DLST

UNIVERSITÉ Grenoble Alpes

INF203 - Examen Terminal
Année 2021-2022

Durée : 2h.
Une feuille A4 manuscrite recto/verso autorisée, dictionnaire papier (non annoté) autorisé pour les étudiants étrangers uniquement.
Tout autre document, calculatrices et appareils électroniques interdits.
Pour chaque question, une partie des points peut tenir compte de la présentation.
Le barème est indicatif.

Toute réponse même incomplète sera valorisée à partir du moment où elle réalise au moins une partie de ce qui est demandé. Les questions sont relativement indépendantes, et dans tous les cas vous pouvez utiliser les scripts et les fonctions demandées dans les questions précédentes même si vous n'avez pas réussi à les écrire.

Par ailleurs, tout code ou algorithme *élégant*, en plus d'être correct, sera bonifié.

1 Programmation en C : Puissance 4 (10 pts)

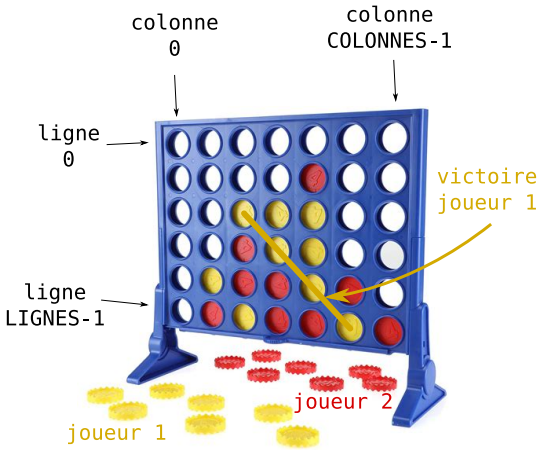
Le but de cet exercice est de recréer un jeu du *Puissance 4*.

Le jeu du *Puissance 4* oppose 2 joueurs sur une grille de `LIGNES` lignes et `COLONNES` colonnes. Tour par tour, chaque joueur place un jeton en haut de la grille (sur l'une des colonnes 0 à `COLONNES-1`). Le jeton va tomber jusqu'à atteindre la ligne la plus basse disponible (entre 0, tout en **haut**, et `LIGNES-1`, tout en **bas**) dans la colonne choisie. Le premier joueur parvenant à aligner 4 jetons sur la grille sans interruption par un jeton adverse ou une case vide, que ce soit en diagonale ou en ligne (horizontale ou verticale), gagne la partie. Si la grille est remplie sans qu'aucun joueur n'ait gagné, il y a match nul.

On fournit la structure suivante :

```
#define LIGNES 6
#define COLONNES 7

typedef struct{
    int grille[LIGNES][COLONNES];
    int etat_colonne[COLONNES];
    int joueur;
} jeu;
```



Le tableau `grille` représente le tableau de jeu. Les valeurs de ce tableau sont égales à 1 si le joueur 1 y a placé un jeton, 2 si le joueur 2 y a placé un jeton, et 0 sinon.

La case d'indice `i` du tableau `etat_colonne` représente la ligne à laquelle le jeton devra tomber si un joueur décide de jouer dans la colonne `i` du tableau grille. Elle sera égale à `-1` si la colonne est pleine.

Le champ `joueur` désigne le numéro du joueur qui jouera le prochain jeton. Au début d'une partie, le premier joueur à jouer sera toujours le joueur 1.

Questions :

1. (1 point) Écrire une fonction de profil `void init(jeu * J)` qui initialise les champs de la structure `jeu` fournie en arguments.
2. (2 points) Écrire une fonction de profil `void affiche(jeu * J)` qui affiche l'état courant de la grille du jeu `J`. On imprimera le symbole `|` pour délimiter les colonnes, le symbole `X` pour signifier l'emplacement d'un jeton du joueur 1, le symbole `O` pour un jeton du joueur 2 et un espace si aucun jeton n'est présent dans la case. On pourrait par exemple obtenir :

```
| | | | | | | |
| | | | | | | |
|X| | | | | | |
|O|O|X| | | | |
|O|X|X|O| | | |
|X|O|X|O|O|X|X|
```

3. (1 point) Écrire une fonction de profil `int lire_entree(jeu *J)` qui affiche le numéro du joueur pour qui c'est le tour de jouer et lit un entier sur l'entrée standard. Cette valeur représente la colonne où le joueur souhaite jouer. Cette valeur sera renvoyée par la fonction.
4. (2 points) Écrire alors une fonction de profil `void jouer(jeu * J, int entree)` qui utilise l'entier `entree` pour essayer de jouer dans l'état `J` du jeu pour le joueur `joueur` contenu dans le jeu `J`. Si le coup exécuté par `joueur` est autorisé, le jeu `J` se met à jour. Sinon la fonction affiche une erreur mentionnant que le coup est illégal à l'intention du joueur `joueur` et ne met pas le jeu `J` à jour.

   On souhaite maintenant pouvoir reprendre une partie en cours. L'idée étant que si un joueur émet une commande avec un nombre négatif `Nombre` au moment de choisir sa colonne de prochain coup, la partie est sauvegardée dans le fichier `partie_en_coursNombre`. Par exemple, si le joueur tape -42, la partie sera sauvegardée dans le fichier `partie_en_cours-42`. On va pour cela créer 2 fonctions permettant de sauvegarder et charger une partie en cours dans un fichier.

5. (2 points) Le fichier de sauvegarde contient (i) l'état de la `grille` du jeu `J` en cours, (ii) le numéro du joueur qui doit jouer le prochain coup.
   1. Écrire une fonction de profil `void sauvegarder_partie(jeu * J, int Nombre)` qui va enregistrer dans un fichier `partie_en_coursNombre` l'état de la partie.
   2. Écrire une fonction de profil `int charger_partie(jeu * J, char * nom_fichier)` qui va charger dans la variable `J` la partie en cours contenue dans le fichier `nom_fichier`.

   On suppose avoir à notre disposition une fonction `int etat_partie(jeu * J)` qui renvoie la valeur `0` si la partie est terminée et s'achève par un match nul, `1` si la partie est terminée et s'achève par la victoire du joueur 1, `2` si la partie est terminée et que le joueur 2 en sort vainqueur, et `-1` si la partie n'est pas terminée. *Il ne s'agit pour le moment pas de réaliser cette fonction.*

6. (2 points) En utilisant les fonctions précédentes, créer un programme principal. Ce programme vérifie si un argument lui a été fourni. Si c'est le cas, cet argument contient le nom d'un fichier d'une partie sauvegardée qui devra alors être chargée. Sinon, on initialise une partie vide. On simulera ensuite la partie en affichant à chaque tour l'état de la grille et en demandant au joueur suivant de jouer. Si un joueur fournit un nombre négatif, on sauvegarde la partie et on quitte le programme. Si la partie se termine, on affiche le résultat de la partie avant de finir le programme.
7. (Bonus points) Implémenter la fonction de profil
   - `int scan_lignes(jeu * J)`

   qui renvoie 1 ou 2 si une ligne contigüe de 4 jetons du joueur 1 ou 2 est découverte dans la grille, sinon -1.
   On suppose de manière similaire que les fonctions de profils
   - `int scan_colonnes(jeu * J)`
   - `int scan_diagonales_NE(jeu * J)` (pour Nord-Est)
   - `int scan_diagonales_NO(jeu * J)` (pour Nord-Ouest)

   qui renvoient elles aussi 1 ou 2 si une suite contigüe de 4 jetons du joueur 1 ou 2 en colonne ou diagonale est découverte, sinon -1, ont été également implémentées. En vous appuyant sur ces quatre fonctions, implémenter alors la fonction `int etat_partie(jeu * J)`.

2 Programmation en bash : Pendu `cowsay` (10pts)

On veut implémenter ici en Bash une version du jeu du pendu en exploitant le script `cowsay`. Pour cela, on dispose d'un fichier `dictionnaire` qui contient un ensemble de mots écrits en majuscule que le joueur sera amené à retrouver lettre par lettre. Le fichier `dictionnaire` contient un mot par ligne, dont le début prend la forme suivante :

```
ARBRE
COMPILATION
IMPLEMENTATION
LANGAGE
```

La vache affichera au moyen du symbole `_` l'ensemble des lettres qu'il reste à deviner, et remplacera toutes les occurrences de `_` correspondant à une lettre découverte par l'utilisateur. Elle affichera également le nombre d'erreurs en cours sur le nombre total d'erreurs autorisées.

Par exemple, si le mot à deviner est `IMPLEMENTATION` et que l'utilisateur entre successivement les caractères `E`, `U`, `T`, `I`, `R`, `P`, on obtiendra le résultat suivant.

```
 ----------------
/ I_P_E_E_T_TI__ \
|                |
\ 2/10 erreurs   /
 ----------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Si le mot est découvert avant d'atteindre le nombre maximal d'erreurs, la vache est heureuse et renvoie

```
 ----------------
/ IMPLEMENTATION \
|                |
\ Victoire !     /
 ----------------
        \   ^__^
         \  (@@)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Si le nombre maximal d'erreurs est atteint, la pauvre vache décède et envoie

```
 ----------------
/ I_PLE_ENT_T_ON \
|                |
\ Défaite !      /
 ----------------
        \   ^__^
         \  (XX)\_______
            (__)\       )\/\
             U  ||----w |
                ||     ||
```

Questions : La commande `echo -e` permet de prendre en compte les caractères spéciaux d'une chaine de caractères à afficher, tels que `\n` pour afficher un passage à la ligne.

1. (1 point) Grâce à cette remarque et la commande `cowsay`, proposer 3 instructions qui produisent respectivement les 3 affichages précédents. On rappelle que les options `-e` suivie de 2 caractères (pour "eyes") et `-T` suivie d'un caractère (pour "tongue") de `cowsay` modifient l'affichage des yeux et de la langue. On rappelle également que, par défaut, `cowsay` prononce la phrase écrite sur l'entrée standard.

2. (2 points) Écrire un script `choix_mot.sh` sans argument qui affiche un mot aléatoirement choisi dans le fichier `dictionnaire`. Pour cela, on pourra dans un premier temps compter le nombre de mots présents dans le fichier `dictionnaire`. Afin de sélectionner au hasard l'un de ces mots, on pourra alors utiliser la variable `$RANDOM` qui contient un nombre généré aléatoirement (entre 0 et 32 767), ainsi que l'opérateur 'modulo' `%` de la commande `expr`.

   Le comportement suivant pourra par exemple être observé :

   ```
   user@laptop:~$ ./choix_mot.sh
   IMPLEMENTATION
   ```

   On suppose disposer d'une variable `lettres_restantes` qui contient initialement l'ensemble des lettres majuscules de l'alphabet :

   ```
   user@laptop:~$ lettres_restantes=$(echo -e {A..Z})
   user@laptop:~$ echo $lettres_restantes
   A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
   ```

3. (2 points)

   1. En utilisant la commande `tr`, donner une commande qui modifie la variable `lettres_restantes` en remplaçant le caractère `CARACTERE` (entre 'A' et 'Z') par un espace.

      Par exemple, si `CARACTERE=I` et que `lettres_restantes` contient initialement l'ensemble des caractères de A à Z, on doit obtenir suite à la commande :

      ```
      user@laptop:~$ echo "$lettres_restantes"
      A B C D E F G H   J K L M N O P Q R S T U V W X Y Z
      ```

   2. Donner ensuite un script `affichage.sh` qui prend deux arguments : le mot à deviner et une chaîne de caractères contenant les lettres restantes. Ce script affiche le mot à deviner en remplaçant tous les caractères contenus dans les lettres restantes par le symbole `_`.

      Par exemple, si `lettres_restantes` a été modifiée comme dans la question précédente, on devra obtenir :

      ```
      user@laptop:~$ ./affichage.sh IMPLEMENTATION "$lettres_restantes"
      I__________I__
      ```

   On veut enfin mettre en place un script bash complet `cow_pendu.sh` qui simule le jeu du pendu. L'exécutable prend 1 argument optionnel correspondant au niveau de difficulté, à savoir le nombre d'erreurs autorisées. Par défaut (si aucun argument n'est donné), le nombre d'erreurs autorisé est mis à 10. Le jeu se déroule ainsi :

   a. à l'exécution, un mot est choisi au hasard dans le fichier `dictionnaire`
   b. à chaque tour du jeu, une utilisation pertinente de `cowsay` affiche, comme proposé en début d'exercice, une vache qui donne l'état du jeu à l'instant courant
   c. le programme demande à l'utilisateur de taper une lettre au clavier et met à jour les lettres restantes
   d. si le mot est découvert avant d'avoir atteint le nombre d'erreurs maximal, la vache déclare une victoire ; si le nombre maximal d'erreurs est atteint, la vache déclare une défaite ; sinon le jeu se poursuit.
   e. En cas de victoire ou défaite, le programme demande alors si le joueur souhaite rejouer. Si oui, une autre instance du jeu est lancée. Sinon, le programme se termine.

4. (2 points) On fait temporairement l'hypothèse simplificatrice où `dictionnaire` ne contient que des mots de deux lettres *distinctes* de la forme $l_1 l_2$ avec $l_1 \neq l_2$ et qu'une seule erreur est autorisée. Dans ces conditions, décrire un automate complet du jeu `cow_pendu.sh` en précisant l'ensemble de ses états, de ses états de sortie, de ses entrées et de ses sorties. Dessiner l'automate (pour des raisons de lisibilité, on pourra omettre d'afficher les sorties sur le dessin).
5. (3 points) On se replace ici dans le cadre général de `cow_pendu.sh`. En réutilisant tout ou partie des scripts produits précédemment, proposer une implémentation Bash pertinente *et duement commentée* de `cow_pendu.sh`.