TP4 : Structures, fichiers

MP2I Lycée Pierre de Fermat

Consignes

Vous déposerez votre rendu sur cahier de prépa, avant ***mardi 01/11 à 22h00***.

Veillez à bien respecter les attendus précis des questions : si une question demande un ***programme***, vous devrez rendre un programme, c'est à dire un fichier C contenant un main, compilable.

Vous devez ***commenter*** toutes vos fonctions. Le commentaire d'une fonction doit :

- Expliquer le rôle de la fonction, ce qu'elle fait, sans expliquer les rouages internes de la fonction. Si votre fonction utilise un algorithme qui n'est pas trivial, vous pouvez décrire l'algorithme dans une deuxième partie, après avoir expliqué le rôle de la fonction. Les explications de comment est codée la fonction en interne sont faites dans le corps de la fonction, en commentant les lignes nécessaires, et pas dans le commentaire de documentation générale de la fonction.
- Mentionner explicitement le nom de chaque paramètre de la fonction.
- Ne pas commencer par "Cette fonction", "Prend en paramètre", ou "Cette fonction prend en paramètre", et aller directement à l'essentiel.
- Être valable indépendamment du sujet, et du reste du programme. La documentation d'une fonction doit dire ce que cette fonction fait précisément, on doit pouvoir comprendre comment l'utiliser juste en lisant le commentaire, sans document additionnel

Par exemple, "Prend en entrée un tableau et le trie comme indiqué dans l'énoncé" n'est pas un bon commentaire, mais "Trie le tableau tab dans l'ordre croissant" l'est.

Vous devez aussi mettre des ***assertions*** lorsque les paramètres de vos fonctions doivent vérifier des hypothèses : nombres positifs, pointeurs non nuls, etc...

Votre archive de rendu doit contenir un fichier de réponse, et doit aussi contenir un sous-dossier par exercice. Chaque sous-dossier contiendra le ou les fichiers C demandés dans l'énoncé de l'exercice. Vous devez rendre uniquement le code et supprimer les fichiers exécutables.

Votre fichier de réponse doit être un fichier ***".txt"***. Si vous êtes sur Mac, veillez à bien me rendre un fichier texte simple et pas un fichier .docx car je ne peux pas les lire. Cherchez sur internet comment faire des fichiers .txt sur Mac si vous ne savez pas le faire.

Vous êtes encouragés à travailler à plusieurs, mais veillez à ***ne pas rendre du code copié*** sur vos camarades.

Vous trouverez sur Cahier de Prépa une archive contenant des fichiers pour ce TP.

*
* *

Dans certains exercices de ce TP, vous aurez besoin de manipuler les chaînes de caractères. Au dernier TP, vous avez codé des fonctions pour calculer la longueur d'une chaîne, copier une chaîne dans une autre, concaténer des chaînes, et comparer. Plutôt que de reprendre vos fonctions, vous pouvez utiliser la librairie <string.h>, qui contient les fonctions :

- strlen (comme string length) calcule la longueur d'une chaîne de caractères
- strcpy (comme string copy) copie le contenu d'une chaîne de caractères vers un autre pointeur
- strcat (comme string concatenate) concatène une chaîne de caractères à la fin d'une autre.
- strcmp (comme string compare) compare deux chaînes de caractères, et renvoie 0 si elles sont égales, et ± 1 si elles ne sont pas égales, selon laquelle est la plus grande dans l'ordre lexicographique.

Vous pouvez regarder la spécification exacte de ces fonctions en regardant dans le ***Manuel***, avec la commande ***man*** du terminal. Par exemple pour savoir comment fonctionne strcat :

man strcat

Allocation dynamique

Exercice 1.

Question 1. Écrire une fonction prenant en entrée un entier n et renvoyant un tableau de n entiers, tous nuls.

Question 2. Écrire une fonction prenant en entrée deux entiers n et m, et renvoyant un tableau 2D de $n \times m$ entiers tous nuls.

Question 3. Testez vos fonctions dans un programme. N'oubliez pas de libérer toute la mémoire allouée.

Structures

Exercice 2.

Question 1. Proposez et implémentez des types struct permettant de modéliser la situation suivante :

— Une maison est constituée de plusieurs étages, chaque étage ayant plusieurs pièces

— On suppose toutes les pièces rectangulaires. Une pièce possède un nom ("chambre 1", "salon", etc...), des dimensions $a \times b$, un nombre de fenêtres et peut posséder du chauffage ou non.

Question 2. Pour chaque nouveau type, écrivez une fonction qui permet de libérer l'espace mémoire réservé par un pointeur de ce type.

Question 3. Écrivez une fonction qui permet de compter le nombre totale de fenêtres d'une maison

Question 4. Écrivez une fonction qui permet de compter la surface totale d'une maison.

Question 5. Écrivez une fonction qui permet de compter le nombre total de pièces chauffées d'une maison.

Question 6. Écrivez une fonction permettant d'empiler une maison m_1 sur une maison m_2, le résultat étant une nouvelle maison m_3 dont les étages inférieurs sont les étages de m_2, et dont les étages supérieurs sont les étages de m_1.

Question 7. Écrivez une fonction permettant de générer une maison aléatoire.

Question 8. Implémentez des types structs permettant de représenter une ville divisée en quartiers. On suppose qu'une ville possède des quartiers parfaitement réguliers, formant une grille rectangulaire. Chaque quartier a un certain nombre de maisons et un prix moyen du mètre carré.

Question 9. Écrivez une fonction permettant d'obtenir le coût total des maisons d'une ville.

Question 10. Écrivez dans la fonction main une démonstration des différentes fonctions précédentes.

Fichiers

On rappelle les fonctions permettant de manipuler les fichiers :

— ***fopen*** permet d'ouvrir un fichier, en mode "w" (écriture), "r" (lecture) ou "a" (ajout) :

```
1 FILE* fopen(char* filename, char* mode)
```

— ***fclose*** permet de fermer un fichier, et renvoie 0 si la fermture s'est bien passée :

```
1 int fclose(FILE* file)
```

— ***fprintf*** permet d'écrire dans un fichier. Exemple d'utilisation :

```
1 FILE* file = fopen("mon_fichier.txt", "w");
2 fprintf(file, "Blabla\n")
3 fprintf(file, "%d %f %s\n", 5, 2.39, "bla");
4 fclose();
```

— ***fscanf*** permet de lire dans un fichier. Exemple d'utilisation, si on a un fichier "test.txt" avec écrit 5 sur la première ligne, 6.32 sur la deuxième ligne :

```
1 FILE* file = fopen("test.txt", "r");
2 int n;
3 float f;
4 fscanf("%d", &n);
5 fscanf("%f", &f);
6 fclose(file);
```

Exercice 3.

Au TP dernier, nous avons vu que scanf("%s", str); a pour effet de lire dans le terminal et de stocker dans str ce qu'il lit jusqu'à rencontrer un espace, ou un retour à ligne.

Question 1. Implémentez une fonction void scan_ligne(char* str) qui lit le terminal caractère par caractère jusqu'à rencontrer un retour à la ligne, et écrit le résultat dans la chaîne str, sans stocker le retour à la ligne, mais en n'oubliant pas le '\0' final. On suppose que le tableau str a été alloué avec assez d'espace pour stocker la ligne.

Question 2. Proposez, sans nécessairement l'implémenter, une manière de modifier la fonction précédente pour qu'elle puisse scanner des lignes arbitrairement grandes. Cette nouvelle fonction ne prendra pas d'argument, et devra renvoyer un pointeur vers la chaîne résultante. (Indice : il faudra utiliser un ou plusieurs appels à malloc).

On peut facilement adapter les idées précédentes pour lire dans un fichier ligne par ligne. Appliquons cela à la lecture d'informations structurées dans un fichier.

Le fichier “”promo_2012.txt” contient la liste des élèves de MPSI2 qui étaient à votre place il y a 10 ans, en 2012. Le fichier contient :

- Le nombre total d'élèves, sur une ligne
- Pour chaque élève :
  - Sur une ligne, le prénom suivi du nom de famille
  - Sur une ligne, la date de naissance sous le format JJ MM AAAA
  - Sur une ligne, le métier de l'élève 10 ans plus tard

On propose de stocker ces informations dans le type struct suivant :

```
1 typedef struct eleve{
2   char nom_complet[40];
3   int jour;
4   int mois;
5   int annee;
6   char metier[40];
7 } eleve_t;
```

Question 3. Écrivez un programme qui demande à l'utilisateur un nom de fichier (maximum 20 caractères) et qui l'ouvre, et affiche les informations de la classe dont le fichier contient les informations : nombre d'élèves puis présentation de chaque élève en une phrase, chaque présentation séparée d'une ligne vide. Testez le sur le fichier “”promo_2012.txt”.

Utilisation du terminal

Exercice 4.

Pour cet exercice, vous allez avoir besoin de scanner dans le terminal des caractères seuls. Malheureusement, utiliser scanf seule ne suffit pas car, lorsque l'on utilise scanf avec des caractères, contrairement aux strings, les espaces et les retours lignes ne sont pas ignorés ! On propose la fonction suivante :

```
/* Renvoie true si le caractère c est blanc, i.e. un espace, un retour à la ligne,
   une tabulation... et false sinon (i.e. si c est un caractère lisible */
bool ignore(char c){
  if (c == ' ' || c == '\t' || c == '\r' || c == '\n'){
    return true;
  } else {
    return false;
  }
}
```

Question 1. En utilisant cette fonction, écrivez une fonction char scan_char() qui renvoie le premier caractère non blanc lisible dans le terminal. Testez cette fonction.

Question 2. Créez un programme "./afficheur" qui affiche toutes les 0.25 secondes une lettre aléatoire de l'alphabet en minuscule, et qui a une chance sur 20 de s'arrêter à chaque lettre rentrée.

Question 3. Lancez dans le terminal une commande qui exécute le programme "./afficheur" et stocke la sortie dans un fichier appelé "out.txt". Notez la commande que vous avez utilisé

Question 4. Créez un programme "./crieur" qui lit en boucle des lettres minuscules dans le terminal, et pour chacune affiche trois fois une version majuscule de la lettre, suivi d'un retour à la ligne. Si la lettre lue est "z", le programme s'arrête après avoir effectué le tour de boucle correspondant

Question 5. Lancez dans le terminal une commande qui exécute le programme "./crieur" en lisant dans le fichier "out.txt" au lieu de lire dans le terminal. Notez la commande que vous avez utilisé.

Question 6. Lancez dans le terminal une commande qui exécute le programme "./crieur" en lisant dans le fichier "out.txt" au lieu de lire dans le terminal, et qui stocke la sortie dans un fichier appelé "out2.txt". Notez la commande que vous avez utilisé.

Question 7. Lancez une commande qui exécute les programmes "./afficheur" et "./crieur", en utilisant la sortie du premier comme entrée pour le deuxième. Notez la commande que vous avez utilisé.

Exercice 5.

En C, la fonction ***main*** peut en fait prendre des arguments. Quand c'est le cas, au lieu d'écrire :

```
int main(){
  ...
}
```

on peut écrire :

```
int main(int argc, char** argv){
  ...
}
```

Alors, argc est un entier, et argv est un tableau de strings, de longueur argc. Le tableau argv contient tous les mots qui ont été écrits dans le terminal pour lancer le programme. Par

exemple, si on a un programme ./test où la fonction main a été écrite ainsi, et qu'on lance dans le terminal :

```
./test blabla 2.31 5
```

alors argv contiendra : {"./test", "blabla", "2.31", "5"}. En particulier, argc vaut toujours au moins 1, et argv[0] contient toujours le nom de l'exécutable. Remarquez que les nombres flottants et entiers sont stockés comme des chaînes de caractère. Si on veut les reconvertir en int ou en float il faut faire appel à des fonctions particulières : atoi et atof de la librairie <stdlib.h>. Ces fonctions prennent en entrée une chaîne de caractère et renvoient le nombre (int ou float) écrit dans la chaîne. Par exemple :

```c
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char** argv){
  int arg1 = 0;
  float arg2 = 0;
  if (argc > 2){
    arg1 = atoi(argv[1]);
    arg2 = atof(argv[2]);
  }

  printf("%d %f\n", arg1, arg2);

  return 0;
}
```

Question 1. Écrivez un programme qui prend en entrée (directement via la commande d'exécution) un nom de fichier, suivi d'une ou plusieurs lettres en minuscule et qui, pour chaque lettre, crée un fichier [cette_lettre].txt contenant tous les mots du fichier qui commencent par cette lettre. Le fichier "dico.txt" dans l'archive du TP contient une liste de mots, pas dans l'ordre alphabétique, pour que vous puissez tester votre programme. Les fichiers créés n'ont pas non plus besoin d'être dans l'ordre alphabétique.

Question 2. Sans le programmer, comment feriez-vous pour remettre un fichier comme "dico.txt" dans l'ordre alphabétique avec un programme C ? Vous décrirez les structures / algorithmes dont vous auriez besoin, même si vous ne savez pas comment les implémenter en pratique.

Test de programme

Dans cette section, vous allez apprendre à automatiser les tests de programmes.

Exercice 6.

Question 1. Créez un programme qui demande en entrée un entier n, et qui affiche en sortie le carré de n, sans rien afficher d'autre. Compilez ce programme, en appelant l'exécutable "carre", et testez le sur une ou deux valeurs.

Pour automatiser nos tests, nous allons créer dans le répertoire de l'exercice deux dossiers, test_in et test_out. Dans test_in, on mettra des fichiers contenant des entrées pour le programme, et dans test_out, les sorties correspondantes.

Question 2. Créez dans le dossier test_in des fichiers 1.in, 2.in, 3.in, et créez dans le dossier test_out des fichiers 1.out, 2.out, 3.out

Question 3. Pour i valant $1, 2, 3$, écrivez dans i.in un entier quelconque, et dans i.out le carré de cet entier.

Question 4. Donnez une commande du terminal permettant de lancer ./carre en précisant que le programme doit lire dans le fichier test_in/1.in au lieu du terminal. Testez cette commande et vérifiez qu'elle fonctionne bien.

Le terminal est muni de son propre langage de programmation : le langage Bash ! En Bash, les if-else s'écrivent :

```
if [[CONDITION]];
  INSTRUCTIONS
else
  AUTRES INSTRUCTIONS
fi
```

Voici deux commandes Bash particulièrement utiles :

— ***echo*** affiche dans le terminal ce qu'on lui donne en argument. Par exemple :

```
guillaume@MSI:~/TP3/exo3$ echo bonjour tout le monde
bonjour tout le monde
```

— ***cat*** affiche dans le terminal le contenu du fichier qu'on lui donne en argument. Par exemple, si l'on a écrit 36 dans test_out/1.out :

```
guillaume@MSI:~/TP3/exo3$ cat test_out/1.out
36
```

Question 5. Sans la lancer, que fait la suite de commandes suivante dans le terminal :

```
if [[ $(./carre < test_in/1.in) == $(cat test_out/1.out) ]]; then
  echo Oui;
else
  echo Non;
fi
```

Question 6. Testez la commande précédente. Que se passe-t'il si l'on remplace 1.in par 2.in ?

L'archive du TP contient un fichier ***do_test.sh***. Les fichiers finissant par ".sh" sont des scripts écrits en langage Bash. Vous pouvez regarder le contenu du script et tenter de comprendre comment ce langage fonctionne si vous le souhaitez.

Le script do_test.sh prend en argument un nom d'exécutable. Il regarde si, dans le dossier où l'exécutable se trouve, il y a des sous-dossiers test_in et test_out et, le cas échéant, pour chaque fichier x.in dans le dossier test_in/, il compare la sortie du programme sur x.in avec ce qui est écrit dans test_out/x.out.

Question 7. Pour exécuter un script Bash, il faut lui donner des permissions. Pour ce faire, tapez :

```
chmod 777 do_test.sh
```

Question 8. Lancez maintenant la commande suivante et vérifiez qu'elle fait bien ce qui est annoncé :

```
./do_test.sh ./carre
```

Question 9. Rajoutez quelques tests unitaires, et modifiez le programme pour qu'il renvoie une mauvaise réponse pour une entrée particulière, par exemple pour qu'il renvoie 29 pour l'entrée 10 (on simule un bug). Rajoutez un test qui permet de détecter ce bug. Relancez la commande précédente et vérifiez que le bug est bien détecté

Livre dont *vous* êtes le héros ou la héroïne

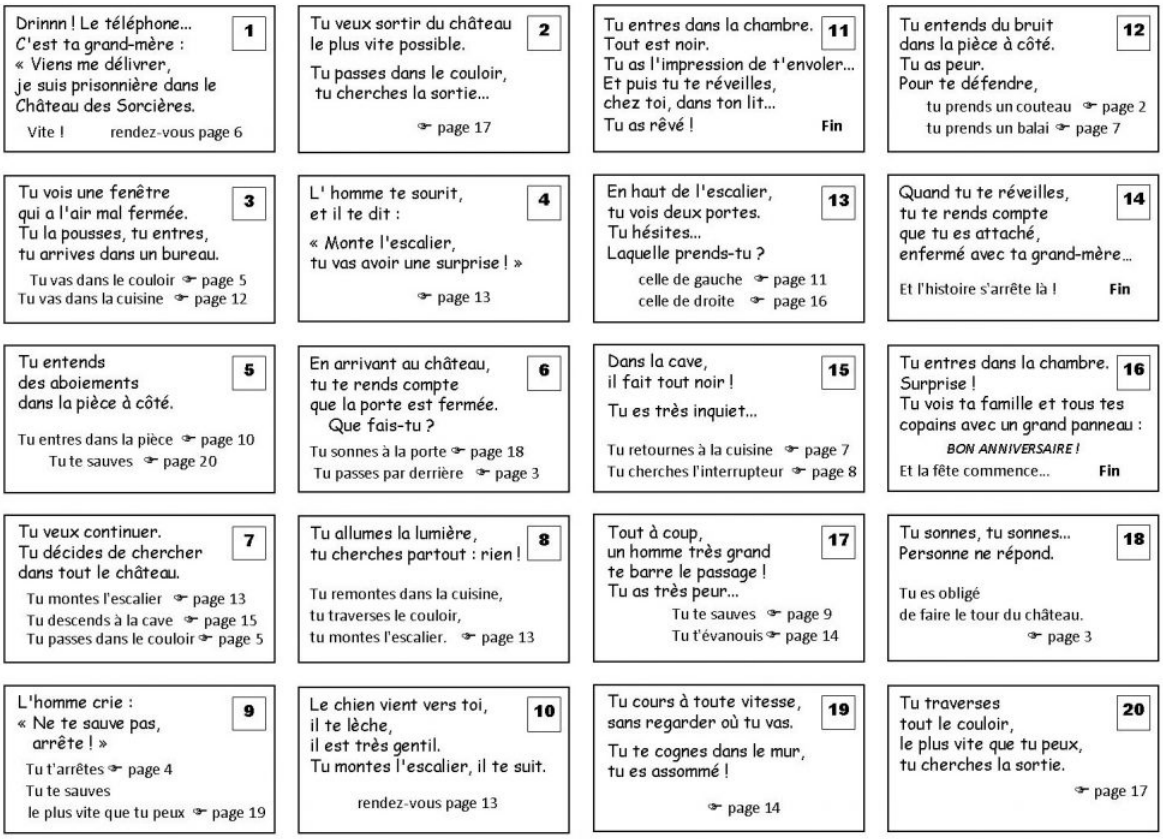

FIGURE 1 – Un LDVELH assez court

Exercice 7.

Un ***Livre dont vous êtes le héro / la héroïne***, où LDVELH, est un type de livre interactif, où le lecteur incarne le protagoniste. A chaque page, il est amené à faire un choix, et selon l'option choisie, à se rendre à une page particulière. Ainsi, un LDVELH ne se lit pas dans l'ordre mais en sautant de page en page en suivant les choix faits, en commençant à la page 1.

Le but de l'exercice est d'écrire un programme qui peut lire un LDVELH écrit dans un fichier texte sous un format bien spécifique, et en faire une version interactive, où l'utilisateur peut rentrer ses choix au clavier.

Pour représenter un LDVELH, on propose la modélisation suivante :

- Un Livre est constitué de plusieurs pages, indexées par $0, 1, \ldots N-1$, où N est le nombre de pages
- Une Page est constituée d'un texte, ainsi que de plusieurs choix, indexés par $0, 1, \ldots N-1$, où N est le nombre de choix
- Un Choix est constitué d'un texte ainsi que du numéro de la page suivante lorsque l'on suit ce choix.

Pour représenter sous format texte un LDVELH, on écrit dans un fichier texte :

- Sur la première ligne, un entier n indiquant le nombre de pages total du livre
- Pour chaque page, on écrit :
  - Sur une ligne, le nombre de choix
  - Le texte de cette page sur une ligne
  - Pour chaque choix de la page, on écrit :
    - Le texte du choix sur une ligne
    - Le numéro de la page suivante

Une page marque la fin de l'aventure si, et seulement si, elle a 0 choix.

Par exemple, les premières lignes du fichier "mamie.txt" représentant le LDVELH Fig. 1 seraient les suivantes (on indexe les pages à partir de 0) :

```
20
1
Drinnn ! Le téléphone... C'est ta grand mère: "[...]"
Vite !
5
1
Tu veux sortir du chateau [...] tu cherches la sortie...
   (cette ligne est vide)
16
2
Tu vois une fenêtre ... tu arrives dans un bureau.
Tu vas dans le couloir
4
Tu vas dans la cuisine
11
```

En C, on propose les structures suivantes pour représenter ces informations :

```c
struct CHOIX{
  char* texte;
  int page_suivante;
}

struct PAGE{
  char* texte;
  int n_choix;
  struct CHOIX** choix; //tableau de pointeurs vers des choix
}

struct LIVRE{
  int n_pages;
  struct PAGE** pages //tableau de pointeurs vers des pages
}
```

Vous pouvez utiliser typedef pour renommer ces structs si vous le souhaitez.

Question 1. Écrivez un fichier text.txt avec un exemple minimal de LDVELH (au moins 3 pages, au moins une page avec plusieurs choix)

Question 2. Écrivez une fonction `struct CHOIX* generer_choix(FILE* f)` qui, étant donné un flux de fichier f, dont on suppose que la tête de lecture est positionnée à un endroit contenant un choix sous forme textuelle, crée une structure de type `struct CHOIX` en utilisant les données lues dans le fichier, et renvoie une pointeur vers cette structure.

Question 3. Écrivez une fonction `struct PAGE* generer_page(FILE* f)` qui, étant donné un flux de fichier f, dont on suppose que la tête de lecture est positionnée à un endroit contenant une page de LDVELH sous forme textuelle, crée une structure de type `struct PAGE` en utilisant les données lues dans le fichier, et renvoie une pointeur vers cette structure.

Question 4. Écrivez une fonction `struct LIVRE* generer_livre(char* filename)` qui, étant donné un ***nom de fichier*** filename, crée une structure de type `struct LIVRE` en utilisant les données lues dans le fichier filename, et renvoie une pointeur vers cette structure.

Question 5. Écrivez une fonction `void free_livre (struct LIVRE* livre)` qui, étant donné un pointeur vers une structure struct LIVRE, libère toute la mémoire allouée à ce livre, y compris pour le texte, pour les pages. Vous pouvez écrire des fonctions auxiliaires pour vous aider.

Question 6. En utilisant les fonctions précédentes, écrivez un programme prenant comme premier argument un nom de fichier texte, dont on suppose qu'il contient un LDVELH sous format texte, et qui permet à l'utilisateur de jouer/lire le livre.
L'exécution du programme doit ressembler vaguement à :

```
guillaume@MSI:~/prof/code/cyoa$
guillaume@MSI:~/prof/code/cyoa$ ./cyoa livre.txt
Nous sommes vendredi matin. Demain, il y a DS d'informatique. La journée doit se dérouler sans accroc afin que vous
ayez le temps de réviser ce soir, de dormir, et donc que vous puissiez obtenir une bonne note au DS. Il est 10h, le
cours de physique de Mme Goutelard vient de se terminer. Que faites-vous ?

  1. Vous allez en cours d'informatique.

  2. Vous n'allez pas en cours d'informatique, il fait beau et vous préférez aller vous balader sur les quais de la
Garonne.

Votre choix: 2
Vous n'allez pas en cours d'informatique, et vous ratez des informations capitales pour réviser. Vous n'êtes pas bie
n préparé pour le DS. GAME OVER
```

Vous trouverez dans l'archive du TP un fichier "aventure.txt" écrit sous le format décrit plus haut, avec lequel vous pouvez tester votre programme une fois qu'il fonctionne sur votre exemple minimal.