivantes : Créez une grille avec des cases INCONNU_SANS, affichez-la. Marquez ensuite la case 4 comme explorée sans mine, les cases $2, 6$ comme inconnues avec mine et 8 comme avec mine détectée. Affichez le résultat.

Q4 Écrivez une fonction `int miner(int c, int* grille, int N, int M)` qui prend en argument le tableau et place une mine à la case numéro c. Si c contient déjà une mine, la fonction rend la valeur 0, sinon elle rend 1.

Q5 Testez les fonctions précédentes dans une fonction `int* test_miner()`. **À l'intérieur de cette fonction**, exécutez toutes les actions suivantes : Créez une grille avec des cases INCONNU_SANS. Placez ensuite 3 mines avec "miner". Affichez le résultat. La fonction donne en valeur de retour un pointeur vers la grille créée.

Q6 Écrivez une fonction

`void miner_alea(int* grille, int N, int M)`

qui prend en argument la grille et place une mine de façon aléatoire. Utilisez la fonction `miner(...)` pour place la mine.

Attention : Utiliser la valeur de retour de la fonction miner(...) pour tirer une autre case s'il y a déjà une mine sur la case choisie. On peut supposer qu'il reste au moins une case de libre.

Astuce : Si vous ne trouvez pas la solution avec placement aléatoire, passez à la prochaine question.

Pour tirer un nombre aléatoire entre 0 et $N-1$, vous pouvez faire appel au générateur de nombres aléatoires avec `rand()%N`. Pensez à initialiser le générateur de nombres aléatoires en plaçant l'appel `srand(time(0));` une seule fois au début du main(). Il faudra inclure `stdlib.h` et `time.h`.

Q7 Écrivez une fonction

`void miner_alea_k(int* grille, int N, int M, int K)`

qui prend en argument la grille et place K mines sur des cases choisies de façon aléatoire. On peut supposer qu'il reste au moins K cases libres. Utilisez la fonction `miner_alea(...)` pour placer les mines.

Astuce : Si vous ne trouvez pas la solution avec placement aléatoire, placez 3 mines à des positions fixes de votre choix et continuez avec la prochaine question.

Q8 Définissez une fonction

`void transformer(int c, int* grille, int N, int M)`

qui prend en argument la grille et un numéro de case c. La fonction modifie la case correspondante pour transformer un case inconnue avec mine en case mine détectée et une case inconnue sans mine en case déminée.

Q9 Ecrivez une fonction `int jouer(int c, int* grille, int N, int M)` qui prend en argument la grille et execute l'action du joueur à la case correspondant au numéro c. Si la case est occupée par une mine, la fonction rend 0. Sinon, elle modifie la case c ainsi que les cases adjacentes, en utilisant la fonction `transformer` et rend 1 comme valeur de retour.

Attention : Attention aux cases sur le bords, qui n'ont pas de voisin dans toutes les directions. Souvenez vous de l'utilité de `%M` et `/M` pour calculer lignes et colonnes.

Q10 Testez les fonctions précédentes dans une fonction `test_jouer()`. **À l'intérieur de cette fonction**, exécutez toutes les actions suivantes : Appelez `test_miner()` pour qu'elle vous donne comme valeur de retour la grille avec mines. Faites ensuite des placements aux cases 7 et 9. Affichez le résultat après chaque placement et contrôlez que la valeur de retour soit bonne.

Q11 Complétez votre `main` pour réaliser la totalité du jeu. Créez d'abord une grille et placez des mines de façon aléatoire. Ensuite, utilisez une boucle dans laquelle on

— demande au joueur humain d'entrer un numéro,
— appelle `jouer` pour effectuer le coup du joueur.
— affiche la grille résultante.

La boucle doit terminer quand le joueur a gagné ou perdu.

Rappel : Suivre les indications dans la section "format de sortie".

Q12 Modifiez votre programme pour qu'après chaque coup, l'affichage indique le nombre de mines autours du dernier coup joué. Précédemment, les cases des environs qui n'ont pas de mine étaient affichées avec ".". Maintenant, afficher à la place du "." le nombre de mines dans les environs de cette case (l'environ étant la case même et les 4 cases adjacentes). Par exemple, le joueur place sur la case 7 (sans mine). Il y a une mine sur la case 6 et la case 2. On affichera "*" sur la case 6 (dans les environs de la case 6 il y a 2 mines), "1" sur la case 3 (dans les environs de 3 il y une seule mine), "0" sur les cases 8 et 11 (dans les environs il n'y a pas de mine).

Voici un exemple d'un jeu :

```
./demin
??!!
?!??
????
0␣trouvées,␣3␣restantes
numéro␣:␣9
??!!
?*??
010?
1␣trouvées,␣2␣restantes
numéro␣:␣6
??*!
?*21
010?
2␣trouvées,␣1␣restantes
numéro␣:␣7
??**
?*21
0100
3␣trouvées,␣0␣restantes
Gagné␣en␣3␣tours.
```

—— Fin du sujet ——