Algorithmique et programmation en C

Franck Pommereau

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

Consignes

Les exercices signalés par 🏠 sont à préparer individuellement à l'avance et à remettre en ligne sur `https://badass.ibisc.univ-evry.fr` avant la séance (sauf bien sûr pour la première). Votre chargé·e de TD vous donnera l'échéance exacte pour remettre vos travaux chaque semaine. Les exercices signalés par ✹ sont à préparer et remettre de la même façon, mais seront notés dans le cadre du contrôle continu intégral. *Un logiciel de détection de plagiats sera utilisé pour vérifier que vous rendez des solutions individuelles.* Les exercices signalés par 🏛 sont à réaliser pendant la séance, ils peuvent avoir été préparés mais n'ont pas à être remis en ligne. La difficulté des exercices est indiquée par des étoiles :

★☆☆ Exercice facile qui devrait être résolu directement.

★★☆ Exercice qui nécessite une réflexion préalable.

★★★ Exercice long ou qui demande une réflexion approfondie.

Enfin, du texte signalé par ☑ donne les objectifs d'un exercice, et du texte signalé par 🧭 donne des indications pour le résoudre.

Séance 1 : prise en main

Cette séance permet de se familiariser avec les outils `CoW` et `badass`, les exercices sont notés 🏛 car ils sont réalisés en séance. Mais vous devez les soumettre en ligne afin de vérifier le fonctionnement de votre compte `badass` et votre compréhension de ses diagnostics.

🏛 Exercice 1 (★☆☆)

☑ Prendre en main des outils et de l'environnement de travail.

🧭 La plateforme de soumission en ligne est `badass` : `https://badass.ibisc.univ-evry.fr`
Vous devrez y créer un compte, puis vous y connecter avec le mot de passe fourni lors de l'inscription.

🧭 La plateforme de programmation est `CoW` : `https://cow.ibisc.univ-evry.fr`
Prenez soin de télécharger vos travaux car rien n'y est sauvegardé.

🧭 Programmez chacune des questions dans `CoW` puis soumettez la sur `badass`.

1. Reprenez le programme `hello.c` vu en cours, compilez-le, et exécutez-le.
2. Que se passe-t-il si vous supprimez la ligne commençant par `#include` ? Pourquoi ?
3. Reprenez le programme `helloname.c` vu en cours, compilez-le, et exécutez-le.
4. Supprimez l'instruction `free`. Pourquoi le programme fonctionne-t-il toujours ? Comment expliquez-vous le diagnostique fourni par `badass`

🏛 Exercice 2 (★☆☆)

☑ Modifier un programme, enchaîner plusieurs instructions.

Reprenez le programme `hello.c` et modifiez-le pour qu'il affiche plusieurs messages.

🏛 Exercice 3 (★☆☆)

☑ Déclarer des variables, les utiliser.

🧭 Inspirez-vous de `helloname.c`

Écrivez un programme qui lit au clavier un prénom, un nom, et un âge puis affiche le message "`Bonjour PRÉNOM NOM, vous avez ÂGE ans.`" où les mots en majuscules sont remplacés par les valeurs lues au clavier.

🏛 Exercice 4 (★☆☆)

☑ Comprendre la notion de types et les conversions automatiques.

🧭 Dans `printf`, utilisez la directive `%u` pour afficher des entiers non signés, et `%i` ou `%d` pour des entiers signés.

Écrivez un programme qui lit un `int` au clavier puis le copie dans une variable de type `unsigned int`. Cette affectation est-elle légale ? Le compilateur proteste-t-il ? Que se passe-t-il si l'entier lu au clavier est négatif ? Affichez la valeur du `unsigned int` après son affectation. Que pouvez-vous dire de la valeur affichée ?

Séance 2 : expressions, fonctions, calculs

Exercice 5 (★☆☆)

Utiliser des expressions. Observer le comportement du programme sur une erreur.

Écrivez un programme lisant deux entiers a et b au clavier puis affichant les valeurs de $a+b$, $a-b$, $a*b$, et a/b. Que se passe-t-il si vous saisissez $b=0$?

Exercice 6 (★☆☆)

Faire des calculs et représenter les résultats.

Écrivez un programme lisant deux entiers t et p au clavier représentant respectivement un nombre total t d'étudiants inscrits, et un nombre p d'étudiants présents. Faites afficher le pourcentage d'étudiants présents par votre programme avec au moins deux décimales de précision.

Exercice 7 (★☆☆)

Définir et appeler une fonction. Utiliser une conditionnelle.

Écrivez une fonction `min` prenant deux entiers en paramètres et renvoyant le plus petit des deux. Ajoutez un programme principal qui lit deux entiers au clavier et affiche le plus petit des deux en faisant appel à cette fonction.

Exercice 8 (★☆☆)

Définir et appeler une fonction, faire des calculs.

Pour $n \geq 0$, la somme $0+\cdots+n$ vaut $n \times (n+1)/2$. Écrivez une fonction `sum` prenant en paramètre un tel entier n et renvoyant cette somme calculée avec cette formule.

Exercice 9 (★☆☆)

Comprendre la priorité des opérateurs.

Écrivez une programme qui montre que les expressions C "`a + b / c`" et "`(a + b) / c`" n'effectuent pas le même calcul.

Exercice 10 (★☆☆)

Approfondir les conditionnelles.

Écrivez une fonction `cmp` qui prend deux paramètres entiers a et b et renvoie 0 si $a=b$, -1 si $a<b$, et 1 si $a>b$.

Exercice 11 (★★☆)

Découvrir les opérateurs *bitwise* de C.

Essayez sur quelques valeurs pour en déduire l'opération. Recherchez ce que font les opérateurs proposés.

Donnez un équivalent des opérations suivantes à l'aides d'opérations arithmétiques sur les entiers :

- `a << 1`
- `a << 2`
- `a & 1`
- `a >> 1`
- `a >> 2`
- `a & 3`

Séance 3 : récursivité et boucles simples

Exercice 12 (★★☆)

Définir et appeler une fonction récursive. Comprendre l'enchaînement des appels récursifs. Traduire en C une définition mathématique.

Si vous ne comprenez pas le déroulement du programme pour la question 4, tentez d'ajouter un `printf` au début de votre fonction pour afficher chaque appel.

Vous ne pouvez pas répondre aux questions sur badass mais vous y soumettrez votre programme avec la fonction `fibo` et le `main` qui l'appelle.

Les nombres de Fibonacci forment une suite $(F_n)_{n\geq 0}$ telle que : $F_0=0$, $F_1=1$, et $F_n=F_{n-1}+F_{n-2}$.

1. Écrivez une fonction récursive `fibo` pour calculer le $n^{\text{ème}}$ nombre de Fibonacci.
2. Lancez cette fonction pour de petites, puis de grandes valeurs de n.
3. Que constatez-vous ?
4. Expliquez ce constat en listant tous les appels à votre fonction.

Exercice 13 (★★☆)

Comprendre le lien entre l'expression mathématique d'un calcul et sa réalisation en C. Utiliser une boucle.

La fonction demandée a été vue en cours, essayez de la reconstruire et non de la recopier.

On a vu en cours une implantation récursive de la fonction factorielle, basée sur la définition : $\mathsf{fact}(n) = 1$ si $n = 0$, $\mathsf{fact}(n) = n \times \mathsf{fact}(n-1)$ sinon. Cependant :

— on peut remplacer la partie récursive de cette définition par $\mathsf{fact}(n) = \prod_{1 \leq i \leq n} i$;

— on sait que la multiplication est associative et donc : $1 \times 2 \times 3 \times \cdots = ((1 \times 2) \times 3) \times \cdots$, c'est-à-dire qu'on peut réaliser les multiplications une à une.

Exploitez ces deux propriétés pour écrire une version itérative du calcul de la factorielle.

Exercice 14 (★☆☆)

Utiliser une boucle `while` et une conditionnelle.

Ajoutez "`#include <stdlib.h>`" et "`#include <time.h>`" au début du programme, puis ajoutez "`srand((unsigned int)time(NULL));`" au début du `main`. Cela permet d'utiliser "`rand() % MAX`" pour obtenir un nombre aléatoire entre `0` et `MAX-1`.

Écrivez un programme choisissant un nombre au hasard dans l'intervalle $[0, 99]$ et le faisant deviner à l'utilisateur : à chaque tentative, le programme indique "trop grand", "trop petit", ou "bravo" (et alors le programme s'arrête).

Exercice 15 (★☆☆)

Utiliser une boucle `for`.

Inspirez-vous de la factorielle.

Reprenez l'exercice 8 et ajoutez une fonction `seum` qui calcule la somme des entiers naturels jusqu'à $n \geq 0$ en utilisant une boucle. Vérifiez grâce à cette fonction que la formule utilisée à l'exercice 8 est juste pour $0 \leq n \leq 100$.

Séance 4 : plus de boucles

Exercice 16 (★★★)

Transformer une récursivité en boucle.

Comme pour l'exercice 12, vous ne répondrez pas aux questions sur badass mais vous y soumettrez votre programme.

On a vu à l'exercice 12 une façon simple mais peu efficace d'implanter la suite de Fibonacci. On se propose de l'améliorer.

1. Sur une feuille de papier, calculez dans l'ordre tous les six premiers termes de la suite de Fibonacci : F_0, F_1, F_2, F_3, F_4, F_5, et enfin F_6.
2. Quels sont les valeurs que vous avez utilisé pour calculer F_6 ?
3. Exploitez ce constat pour proposer une implantation itérative de cette suite.
4. Testez-la pour de grandes valeurs de n. Faites-vous le même constat que pour la version récursive ? Pourquoi ?

Remarque : ceci n'est pas une condamnation de la récursivité, ce n'est tout simplement pas le même algorithme qui est utilisé dans les deux cas. D'ailleurs, le meilleur algorithme pour calculer la suite de Fibonacci est récursif et plus efficace que celui qu'on vient d'implanter.

Exercice 17 (★★☆)

Manipuler différentes sortes de boucles.

On rappelle qu'un nombre premier a exactement deux diviseurs : 1 et lui-même. (Et ainsi, 1 n'est pas premier car n'a qu'un seul diviseur.)

Attention à l'énoncé : les *100 premiers nombres premiers* ne sont pas les *nombres premiers inférieurs à 100*.

Écrivez une fonction `prime` qui teste si un nombre est premier en lui cherchant un diviseur. Utilisez cette fonction pour afficher dans l'ordre les 100 premiers nombres premiers.

Exercice 18 (★★★)

Transformer une récursivité en boucle.

La fonction demandée a été vue en cours, essayez de la reconstruire en vous inspirant des deux exercices précédents. La recopier ne vous servira à rien.

Proposez une implantation itérative du calcul de PGCD avec l'algorithme d'Euclide présenté en cours. On rappelle que $pgcd(0, 0) = 0$.

Exercice 19 (★★☆)

- Manipuler des boucles `for` et leurs indices.
- Essayez de construire votre réponse à chaque question en adaptant celles aux précédentes.
- Pour pimenter l'exercice, essayez de limiter les variables locales au minimum.
- Pour un meilleur affichage, dessinez chaque étoile suivie d'un espace.

1. Écrivez une fonction prenant un paramètre un entier naturel n et dessinant un carré d'étoiles de côté n.
2. Même chose en supprimant la première diagonale.
3. Même chose en supprimant la seconde diagonale.
4. Même chose en supprimant la moitié supérieure droite.
5. Même chose en supprimant la moitié supérieure gauche.

Exemples, pour $n = 4$, de gauche à droite :

```
* * * *          * * *          * * *          *                    *
* * * *        *   * *        * *   *        * *                  * *
* * * *        * *   *        *   * *        * * *              * * *
* * * *        * * *            * * *        * * * *          * * * *
```

Séance 5 : chaînes de caractères

Exercice 20 (★★☆)

- Manipuler les tableaux particuliers que sont les chaînes de caractères.
- On rappelle qu'une chaîne de caractères en C de type `char*` et qu'un caractère nul (`'\0'`) en marque la fin.
- Vous ne devez pas utiliser `#include <string.h>` ou une bibliothèque équivalente mais programmer vous même les fonctions demandées.

Écrivez les fonctions suivantes.

1. Une fonction qui prend une chaîne en paramètre et renvoie sa longueur. (Nombre de caractères utiles, c'est-à-dire sans compte le `'\0'` final.)
2. Une fonction qui prend une chaîne en paramètre et renvoie son miroir.
3. Une fonction qui vérifie si une chaîne est une palindrome (c'est-à-dire qu'il est son propre miroir).
4. Une fonction qui vérifie si deux chaînes sont miroir l'une de l'autre.

Exercice 21 (★★☆)

- Comprendre la correspondance tableaux/pointeurs.
- Les programmes vont se ressembler d'un point de vue algorithmique, mais ils prendront une forme assez différente.
- L'idée de l'exercice est de ne pas utiliser la notation `s[i]`.

Mêmes questions que dans l'exercice 20 mais en n'utilisant que des accès par pointeurs à la place des accès de type tableau.

Séance 6 : encore des chaînes

Exercice 22 (★★☆)

- Manipuler des chaînes de caractères de longueurs diverses.
- L'exercice se base sur la représentation d'entiers naturels en binaire dans des chaînes de caractères. Pour simplifier les opérations, on considère que le bit de poids faible est positionné à l'indice 0. En revanche, cela complique l'affichage puisqu'on a l'habitude de représenter les nombres avec le chiffre de poids faible à droite.

1. Écrivez une fonction faisant l'addition en binaire de deux chaînes de caractères `'0'` ou `'1'` représentant des entiers naturels.
2. Écrivez une fonction qui affiche un nombre binaire ainsi codé, mais en présentant le bit de poids faible à gauche comme c'est l'habitude.
3. Écrivez une fonction calculant la valeur d'un entier ainsi représenté en binaire.
4. Écrivez une fonction calculant la représentation binaire d'un entier naturel.

Exercice 23 (★★☆)

Approfondir la manipulation de chaînes de longueurs diverses

Comme pour l'exercice 20, n'utilisez pas de bibliothèque externe mais programmez les fonctions vous même. En revanche, vous pouvez réutiliser des fonctions de l'exercice 20, et même celles d'autres questions de cet exercice.

Essayez de choisir entre les notations tableaux ou pointeurs pour obtenir le code le plus direct et simple.

Écrivez les fonctions suivantes.

1. Une fonction prenant deux chaînes en paramètres et renvoyant leur concaténation.
2. Une fonction prenant une chaîne en paramètre et en renvoyant une copie dans laquelle les majuscules ont été changées en minuscules et réciproquement.
3. Une fonction comptant combien de fois un caractère apparaît dans une chaîne, les deux étant passés en paramètres.
4. Une fonction prenant deux chaînes `str` et `del` en paramètres et renvoyant une copie de `str` dans laquelle tous les caractères présents dans `del` ont été supprimés. Par exemple si `str="hello world!"` et `del="lol"`, la fonction doit renvoyer `"he wrd!"`.
5. Une fonction prenant deux chaînes en paramètres et renvoyant `1` si la seconde apparaît dans le première et `0` sinon. Par exemple, `"llo"` apparaît dans `"hello world!"`, mais `"ol"` n'y apparaît pas (bien qu'on trouve un `'o'` puis un `'l'` un peu plus loin, mais il y a un `'r'` entre les deux).

Exercice 24 (★★☆)

Manipuler des tableaux de nature diverses.

Utilisez une tableau pour compter le nombre d'apparitions de chaque caractère dans une chaîne.

On rappelle que le type `char` correspond à des entiers sur 1 octet, c'est à dire entre 0 et 255.

Écrivez une fonction qui teste si deux chaînes sont des anagrammes l'une de l'autre.

Séance 7 : tableaux

Dans la suite, on utilisera le type `Tab` défini en cours :

dicho.c

```
3  typedef struct {
4    unsigned int len;
5    int* val;
6  } Tab;
```

Exercice 25 (★★☆)

Utiliser des tableaux. Imbriquer des boucles.

L'algorithme du crible d'Érathosthène permet de déterminer les nombres premiers inférieurs ou égaux à un entier N. Il est assez simple : on écrit dans un tableau (le crible) tous les entiers entre 2 et N. On répète ensuite les opérations suivantes :

— prendre le premier entier du crible, il est premier ;

— l'effacer ainsi que ses multiples.

On s'arrête quand le crible est vide, on a alors récupéré tous les nombres premiers qu'il contenait.

1. Proposez un type pour représenter un crible de taille quelconque en permettant facilement l'effacement d'un entier.
2. Écrivez une fonction `sieve` :
   — prenant en paramètre un entier N,
   — affichant tous les nombres premiers inférieurs ou égaux à N au moyen de l'algorithme du crible d'Érathosthène,
   — et renvoyant le nombre d'entiers premiers trouvés.
3. Que pensez-vous de son efficacité par rapport à une boucle exploitant la fonction de l'exercice 17 ? Pourrait-on utiliser le crible pour afficher les n premiers nombres premiers ?

Exercice 26 (★★★)

Parcourir, modifier, copier, et manipuler des tableaux.

1. Écrivez une fonction qui renverse l'ordre des éléments d'un tableau de type `Tab` (en place, c'est-à-dire sans faire une copie du tableau contrairement au miroir d'une chaîne vu plus tôt).
2. Écrivez une fonction qui renvoie la plus petite valeur d'un tableau.
3. Écrivez une fonction qui renvoie la deuxième plus petite valeur d'un tableau, en ne faisant qu'un parcours de ce tableau.
4. Écrivez une fonction qui teste si aucun élément d'un tableau n'y est dupliqué. (Autrement dit, la fonction renvoie `1` si chaque élément du tableau n'y est présent qu'une seule fois, `0` sinon.)
5. Écrivez une fonction qui prend un tableau en argument et renvoie un nouveau tableau tel que :
   — toute valeur dans le tableau de départ est présente dans le tableau résultat ;
   — le tableau résultat ne contient que des valeurs du tableau de départ ;
   — le tableau résultat ne contient aucun élément dupliqué.

Exercice 27 (★★☆)

Ceci est le premier exercice d'une série qui vous conduira à programmer votre version du jeu 2048.

Le jeu 2048 se joue seul contre l'ordinateur, sur un plateau de 4×4 cases. À chaque tour, l'ordinateur fait apparaître le nombre 2 dans une case libre du tableau, si aucune case n'est libre, le jeu est terminé. Si une case est libre, le joueur doit choisir une direction (gauche, haut, droite, bas) pour "compresser" le plateau, par exemple, s'il choisit la gauche :

— tous les nombres du plateau sont décalés vers la gauche de manière à éliminer toutes les cases libres à gauche ;
— si une case contient le même nombre que celle à sa gauche, leurs contenus sont additionnés, le résultat remplace celui de la case de gauche (cette valeur s'ajoutant au score de la joueuse) et celle de droite est vidée.

La figure 1 donne un exemple, avec des couleurs pour identifier quel nombre fusionne avec lequel. Notez comment, à la ligne 3, les 8 sont additionnés à partir de la gauche (donc les deux premiers ensemble), le troisième restant seul. Le but du jeu est d'accumuler le plus de points possibles en jouant le plus longtemps possible.

Pour l'affichage, on utilisera la bibliothèque `ncurses` qui permet d'utiliser le terminal comme une matrice de caractères et d'y faire des dessins à base de texte. Par exemple, notre jeu s'affichera comme représenté sur la figure 1. Sur cette figure, on donne aussi le code source en C permettant d'inclure la bibliothèque `ncurses`, ainsi qu'une fonction `INIT_2048()` pour initialiser le terminal, et une autre `DONE_2048()` pour le rétablir le terminal à son fonctionnement normal et quitter le jeu.

Pour compiler sous `CoW` avec la bibliothèque `ncurses`, prenez soin d'ajouter le commentaire `// ldd: -lncurses` comme dans le code source fourni.

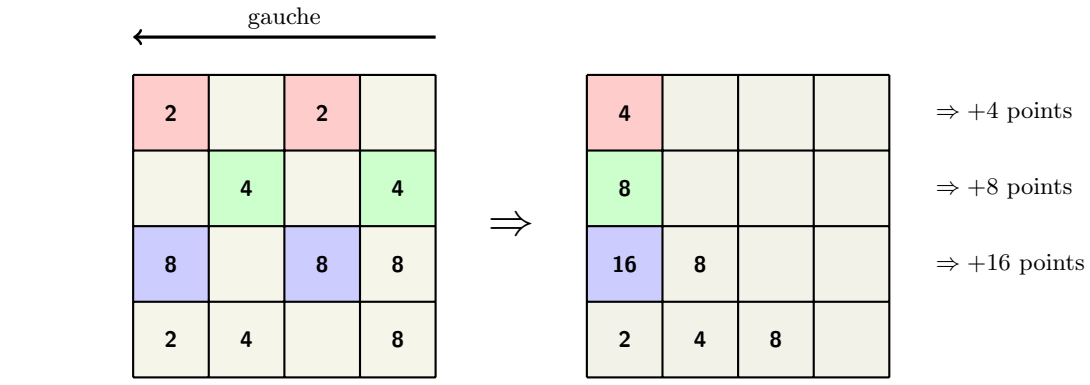
La première fonction se charge aussi d'initialiser le générateur de nombres pseudoaléatoires qu'on utilisera pas la suite pour choisir une case au hasard. On dispose alors des fonctions suivantes :

— `move(0,0)` ⇒ déplace le curseur dans le coin supérieur gauche du terminal ;
— `printw("...", ...)` ⇒ version de `printf` compatible avec `ncurses` ;
— `refresh()` ⇒ affiche sur le terminal les résultat des appels à `printw` ;
— `getch()` ⇒ lit une touche du clavier, y compris les touches de direction, et renvoie son code (on verra plus tard comment reconnaître ces codes) ;
— `random()` ⇒ renvoie un `long int` aléatoire.

Toutes ces fonctions nous serviront pour gérer le jeu.

1. Proposez un type pour représenter le plateau de jeu, et permettant en particulier de représenter les cases vides.
2. Déclarez une variable globale `board` de ce type.
3. Écrivez une fonction `void init_board()` pour remplir le plateau de cases vides.
4. Déclarez une variable globale `score` initialisée à zéro pour stocker le score du joueur.
5. Écrivez une fonction `void display_board()` pour afficher le plateau de jeu, ainsi qu'une bannière avec le nom du jeu et le score de la joueuse.

   `printw("%5i", n)` permet d'afficher la valeur d'un `int n` sur 5 colonnes de texte, alignée à droite.
6. Écrivez une fonction `main` qui initialise le terminal avec `INIT_2048()`, initialise le plateau de jeu avec `init_board()`, l'affiche avec `display_board()`, lit une touche au clavier avec `getch()` (pour laisser le temps de voir le plateau), et quitte le jeu avec `DONE_2048()`.




```
== 2048 ============================
====================== score:     52

+--------+--------+--------+--------+
|        |        |        |        |
|        |        |        |        |
|        |        |        |        |
+--------+--------+--------+--------+
|        |        |        |        |
|        |        |        |        |
|        |        |        |        |
+--------+--------+--------+--------+
|        |        |        |        |
|    2   |        |        |        |
|        |        |        |        |
+--------+--------+--------+--------+
|        |        |        |        |
|   16   |    4   |        |        |
|        |        |        |        |
+--------+--------+--------+--------+
```

```c
// ldd: -lncurses
#include <ncurses.h>
#include <time.h>
#include <stdlib.h>

void INIT_2048 () {
  initscr();
  raw();
  keypad(stdscr, TRUE);
  noecho();
  srand(time(NULL));
}

void DONE_2048() {
  endwin();
  exit(0);
}
```

Figure 1. En haut : un pas du jeu 2048 si le joueur choisit la direction gauche. **En bas à gauche :** l'écran de jeu dessiné avec `ncurses`. **En bas à droite :** le code C à réutiliser dans votre programme.

Séance 8 : tris

Exercice 28 (★★★)

Manipuler des tableaux de façon plus complexe : décalages, copies, recherche par dichotomie, travail sur des tranches.

On a vu en cours le principe du tri par insertion : on sélectionne la première valeur de la partie non triée du tableau (tout le tableau sauf sa première case au départ), et on l'insère à sa place dans la partie triée (une seule case au départ, mais qui grandit d'une case à chaque insertion). On va le réaliser en plusieurs étapes, sur des tableaux de type `Tab`.

1. Reprenez la fonction `slice` vue en cours qui renvoie une "tranche" de tableau, écrivez une fonction qui recopie une tranche sur une autre de même taille.
2. Écrivez une fonction qui recherche par dichotomie la place d'un entier dans un tableau trié.
3. Écrivez finalement le tri par insertion en insérant un à un les éléments non triés d'un tableau. L'insertion utilise la recopie de tranches pour décaler les éléments qui suivent la position d'insertion.
4. Pensez-vous que l'insertion dans un tableau est une opération efficace ? Comparez au tri par sélection vu en cours.

Exercice 29 (★★☆)

Découvrir un tri atypique pour ouvrir de nouveaux horizons.

Inspirez-vous de l'exercice 24.

Le tri compteur (ou tri de comptage) est applicable si on trie des valeurs d'un type qui en comporte peu de valeurs distinctes. Par exemple, les caractères d'une chaîne : chacun est codé sur un seul octet et il n'y en a donc que 256 possibles. On va donc supposer qu'on trie des chaînes de caractères. L'algorithme se déroule en trois étapes :

1. On déclare un tableau indexé par les valeurs possibles pour un `char`, c'est-à-dire les entiers entre 0 et 255. On initialise toutes les cases à zéro.
2. On parcours la chaîne une première fois, et on incrémente la case du tableau de compteurs correspondant à chaque caractère rencontré.

3. Pour obtenir les valeurs dans l'ordre, il suffit de parcourir le tableau de compteurs et, pour chaque cellule d'index `c` et de valeur `n`, on doit placer `n` fois le caractère `c` dans la chaîne finale.

Implantez ce tri.

Exercice 30 (★★☆)

☑ Suite du jeu 2048 : on ajoute la progression et la détection de fin de partie.

1. Écrivez une fonction `int count_empty()` qui renvoie le nombre de cases libres sur le plateau.
2. Écrivez une fonction `void add_two(int empty)` qui prend en paramètre le nombre de cases libres du plateau et insère le nombre 2 sur le plateau sur la n-ième case vide, n étant choisi aléatoirement entre 0 et `empty-1`.
3. Écrivez la fonction `int game_over(int add)` qui compte le nombre de case libres, puis :
   - s'il n'y a pas de case libre, affiche un message comme sur la figure 2, lit une touche au clavier avec `getch()`, et renvoie 1 ;
   - s'il y a au moins une case libre et que `add` est vrai, ajoute un 2 au plateau, l'affiche, et renvoie 0 ;
   - sinon, renvoie 0.
4. Modifiez `main` pour y déclarer une variable `add` initialisée à `1`, et remplacer l'affichage du plateau et la lecture d'une touche par une boucle `while(!game_over(add))` avec juste un `getch()` dans le corps de boucle. Ainsi, le jeu se déroule jusqu'à la fin, mais sans compression du plateau.

```
== 2048 ============================
====================== score:     124

+--------+--------+--------+--------+
|        |        |        |        |
|    2   |    4   |    8   |    4   |
|        |        |        |        |
+--------+--------+--------+--------+
|        |        |        |        |
|    2   |   16   |    4   |    2   |
|        |        |        |        |
+--------+--------+--------+--------+
|        |        |        |        |
|    4   |    8   |    4   |    2   |
|        |        |        |        |
+--------+--------+--------+--------+
|        |        |        |        |
|    4   |    8   |    4   |    2   |
|        |        |        |        |
+--------+--------+--------+--------+

============= GAME OVER =============
=========== (press a key) ===========
```

Figure 2. Fin du jeu 2048.

Séance 9 : encore un tri

Exercice 31 (★☆☆)

☑ Une fonction simple pour se préparer au tri à bulles.

Écrivez une fonction qui teste si un tableau de type `Tab` est déjà trié.

Exercice 32 (★★★)

☑ Découvrir un algorithme de tri classique aux vertus pédagogiques : le tri à bulles n'est jamais utilisé en pratique, mais il est systématiquement enseigné, et même utilisé comme test pour des entretiens d'embauche.

Le principe du tri à bulles est de faire "remonter" dans le tableau les valeurs les plus grandes, comme des bulles. Un parcours traverse le tableau en échangeant les valeurs successives qui n'ont pas le bon ordre. À la suite de ce parcours, la plus grande valeur se trouve à sa place. En recommençant le processus sur la partie non triée du tableau, on va remettre successivement toutes les valeurs à leur place. Il est possible de vérifier au cours d'un passage qu'on ne fait aucun échange et donc que le tableau est déjà trié, ce qui permet d'arrêter le tri. Implantez le tri à bulles sur un tableau de type `Tab`.

Exercice 33 (★★☆)

Suite du jeu 2048 : on ajoute la compression du plateau.

Dans un premier temps, on ne va pas gérer toutes les directions, mais uniquement la gauche. Nous verrons dans un prochain exercice comment gérer les autres.

1. Écrivez une fonction `int shift_board()` qui décale le plateau à gauche, mais sans faire les additions. Cette fonction renvoie vrai si au moins un entier a bougé, faux sinon.
2. Écrivez une fonction `int update_board()` qui :
   - appelle `shift_board()` pour décaler les valeurs vers la gauche ;
   - fait les additions des valeurs consécutives égales (et met à jour le score) : quand deux cases successives de gauche à droite contiennent les mêmes entiers, celle de gauche reçoit la somme des deux entiers, et celle de droite est vidée ;
   - appelle `shift_board()` de nouveau pour compresser les cases vidées par une addition.

   La fonction `update_board()` renvoie vrai si au moins un entier a bougé ou a été additionné, faux sinon.
3. Modifiez `main` pour que le plateau soit compressé après chaque lecture de touche.

Séance 10 : matrices

Exercice 34 (★★☆)

Manipuler des tableaux à plusieurs dimensions

Le type `matrix` doit s'inspirer du type `Tab` pour les tableaux à une dimension.

Le produit de deux matrices a de dimensions $p \times q$ et b de dimensions $q \times r$ est une matrice c de dimensions $p \times r$ définie par $c_{i,j} = \sum_{1 \leq k \leq q} a_{i,k} \times b_{k,j} = a_{i,1} \times b_{1,j} + \cdots + a_{i,q} \times b_{q,j}$ pour tout $1 \leq i \leq p$ et $1 \leq j \leq r$.

1. Définissez un type `matrix` pour représenter une matrice $p \times q$ de nombres flottants.
2. Écrivez une fonction `matrix new_matrix(int p, int q)` qui alloue la mémoire pour une matrice dont les dimensions sont passées en paramètres. Si `p` ou `q` n'est pas strictement positif, la fonction renvoie une matrice de taille 0×0 sans mémoire associée.
3. Écrivez une fonction `int is_matrix(matrix m)` qui renvoie vrai si la matrice `m` a deux dimensions non nulles.
4. Écrivez une fonction `void free_matrix(matrix* m)` qui libère la mémoire associée à une matrice et s'assure que la matrice prend la taille 0×0.
5. Écrivez une fonction `matrix mult(matrix a, matrix b)` qui renvoie le produit de deux matrices `a` et `b`. Si les dimensions de `a` et `b` ne permettent pas le calcul, la fonction renvoie une matrice de taille 0×0.

Exercice 35 (★★★)

Concevoir une programme du début à la fin en adaptant ses objectifs.

Lorsqu'on tape un caractère imprimable (comme un chiffre), la fonction `getch()` renvoie une valeur égale au `char` correspondant.

Écrivez un *tic-tac-toe*, ou jeu du morpion, avec les fonctionnalités suivantes, listées par ordre de priorité, il n'est donc pas obligatoire de toutes les fournir :

- Le joueur commence et joue contre l'ordinateur.
- Le plateau de jeu affiche les numéros de cases de 1 à 9, qui sont remplacés par des `X` ou des `O` à mesure que le jeu progresse. Le joueur choisit une case en tapant son numéro. S'il tape un numéro de case déjà occupée, ça ne compte pas et il peut choisir une autre case. S'il tape une autre touche, le jeu s'arrête par abandon.
- Le résultat de la partie est affiché à la fin du jeu (abandon, nom du vainqueur, ou partie nulle).
- Les cases sont numérotées comme sur le pavé numérique d'un ordinateur (7, 8, 9, puis au dessous 4, 5, 6, et tout en bas 1, 2, 3).
- L'affichage est géré par `ncurses` comme dans notre version de 2048.
- L'ordinateur essaie de gagner, ou au moins de ne pas perdre.

Exercice 36 (★★☆)

Suite et fin du jeu 2048 : on ajoute la gestion des différentes directions. Au lieu de dupliquer la fonction `update_board` en trois autres variantes, on va utiliser des transformations du plateau pour se ramener au cas qu'on a déjà traité.

Parmi les codes de touches renvoyés par `getch()`, vous aurez besoin de `KEY_UP`, `KEY_DOWN`, `KEY_LEFT`, `KEY_RIGHT`, et `KEY_BACKSPACE` qui sont des constantes définies dans `ncurses`.

1. Définissez un type `key` pour représenter une des quatre directions ainsi qu'une cinquième valeur pour coder "fin de partie".
2. Écrivez une fonction `key get_key()` qui lit des touches au clavier avec `getch()` jusqu'à ce qu'une d'elles corresponde à l'une des quatre flèches de direction ou à la touche *backspace*, et dans ce cas, la fonction renvoie la valeur correspondante. Si d'autres touches sont tapées, elles seront lues mais ignorées.
3. Écrivez trois fonctions auxiliaires permettant de transformer le plateau et qui serviront à gérer toutes les directions dans la question suivante :
   - une fonction `void swap(int* a, int* b)` qui échange deux entiers en mémoire ;
   - une fonction `mirror_board()` qui échange le contenu de chaque ligne du plateau avec celle en position miroir ; Ainsi, une ligne `a b c d` devient `d c b a` ;
   - une fonction `void pivot_board()` qui échange chaque case (i, j) du plateau avec la case (j, i).
4. Écrivez une fonction `int play(key dir)` qui fait un pas de jeu en considérant que `dir` vaut toujours une des quatre directions :
   - selon la valeur de `dir`, elle appelle une combinaison de `mirror_board()` et `pivot_board()` pour modifier le plateau et se ramener au cas de compression vers la gauche ;
   - puis, elle appelle `update_board()` pour compresser le plateau vers la gauche ;
   - enfin, elle appelle la combinaison inverse de `mirror_board()` et `pivot_board()` pour remettre le plateau dans son orientation d'origine.

   La fonction `play()` renvoie vrai si au moins un entier a bougé ou a été additionné, faux sinon. On ne compte pas les mouvements dus à `mirror_board()` et `pivot_board()` puisqu'ils sont compensés et resteront invisibles.
5. Intégrez les appels à `get_key` et `play` dans `main` de façon à gérer la progression du jeu (y compris la fin de partie). Utilisez la valeur renvoyée par `play` pour passer le bon argument à `game_over` afin d'éviter d'ajouter un 2 lorsque la joueuse à utilisé une touche qui ne changeait pas le plateau (on considère alors qu'elle n'a pas joué).