Évaluation de projet (PIX111) – Partie Informatique – Sur Machine

Durée totale : 1 heure 30 / Tiers Temps = 2 heures

Toute communication (orale, téléphonique, par messagerie, etc.) avec les autres étudiant·e·s est interdite. La feuille de syntaxe C est le seul document autorisé. Internet non autorisé.

— Tout comme l'examen de CS101, cette évaluation machine est probablement **trop longue**. Nous en tiendrons compte.

— L'évaluation est sur machine, mais quelques réponses sont à rédiger sur papier (questions avec ✎).

— Nous récupérerons le répertoire de travail. Vous ferez attention à ce que les programmes **compilent sans Warning ni erreur** (cela fait partie de la notation). Si votre programme ne compile pas, mettez en commentaire les essais réalisés pour une correction partielle.

— Le barème (pour l'instant sur 22 points) est indicatif. Il indique la pondération/ la difficulté respectives des questions.

Préparation

Pour les manipulations préparatoires, on donne des commandes Linux examples que vous pouvez choisir d'utiliser **ou non**. (L'utilisation du gestionnaire de fichier, du clic droit "ouvrir un terminal ici", est autorisée ...)

1. Récupérer l'archive pour cette évaluation (instructions données en séance), et la désarchiver quelque part (sauf dans le répertoire `/tmp`). Se placer dans le répertoire parent du répertoire obtenu (qui s'appelle `pixeval22-stud`).
2. Modifier le nom de ce répertoire à votre NOM (pas d'espace, pas d'accent). Vous ne travaillerez que dans ce répertoire.

   ```
   mv pixeval22-stud Turing
   ```

3. Lancez un terminal et un éditeur de texte qui travaillent dans ce répertoire. Laissez le terminal et l'éditeur ouverts tout au long de cette évaluation.

   ```
   cd Turing
   scite &
   ```

4. Il est conseillé de sauvegarder ce répertoire régulièrement en le copiant ailleurs (mais toujours en laissant le reste inchangé)

   ```
   (autre terminal)
   cp -r Turing Turing_save1
   ```

5. Les instructions pour rendre vos programmes seront aussi données durant l'évaluation.

Solution: En séance, ces manipulations ont en fait consisté en le renommage du répertoire fourni directement sur les sessions. Le rendu a été réalisé via : compression en zip, copie sur une clé USB.

Connaître la ligne de commande pour compiler est indispensable à ce stade. De même, vous devez savoir utiliser votre éditeur favori lorsque celui ci n'est pas celui qui s'ouvre en cliquant deux fois : clic droit "ouvrir avec". Enfin, trop de personnes essaient de compiler un fichier sans être au bon endroit.

Exercice 1 : échauffement - exo1.c

Solution: Les solutions sont en fait des indications de correction qu'il faudrait mieux rédiger.

Question #1 (1 point)

Dans `exo1.c` dans le `main` faire imprimer "Bonjour!". Vérifier que le programme compile bien :

```
clang -o exo1 exo1.c -Wall
```

Vérifier l'exécution.

À partir de maintenant, vous compilerez et exécuterez à chaque question, quel que soit l'exercice.

Question #2 (1 point)

Mettre un commentaire au début du fichier avec votre nom.

Question #3 (1 point)

Dans la fonction principale (main), déclarez 3 entiers nommés x, y et z.

Question #4 (1 point)

Initialiser x et y avec des valeurs demandées à l'utilisateur·rice. (utilisation de `scanf` deux fois de suite).

Question #5 (1 point)

Écrire une fonction `int min (int a, int b)` qui calcule et retourne le minimum des deux entiers passés en paramètre. Appeler cette fonction dans le main pour récupérer dans la variable z la valeur minimum de x et y, puis imprimer ce minimum :

```
Le minimum de x et y est ....
```

Solution: Pas besoin de plus de commentaire.

```c
/* Source code from Laure Gonnord*/
#include <stdio.h>
#include <stdlib.h>

int min(int u, int v){
  if (u>v) return v;
    else return u;
}

/* Fonction principale*/
int main(){
  int x,y, z;
  printf("valeurs de x et y?\n");
  scanf("%d",&x);
  scanf("%d",&y);
  z = min(x,y);
  printf("le minimum de x et y est %d\n",z);
  return 0;
}
```

Il n'est pas nécessaire de trop commenter le code. le fait que la fonction min calcule un minimum, bon, ici on le voit! Je rappelle qu'il n'est pas exigé de mettre un unique return (oui, quelquefois jugé bonne pratique, ici on est au delà des) préoccupations de 1A, et c'est non avenu sur une fonction de 5 lignes. Attention si vous ne faites pas un if then else il est possible que vous oubliez des cas.

Exercice 2 programme mystère (exo2.c)

Les question marquées d'un symbole ✎ doivent être rédigées sur votre copie papier.

Solution: Le code proposé était celui-ci :

```c
#include <stdio.h>
#define N 3

// Exercice 2 programme mystère

void imprime_tab (int t[N]){
  // AFAIRE : imprimer le contenu du tableau t
}

void echange(int t[N],int i,int j){
  // AFAIRE: échanger le contenu des cases d'indice i et d'indice j
}

void init_tab(int tt[N]){
  int i;
  for (i=0;i<N;i=i+1){
    tt[i] = i+1;
  }
}

void mysteriousRec(int tab[N],int k) {
  int j;
  if ( k == N-1 ) imprime_tab (tab);
  for (j = k ; j < N ; j ++) {
    echange (tab,k,j);
    mysteriousRec(tab,k+1);
    echange (tab,k,j);
  }
}

int main(){
  int montab[N];

  // Appel à la fonction d'initialisation
  init_tab(t)

  // A FAIRE : tester la fonction d'impression
  // (1 appel de fonction)



  // A FAIRE : tester votre fonction d'échange
  // (1 appel de fonction)
  // puis mettre l'appel en commentaire


  // A FAIRE (à la fin de l'exercice) :
  // décommenter la ligne suivante
```

```
  // mysteriousRec(montab,0);


  return 0;
}
```

Question #6 (1 point)

Ce programme ne compile pas, résoudre les deux erreurs pour que la compilation se passe avec succès.

Solution:

Deux erreurs simples, un mauvais appel (identifiant de tableau) et un ; dans la même ligne :

```
init_tab(t)
```

à changer en :

```
init_tab(montab);
```

Pour résoudre le souci d'identifiant de tableau, certaines personnes ont inventé un tableau, alors qu'un tableau avait été déclaré déjà. IL suffisait de l'utiliser. Je n'ai pas compté faux, mais l'usage de ces deux tableaux n'avait pas été prévu...

Question #7 (1 point)

Dans le fichier, écrire le contenu de la fonction `imprime_tab`, et l'appeler dans le *main*. Tester.

Solution: On ajoute cette fonction :

```
void imprime_tab (int t[N]){
   int i;
  for(i=0;i<N;i=i+1){
    printf("%d␣",t[i]);
  }
  printf("\n");
}
```

et cet appel dans le main :

```
  imprime_tab(montab);
```

Question #8 (2 points)

Même question pour la fonction `echange`.

Solution:

```
void echange(int t[N],int i,int j){
  // AFAIRE: échanger le contenu des cases d'indice i et d'indice j
  int tmp = t[i];
  t[i]=t[j];
  t[j]=tmp;
}
```

Dans le main, on peut faire 1 appel de fonction ou alors vérifier en réimprimant, ce qui est mieux.

```
imprime_tab(montab);
echange(montab,1,2);
imprime_tab(montab);
```

Question #9 (2 points)

✎ Décommenter la ligne d'appel à la fonction mystère. Recopier la sortie obtenue sur votre copie. Dans le cas général, que fait cette fonction ?

Solution: Sur l'entrée donnée cela donne : si on a utilisé le tableau déclaré tout au long du programme, celui-ci est initialisé correctement via la fonction initialise.

```
1 2 3
1 2 3
1 3 2
2 1 3
2 3 1
3 2 1
3 1 2
```

La fonction énumère les permutations des éléments du tableau d'entrée.

Exercice 3 : mots mêlés

Dans cet exercice, nous allons jouer aux mots mêlés, c'est à dire rechercher des mots (appelés *motifs*) dans une grille dans n'importe quel sens de lecture. Par exemple, les mots `VIDE`, `OGM`, et `ARBRE` sont présents (et entourés) dans la grille de la figure 1.

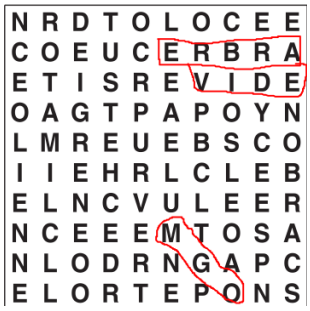

FIGURE 1 – Une grille de mots mêlés.

Dans le jeu, on donne avec la grille une liste de mots à rechercher dans la grille par l'utilisateur·rice, et lorsque l'ensemble des mots est trouvé, la partie est terminée. Nous n'allons pas ici réaliser cette boucle de jeu, mais uniquement quelques fonctions préparatoires.

Implémentation On vous fournit un programme `exo3.c` (unique, il n'y a pas deux .c comme dans les séances de projet) qui réalise une partie du travail, ainsi qu'un fichier `grille.txt`. Nous allons tout d'abord analyser le code fourni, et ensuite vous réaliserez quelques fonctions.

La fonction d'initialisation du jeu vous est donnée (`initialise_jeu`, et appel dans le main). On rappelle que l'accès à une matrice 2d se fait comme ceci `mat[l][c]` avec l indice de ligne et c l'indice de colonne.

Solution: Voici le code fourni :

```c
// Exo 3: mots mêlés.

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <ctype.h>

#define N 10
#define M (2*N)

// La grille sera stockée dans la variable suivante:
char grille[N][N];
//et les mots ici
char mots[M][N];
// et le statut de la grille ici. true == lettre barrée
bool statut[M][N];


/*Initialise le jeu  à partir du fichier fic, ouvert*/
int initialise_jeu(FILE* fic)
{
  int i, j ;
  // Récupération de la grille
  char oneline[N]; // chaine
  for (i=0;i<=N-1;i++)
    {
      // récupération d'une ligne
      fscanf(fic, "%s",oneline);
      // copie de N caractères dans la matrice t, "ligne i"
      strncpy(grille[i],oneline,N);
    }
  // récupération des mots cachés
  // On considère qu'il y au plus 2N mots cachés, on ne vérifie pas.
  int nbmots;
  fscanf(fic, "%d", &nbmots);
  for (i=0;i<=nbmots-1;i++)
    {
      // récupération d'une ligne
      fscanf(fic, "%s",oneline);
      // copie du mot
      strncpy(mots[i],oneline,N);
    }
  // Initialise le tableau de statut de la grille
  for (i=0;i<N;i++){
    for (j=0; j<N; j++)
      statut[i][j]=false;
  }

  return nbmots;
}

/*imprime un tableau de chaînes de caractères (tableau 2D)*/
void imprime_chaines(char strtab[M][N], int nbmots){
```

```
  int i;
  for (i=0;i<nbmots;i++){
    printf("mot␣%d:␣%s,␣", i, strtab[i]);
  }
  printf("\n");
}


/*imprime une matrice de caractères*/
void imprime_mat(char t[N][N]){
  int i,j;
  for (i=0;i<N;i++){
    for (j=0;j<N;j++)
      printf("%c,",t[i][j]);
    printf("\n");
  }
}


bool trouve_mot_horizontalementGD(int motindex, int l, int c){
  // Le mot que l'on cherche est mots[motindex]
  // renvoie true si le mot est trouvé horizontalement à partir de l,c
  // dans la grille, false sinon.
  // // AFAIRE
  return false;
}



/*Fonction principale*/
int main(){

  FILE* f = fopen("grille.txt","r");
  if (f==NULL){
    printf("Fichier␣de␣grille␣manquant!\n");
    return 1;
  }
  // Initialisation de la grille, et du tableau contenant les mots à
trouver
  int nb = initialise_jeu(f);
  printf("impression␣de␣la␣grille␣de␣mots␣mêlés!\n");
  imprime_mat(grille);
  printf("Il␣y␣a␣%d␣mots␣à␣trouver!\n", nb);
  imprime_chaines(mots, nb);

  // Pour tester votre fonction trouve_mot_horizontalementGD:
  // AFAIRE décommenter cette ligne en remplaçant les ? par des vrais nombres
  // if (trouve_mot_horizontalementGD(17,?,?)) printf("le mot num 17 est trouvé


  fclose(f);
  return 0;
}
```

Question #10 (3 points)

✎ Compiler et exécuter le code fourni (`exo3.c`), puis rédiger sur papier les réponses aux questions suivantes :

1. Que réalise ce code fourni ? Quelle est la taille de la grille fournie ? Quelles informations supplémentaires sont présentes dans le fichier `grille.txt` ?
2. Expliquer pourquoi la fonction `initialise_jeu` n'a pas de paramètre tableau.
3. Expliquer comment est récupéré/calculé le nombre de mots à découvrir.

Solution: Le code fourni ouvre le fichier de grille en lecture et remplit la variable globale associée, de taille N*N avec ici N=10. Le fichier de grille contient aussi le nombre de mots à chercher dans la grille et ceux ci. La fonction d'initialisation n'a pas de paramètre car elle modifie les variables globales déclarées au début (à partir d'un FILE passé lui, en paramètre). Le nombre de mots à découvrir dans la grille est un entier dans le fichier de la grille.

L'objet des questions suivantes est d'écrire le code de la fonction `trouve_mot_horizontalementGD`, qui étant donné un indice de ligne l (ℓ), et un indice de colonne c, renvoie true si le mot est effectivement **complétement trouvé dans la grille à la place indiquée**, false sinon. Pour réaliser ce code, nous commençons par prendre un exemple : le mot VIDE (ce mot est le numéro 17 dans la liste des mots à trouver).

Question #11 (1 point)

✎ Regardons la troisième ligne de la grille :

j	0	1	3	4	5	6	7	8	N-1	
grille[??][j]	E	T	I	S	R	E	V	I	D	E

Quels sont les indices (ligne, colonne) du V de VIDE dans la grille ? dans le tableau de mots ?

Solution: `grille[2][6]` et `mots[17][0]`.

Question #12 (1 point)

✎ Supposons que l'on connaît la taille du mot à chercher (ici 4), comment vérifier que le mot "ne dépasse pas de la grille" ?

Solution: Dans la fonction "horizontale gauche droite", il suffit de vérifier que les indices de colonne ne dépassent pas à "droite", cad $c + taille - 1 < N$ avec c l'index de colonne.

Question #13 (2 points)

Implémenter la fonctionnalité dans `exo3.c`.

Solution:

```
    bool trouve_mot_horizontalementGD(int motindex, int l, int c){
  // Le mot que l'on cherche est mots[motindex]
  // renvoie true si le mot est trouvé horizontalement à partir de l,c
  // dans la grille, false sinon.
  int taille=strlen(mots[motindex]);
  // Si on ne se rappelle pas de comment récupérer la taille d'un mot, on trans
la sentinelle et à ne pas dépasser de la grille en même temps. Ou alors on réim
  if ((c+taille -1) >= N) return false;
  else {
    // la ligne est fixe, la colonne change dans grille
    for (int i=0; i<=taille-1; i++){
      if (mots[motindex][i] != grille[l][c+i]) return false;
    }
```

```
    return true;
  }
  return false;
}
```

Question #14 (1 point)

Décommenter et compléter l'appel dans le main à la fonction `trouve_mot_horizontalementGD.` , il manque juste les valeurs de c et l (ℓ) adéquates pour tester cette fonction avec le mot VIDE.

Solution:

```
      if (trouve_mot_horizontalementGD(17,2,6)) printf("le␣mot␣num␣17␣est␣trouv
```

Le mot "OGM" se trouve dans la grille dans les trois dernières lignes, "en biais de bas en haut".

Question #15 (1 point)

✎ Donner les indices dans la grille et dans le tableau mots pour cet exemple. Expliquer soigneusement comment parcourir la grille en diagonale inverse pour vérifier que le mot existe.

Solution: flemme de rédaction.

Question #16 (2 points)

Implémenter (toujours dans `exo3.c`) complètement une fonction qui réalise cette recherche (nom, signature, corps de la fonction) ainsi qu'un test adéquat.

Solution: idem.

Les questions suivantes sont des questions bonus

On désire maintenant modifier le tableau statut lorsqu'un mot est complétement trouvé.

Question #17 (0 points)

✎ Réaliser une analyse pour expliquer votre futur code et comment le tester.

Question #18 (0 points)

Implémenter les fonctions prévues. Si elles ne fonctionnent pas, mettez les en commentaire dans votre programme.

Solution: Le barème final était sur 22 points.