

Algorithmique et programmation en C

Exercice de seconde chance, 2022–2023

Licence 1, Université Paris-Saclay / Évry—Val d’Essonne

L’objectif de cet exercice est de programmer en C une version du jeu sudoku. Le but de ce jeu est de remplir une grille carrée de 9 cases de côté avec des chiffres de 1 à 9 en respectant les contraintes suivantes :

- chaque ligne doit contenir exactement une fois chaque chiffre ;
- chaque colonne doit contenir exactement une fois chaque chiffre ;
- chaque sous-grille de 3×3 doit contenir exactement une fois chaque chiffre.

Lorsqu’une de ces contraintes n’est pas respectée, on dit qu’il y a un conflit entre les chiffres qui se répètent. Les sous-grilles sont représentées ci-contre avec une ligne plus grasse. La grille représentée n’est pas complète mais elle vérifie les contraintes ci-dessus.

				2		9		
5								
		6						
						8		
4	1		3					
								7

Un sudoku avec ncurses

Dans notre version du sudoku, nous utiliserons la bibliothèque `ncurses` pour faire le rendu graphique de l’interface du jeu et pour la saisie des touches, comme dans le jeu 2048 réalisé en TD (reportez-vous à l’énoncé de TD pour les informations générales sur cette bibliothèque). Dans notre version nous n’aurons pas la possibilité de faire des lignes grasses. On obtiendra un rendu comme dans les trois grilles ci-dessous :

sudoku

=	=							
				2		9		
	5							
		6						
						8		
4		1		3				
								7

sudoku

				2		9		
	:5:							
		6						
						8		
4		1		3				
								7

sudoku

	#2#			*2*		9		
	5							
		6						
						8		
4		1		3				
								7

On note que les lignes grasses ont été remplacée par des lignes qui se prolongent légèrement hors de la grille. On note aussi qu’en plus des chiffres, on a un curseur qui indique la position où on peut jouer. Dans l’image de gauche le curseur est `= =` dans la case en haut à gauche, les signes “`=`” indiquent qu’on peut jouer dans cette case. On déplacera le curseur avec les flèches de direction du clavier. Dans l’image centrale, le curseur est `:5:`, les signes “`:`” indiquent qu’on ne peut pas jouer cette case car c’est l’une de celles qui étaient préremplies au démarrage du jeu pour constituer le puzzle. Dans l’image de droite, le curseur est `#2#`, on a ajouté un 2 dans cette case et il entre en conflit avec l’autre 2 plus à droite sur la même ligne, et qui lui est indiqué par `*2*`. Les signes “`*`” indiquent un conflit ailleurs qu’à la position du curseur, et les signes “`#`” indiquent un conflit à la position du curseur.

Le déroulement du jeu est le suivant :

- au démarrage, la grille contient chaque chiffre de 1 à 9 dans des positions aléatoires mais sans conflit ;
- on peut déplacer le curseur avec les touches de direction (il ne peut pas sortir de la grille) ;
- si le curseur se trouve dans une case préremplie au démarrage, on ne peut pas modifier cette case ;
- sinon, on peut réaliser une des actions suivantes sur la case du curseur :

- ajouter ou remplacer un chiffre avec une touche `1` à `9`,
- vider la case avec la touche `x` ou `0`,
- modifier la case avec la barre d'espace comme expliqué ci-dessous,
- si la case est vide, la résoudre avec la touche `h`, c'est-à-dire y ajouter automatiquement un chiffre qui permet de résoudre la grille complète, si elle a une solution ;
- enfin, à tout moment on peut :
 - résoudre complètement le jeu avec la touche `s` en complétant automatiquement toutes les cases, si la grille a une solution ;
 - remettre la grille dans son état initial avec la touche `r` ;
 - quitter le jeu avec la touche `←` (*backspace*) ;
- si on ajoute le dernier chiffre qui produit une grille gagnante, le jeu annonce la victoire et s'arrête.

Sur les ordinateurs équipés de clavier compacts sans pavé numérique, on doit taper `↑+&` au lieu de `1`, `↑+é` au lieu de `2`, etc., sur un clavier français. Mais, lorsqu'on lit une touche avec la fonction `getch()` de `ncurses`, toutes les touches du clavier sont récupérables individuellement, y compris les modificateurs comme `↑`, `alt`, ou `ctrl`. En conséquence, si on tape `↑+&`, `getch()` récupère séparément le code de la touche `↑` et celui de `&`, au lieu de récupérer le code de `1` comme on le souhaiterait. Pour pallier ce problème, notre jeu utilisera la barre d'espace pour changer le contenu d'une case : si elle contient un nombre $n < 9$, elle sera remplacée par $n + 1$, si elle contient 9 elle sera vidée, et si elle est vide elle sera remplacée par 1. Cela permet de "cycler" facilement entre les valeurs possibles.

Structure du programme

Pour démarrer, on se donne le code source suivant, décrit ci-après :

```

1  //ldd: -lncurses
2  #include <string.h>
3  #include <ncurses.h>
4  #include <time.h>
5  #include <stdlib.h>
6
7  void INIT () {
8      initscr();
9      raw();
10     keypad(stdscr, TRUE);
11     noecho();
12     srand(time(NULL));
13 }
14
15 void DONE() {
16     endwin();
17     exit(0);
18 }
19
20 typedef int sudoku[9][9];
21
22 sudoku solution = {
23     {5, 3, 4, 6, 7, 8, 9, 1, 2},
24     {6, 7, 2, 1, 9, 5, 3, 4, 8},
25     {1, 9, 8, 3, 4, 2, 5, 6, 7},
26     {8, 5, 9, 7, 6, 1, 4, 2, 3},
27     {4, 2, 6, 8, 5, 3, 7, 9, 1},
28     {7, 1, 3, 9, 2, 4, 8, 5, 6},
29     {9, 6, 1, 5, 3, 7, 2, 8, 4},
30     {2, 8, 7, 4, 1, 9, 6, 3, 5},
31     {3, 4, 5, 2, 8, 6, 1, 7, 9}
32 };
33
34 int cursor_row = 0;
35 int cursor_col = 0;
36 int init_pos[9][2];
37
38 sudoku grid, solu, init, good;
39
40 char* message = NULL;
41
42 void copy (sudoku src, sudoku tgt);
43 int is_valid (sudoku g, int row, int col, int val);
44 int validate (sudoku src, sudoku tgt);
45 int win (sudoku g);
46 int solve (sudoku src, sudoku tgt);
47 void init_grid ();
48 int cursor_at_init ();
49 void display ();
50 int play();
51
52 int main () {
53     INIT();
54     init_grid();
55     do {
56         display();
57         if (win(grid)) {
58             message = "YOU WIN :)";
59             display();
60             getch();
61             break;
62         }
63     } while (play());
64     DONE();
65 }
```

- la ligne 1 est pour CoW afin de compiler avec la bibliothèque `ncurses` ;
- les fonctions `INIT()` et `DONE()` servent à initialiser et terminer le mode `ncurses`, comme pour 2048 ;
- la ligne 20 définit un type pour représenter une grille de sudoku ;
- le ligne 22 définit une grille de sudoku complète, elle vous permettra de tester que vous savez détecter une grille gagnante ;

- les lignes 34 et 35 définissent la position du curseur, au début il est placé en haut à droite de la grille ;
- la ligne 36 définit une variable pour stocker la position des 9 chiffres insérés initialement dans la grille, ceux qu'on ne peut pas modifier ;
- la ligne 38 définit les variables pour stocker quatre grilles : `grid` est la grille de jeu qu'on remplit, `solu` est une version résolue de `grid` (s'il y a une solution), `init` est la grille créée à l'initialisation du jeu, enfin `good` est une grille spéciale qui sert à mémoriser si chaque valeur de `grid` est en conflit ou non avec une autre ;
- la ligne 40 définit un texte qui pourra être affiché dans la barre horizontale sous la grille, on y reviendra ;
- les lignes 42 à 50 donnent les prototypes des fonctions que vous devez réaliser, et qui sont détaillées plus bas ;
- enfin, à partir de la ligne 52, on a la fonction `main()` qui exécute le jeu.

Dans la suite, on décrit les différentes fonctions à réaliser. Vous devez vous conformer à cette structure autant que possible : elle vous facilitera la réalisation du jeu et le barème sera calé dessus.

```
void copy(sudoku src, sudoku tgt);
```

Cette fonction doit copier une grille `src` sur la grille `tgt`.

```
int is_valid (sudoku g, int row, int col, int val);
```

Cette fonction vérifie si, en plaçant `val` à la ligne `row` et à la colonne `col` dans la grille `g`, on obtient un coup valide, c'est-à-dire ne générant pas de conflit. Il s'agit de vérifier les trois conditions données au début de cet énoncé. Si `val` vaut 0, la fonction utilise la valeur déjà présente dans `g` à la position indiquée. Notez qu'une case vide est toujours valide puisqu'elle ne peut pas engendrer de conflit. `is_valide` renvoie 1 si le coup est valide, 0 sinon.

```
int validate (sudoku src, sudoku tgt);
```

Cette fonction valide toutes les cases de la grille `src` et stocke le résultat dans la grille `tgt` aux positions correspondantes, cette dernière n'est donc pas vraiment une grille de sudoku et ne contiendra que des 0 et des 1. `validate` renvoie 1 si toutes les cases de `src` sont valides, et 0 sinon.

```
int win (sudoku g);
```

Cette fonction vérifie si la grille `g` est gagnante, c'est-à-dire qu'elle est entièrement remplie et que toutes ses cases sont valides ; elle renvoie 1 si la grille est gagnante et 0 sinon.

```
int solve (sudoku src, sudoku tgt);
```

Cette fonction essaie de résoudre la grille `src` en stockant le résultat dans `tgt`. Plus précisément, la grille `src` est copiée dans `tgt` puis elle est complétée jusqu'à la résoudre complètement, si c'est possible. `solve` renvoie 1 si elle a réussi à résoudre `src` (et alors `tgt` doit contenir une version complétée de `src`), et 0 sinon.

```
void init_grid ();
```












Cette fonction initialise la grille `grid` : elle est d'abord vidée puis les chiffres de 1 à 9 sont placés dedans à des positions aléatoires qui doivent former une grille qui a une solution. En effet, c'est cette grille qui est proposée au début du jeu, il faut donc qu'elle ait une solution. Notez que, même si la grille proposée au départ a toujours une solution, il est possible d'ajouter dedans des chiffres qui la rendent impossible à résoudre, c'est d'ailleurs là que réside la difficulté du jeu.

```
int cursor_at_init ();
```

Cette fonction renvoie 1 si le curseur se trouve à une position remplie initialement par `init_grid` et 0 sinon. Cela permet d'ajuster l'affichage du curseur (par exemple :5: au lieu de =5= dans la seconde capture d'écran ci-dessus) et le comportement du programme (en interdisant de changer la valeur).

```
void display ();
```



Cette fonction affiche la grille ainsi que les barres au dessus et au dessous. Pour afficher les lignes et croisements de la grille, `ncurses` fournit des caractères spéciaux : l'appel de fonction `addch('x')` affiche un caractère à la position courante, comme si on avait appelé `printw("x")` (mais `printw` affiche une chaîne). En revanche, les caractères composant une grille ne sont pas des `char`, mais des codes spéciaux, qu'on ne peut afficher qu'avec `addch`, vous ne pouvez pas les insérer, par exemple, avec `printw("%c", ACS_PLUS)`, il faut appeler `addch(ACS_PLUS)` à la place. Les codes qui seront utiles pour afficher la grille sont :



– ACS_ULCORNER		coin supérieur gauche ;	– ACS_PLUS		croisement de deux lignes ;
– ACS_URCORNER		coin supérieur droit ;	– ACS_TTEE		T en haut de grille ;
– ACS_LLCORNER		coin inférieur gauche ;	– ACS_BTEE		T en bas de grille ;
– ACS_LRCORNER		coin inférieur droit ;	– ACS_LTEE		T à gauche de la grille ;
– ACS_HLINE		ligne horizontale ;	– ACS_RTEE		T à droite de la grille.
– ACS_VLINE		ligne verticale ;			

Avec tous ces caractères, on peut dessiner la grille comme sur les captures d'écran ci-dessus, y compris les prolongements toutes les trois lignes/colonnes pour marquer les sous-grilles : il suffit d'utiliser des ACS_PLUS au lieu des T. Notez enfin que chaque case contient trois caractères : (1) un espace, ou le curseur, ou le signe de conflit, (2) le chiffre ou un espace, et (3) à nouveau un espace/curseur/conflit.



Dans la barre horizontale sous la grille, `display` affiche le texte contenu dans la variable globale `message` : si elle vaut `NULL`, une simple ligne contenant des signes = sera affichée, sinon, on insèrera dans cette ligne le texte du message encadré de deux espaces, comme le texte `sudoku` dans la barre du haut. Voyez par exemple dans le `main` fourni, à la ligne 58, comment on utilise `message = "YOU WIN :)"` pour que le `display` qui suit l'affiche. Une fois le message affiché, la variable `message` est remise à `NULL` ce qui fait que le message ne dure qu'entre deux coups de jeux. Prenez soin de n'affecter `message` qu'avec des littéraux chaînes (des chaînes données directement entre guillemets) et non avec des chaînes calculées dynamiquement, sinon il vous faudra gérer la mémoire.

```
int play();
```






Cette fonction fait un pas de jeu, en lisant et traitant une touche, et renvoie 1 s'il faut continuer ou 0 si on a tapé  et qu'il faut arrêter le jeu. Si la touche tapée n'est pas une de celle reconnue, la fonction `play` en lit une autre, et cela jusqu'à ce qu'elle ait une touche valide à traiter. La fin du jeu par victoire est détectée dans `main` puisqu'elle n'est diagnostiquée qu'après un pas de jeu (l'ajout du dernier chiffre ou la résolution avec ).


En plus de faire un pas de jeu, `play` peut renseigner `message` pour indiquer qu'une action n'est pas possible, par exemple si on tape  ou  et que la grille n'a pas de solution.

Simplifications

Si vous n'arrivez pas à programme `solve`, vous pouvez réaliser une version simplifiée du jeu où `init_grid` se content de vérifier qu'il n'y a pas de conflits dans la position initiale, et où la touche  n'est pas active. Et alors vous pouvez remplacer le traitement de la touche  par la recopie de la grille `solution` : cela ne résout pas le puzzle posé au départ, mais ça permet de tester votre fonction `win`.

Vous pouvez ignorer `message` et son affichage pour simplifier `display`. De même, vous pouvez afficher une grille plus simple en n'utilisant pas les caractères spéciaux et `addch`, mais uniquement des caractères disponibles au clavier (+ - |) comme pour 2048.

Si vous avez un clavier avec un pavé numérique, vous pouvez supprimer le traitement de la barre d'espace, si vous n'avez pas de pavé numérique, vous pouvez remplacer les touches  à  par la rangée de lettres  à  qui se trouve juste au dessous des chiffres normalement accessibles en utilisant .

Vous pouvez ignorer la grille `init` et ne pas traiter la touche .

Vous pouvez abandonner la contrainte de non répétition des chiffres dans les sous-grilles qui est un peu plus difficile à programmer que les deux autres.

Toutes ces simplifications, et d'autres que vous pourrez vouloir introduire vous feront potentiellement perdre des points car vous simplifiez le problème posé. Mais vous pouvez aussi les intégrer dans un premier temps puis rajouter progressivement les fonctionnalités supprimées une fois que vous avez un premier programme fonctionnel. Dans tous les cas, il vaut mieux rendre un programme simplifié que vous maîtrisez et serez capable d'expliquer à l'oral plutôt qu'un programme plus complet qui contient des choses que vous ne comprenez pas bien.

Bien sûr, vous pouvez aussi proposer des améliorations ou des ajouts, mais veillez à rester dans le cadre proposé qui constitue la structure du barème.

Rendu et évaluation

Votre programme devra être rendu sur `badass` et vous aurez un entretien individuel comme ceux déjà passés en TD, avec votre enseignant/e habituel/le. Les dates vous seront communiquées par un message envoyé par le forum d'annonces sur eCampus.