Exercice 1 (Manipulation de fichiers) :
`commande > fichier` copie la sortie de la commande `commande` dans le fichier `fichier`. Par exemple, `ls -l > files.txt` va enregistrer dans `files.txt` la liste des fichiers présents dans le répertoire courant. `commande1 | commande2` donne en paramètre de `commande2` la sortie de la commande `commande1`. Par exemple, `echo "bonjour tout le monde" | wc -w` compte le nombre de mots contenus dans la chaîne "bonjour tout le monde".

Soit le fichier *real.csv* contenant le texte suivant :

real.csv
```
Nom ; Prenom ; Naissance
Hitchcock ; Alfred ; 1899
Welles ; Orson ; 1915
Kubrick ; Stanley ; 1928
Fellini ; Federico ; 1920
Godard ; Jean-Luc ; 1930
Coppola ; Francis ; 1939
Renoir ; Jean ; 1899
Bergman ; Ingmar ; 1918
Ford ; John ; 1894
```

1. Donner la commande qui permet d'obtenir le fichier *liste.csv* qui est une copie de *real.csv* sans la première ligne (utiliser `tail` et `>`).
2. Donner la commande pour obtenir un affichage de la liste des réalisateurs par noms croissants (utiliser `cat`, `sort`, et `|`).
3. Sauvegarder le résultat de la dernière question dans un fichier `tri.csv`.

Exercice 2 (Compilation) :
Afin de compiler un fichier contenant du code C (par exemple `mon_programme.c`) et ainsi produire un exécutable il suffit d'utiliser le compilateur `gcc` sur ce fichier. Ce compilateur peut être utilisé en ligne de commande :

```
janson@mp2i-janson:~$ gcc mon_programme.c
```

Cette commande produira alors un exécutable `a.out` dans le dossier où la commande `gcc` a été appelée. Ainsi, afin d'exécuter le programme C, il suffira (après compilation bien sur) d'exécuter l'exécutable ainsi généré, ce qui peut être fait avec la commande (en supposant que l'exécutable `a.out` se trouve dans le dossier courant du terminal) :

```
janson@mp2i-janson:~$ ./a.out
```

Il n'est pas très pratique de ne pas pouvoir choisir le nom de l'exécutable ainsi généré. Heureusement, `gcc` fourni une option permettant de le préciser. La commande suivante compilera le programme `mon_programme.c` et génèrera un exécutable nommé `mon_executable`.

```
janson@mp2i-janson:~$ gcc mon_programme.c -o mon_executable
```

1. Supposons que l'on vienne juste d'exécuter la commande `$ gcc mon_programme.c -o mon_executable`. Donner la commande qui permet d'exécuter l'exécutable ainsi généré.

L'option `-o` est seulement une des options que nous utiliserons couramment pour compiler nos fichiers C. Nous utiliserons aussi les options :

- `-Wall` et `-pedantic` permettant d'afficher plus de warnings du compilateur ;
- `-Werror` permettant de traiter les warnings comme des erreurs.

2. Télécharger le fichier `ready.c` sur la page web du cours. Compilez-le, et exécutez-le.
3. Ouvrir le fichier `ready.c` dans VSCode. Décrire ses composants et ce qu'il fait.

Exercice 3 (Premiers pas) :
Tous les programmes du reste du TP seront stockés dans des fichiers différents. Pour démarrer, n'hésitez pas à vous inspirer de `ready.c`.

1. Écrire un programme qui affiche "`Hello, world!`" suivi d'un retour à la ligne sans le terminal.
2. Écrire un programme qui affiche un rectangle de taille 6×5 composé uniquement d'astérisques (`*`). Le résultat attendu est :

Output
```
*****
*****
*****
*****
*****
*****
```

3. Écrire un programme qui affiche un rectangle de taille 6×5 composé uniquement d'astérisques (`*`) et de plus (`+`) formant des diagonales qui alternent. Le résultat attendu est :

Output
```
*+*+*
+*+*+
*+*+*
+*+*+
*+*+*
+*+*+
```

Exercice 4 (Interaction avec l'utilisateur) :
Nous avons vu dans l'exercice précédant comment afficher des messages dans le terminal pour l'utilisateur. Nous voyons ici comment ajuster le contenu de ces messages pour refléter les valeurs des variables du programme. De même nous allons voir comment l'utilisateur peut faire passer de l'information à un programme qui s'exécute.

La fonction `printf` de la librairie standard que vous avez utilisé dans les questions précédentes accepte des symboles spéciaux à l'intérieur de la chaîne de caractère qui lui est passée comme premier argument. Ces symboles sont remplacés au moment de l'exécution du `printf` par des valeurs passés comme arguments à `printf`. Ainsi le symbole `%d` sera remplacé par un entier, le symbole `%f` par un flottant et `%s` par une chaîne de caractère.

```
1 printf("ent : \%d, \%f, s: \%s et ent2   : \%d", 1, 3.5, "toto", 2)
```

À titre d'exemple, l'exécution de l'instruction ci-dessus imprimera dans le terminal la chaîne de caractère :

Output
```
ent : 1,   3.500000, s: toto et ent2 : 2
```

Les symboles spéciaux ont été remplacés (dans le même ordre) par les valeurs passées en argument à `printf`.

1. Donner le retour dans le terminal de l'exécution du programme suivant :

printf_test.c

```c
#include <stdio.h>

int main() {
  int i;
  float a;
  i = 0 ;
  a = 9.8;
  printf("%d %d ceci est un test\n", i, 3);
  printf("Ce peut %f être mis %simporte où", a, "n'");
  i = i + 1;
  printf("%d", i);
  printf("ent : \%d,   \%f, s: \%s et ent2   : \%d", 1, 3.5, "toto", 2);
}
```

Afin de lire une entrée de l'utilisateur nous utiliserons la fonction `scanf` de la bibliothèque standard, telle que `scanf("%f", &v);` attend que l'utilisateur entre une valeur (`%f` indique qu'il s'agit d'un nombre flottant) et l'enregistre dans la variable `v` (déclarée avec le type `float`) [1]. Évidemment si l'on souhaite lire un entier il suffira de remplacer le `%f` par un `%d`.

2. Écrire un programme qui répète tous les entiers entrés par l'utilisateur.
3. Écrire un programme demandant à l'utilisateur la note n qu'il a obtenu au dernier examen et affichant sa mention selon les règles suivantes :
   - $n < 10$: Échec
   - $10 \leq n < 12$: Peut mieux faire
   - $12 \leq n < 14$: Assez bien
   - $14 \leq n < 16$: Bien
   - $16 \leq n$: Très bien
4. Demander un entier n à l'utilisateur. Afficher tous les entiers de 1 à n (un par ligne).
5. (a) Demandez un entier n à l'utilisateur. Afficher les n premiers termes de la suite de Fibonacci :

$$F_0 = 1; \quad F_1 = 1; \quad F_{n+2} = F_{n+1} + F_n$$

   (b) Expliquer le comportement apparaissant quand $n \sim 50$.

Exercice 5 (Un mini-jeu "C'est plus, c'est moins") :
On génère un nombre aléatoire r dans l'intervalle $[\![0, n[\![$ avec la fonction `rand` :

```c
int r = rand() % n;
```

Cette fonction est inclue dans la bibliothèque standard : vous devez ajouter `#include <stdlib.h>` au début de votre programme.

1. Générer un entier aléatoire $x \in [\![0, 1000[\![$. Demander ensuite un nombre entier à l'utilisateur :
   - S'il est au dessus, le lui dire et redemander un entier ;
   - S'il est en dessous, le lui dire et redemander un entier ;
   - S'il trouve, c'est gagné !
2. Inverser le fonctionnement du jeu pour que cette fois-ci le joueur choisisse le nombre à deviner, et que ce soit l'ordinateur qui devine.

1. Attention à ne pas oublier le symbole '&' devant le nom de la variable. Nous verrons bientôt en cours le sens précis de ce symbole.