



Université de Technologie de Compiègne

NF16

Rapport de TP

3 - Listes chaînées

Automne 2014

Romain PELLERIN - Brianda PIZAÑO
Groupe de TP 2
<i>10 novembre 2014</i>

Table des matières

1	Introduction	3
1.1	Résumé	3
1.2	Structuration du projet	3
1.3	Code source	3
2	Implémentation	4
2.1	Programme principal	4
2.2	Gestion d'une ludothèque	5
2.2.1	Fonction <code>ajouter_jeu</code>	5
2.2.2	Fonction ajoutée	6
2.3	Gestion des jeux	6
2.3.1	Fonctions ajoutées	6
3	Capture d'écran	8

1 Introduction

1.1 Résumé

L'objectif de ce TP est d'écrire un programme permettant la gestion d'une ludothèque composée de jeux de société. Un jeu de société est décrit par une fiche contenant les informations suivantes :

- Nom du jeu
- Genre du jeu : PLATEAU, RPG, COOPERATIF, AMBIANCE ou HASARD
- Le nombre de joueurs minimum
- Le nombre de joueurs maximum
- La durée moyenne d'une partie en minutes

1.2 Structuration du projet

Nous avons choisi de structurer notre projet en 5 fichiers pour une meilleure clarté, selon les deux entités principales que sont les **ludothèques** et les **jeux** :

- **main.c** Le fichier source contenant le programme principal
- **ludotheque.h** Le fichier d'en-tête contenant les déclarations des structures et fonctions d'une ludothèque
- **ludotheque.c** Le fichier source contenant la définition de chaque fonction d'une ludothèque
- **jeu.h** Le fichier d'en-tête contenant les déclarations des structures et fonctions d'un jeu
- **jeu.c** Le fichier source contenant la définition de chaque fonction d'un jeu

1.3 Code source

Le code source est fourni dans l'archive ZIP contenant ce rapport.

2 Implémentation

Les fonctions demandées dans le sujet ont été implémentées, avec quelques spécificités selon la compréhension du sujet. Par exemple, certaines vérifications ont été ajoutées (vérification qu'une ludothèque ou qu'un jeu n'est pas `NULL`). A l'inverse, certaines vérifications sur les entrées utilisateurs n'ont pas été faites (on ne vérifie pas que lorsqu'un entier est attendu, l'utilisateur a bien tapé un entier).

2.1 Programme principal

Le programme principal affiche à l'utilisateur un menu qui lui donne le choix parmi **6 possibilités** :

1. Créer une ludothèque
2. Afficher une ludothèque
3. Ajouter un jeu dans la ludothèque en saisissant ses caractéristiques
4. Effectuer une recherche de jeu à partir de critères saisis par l'utilisateur
5. Créer 2 ludothèques, les afficher, les fusionner puis afficher la nouvelle ludothèque
6. Quitter le programme

L'utilisateur tape au clavier le numéro de son choix. Un `switch` permet d'effectuer les actions correspondantes à l'entrée clavier.

main.c

```
// ...
afficherMenu();
while((inputUser = getchar()) != '6') {
    // ...
    switch(inputUser) {
        case '1':
            // ...
            break;
        case '2':
            // ...
            break;
        case '3':
            // ...
            break;
        case '4':
            // ...
            break;
        case '5':
            // ...
            break;
        default:
            printf("Mauvaise entrée. Veuillez taper 1, 2, 3, 4, 5 ou 6.\n");
            continue; // On recommence la boucle while
    }
}
```

```
}  
// ...  
afficherMenu();  
}  
// ...
```

Une boucle `while` permet d'afficher à nouveau le menu après être sorti du `switch`. L'invariant de boucle est `(inputUser = getchar()) != '6'`.

2.2 Gestion d'une ludothèque

La structure d'une ludothèque a été définie dans le fichier *ludothèque.h* :

ludothèque.h

```
struct ludothèque {  
    int nb_jeu;  
    t_jeu *debut;  
};
```

2.2.1 Fonction ajouter_jeu

La fonction `int ajouter_jeu(t_ludothèque* ludo, t_jeu* j);` permet d'ajouter un jeu même si celui-ci est déjà présent dans l'actuelle ludothèque, en maintenant la ludothèque triée selon le nom des jeux. **Il n'était pas précisé dans le sujet si les doublons devaient être évités pour cette fonction.** Lors de la fusion de 2 ludothèques, une vérification est faite pour ne pas ajouter deux fois un jeu présent dans les 2 ludothèques (fonction vue en TD).

Voici une autre fonction qui permet d'ajouter un jeu à une ludothèque en évitant les doublons, (cette fonction n'est pas dans notre code source) :

Fonction ajouter_jeu_no_doublon

```
int ajouter_jeu_no_doublon(t_ludothèque* ludo, t_jeu* j) {  
    if (j==NULL || ludo == NULL) return 0;  
    t_jeu* tete = ludo->debut;  
    if (tete == NULL || strcmp(j->nom,tete->nom) < 0) {  
        j->suivant = ludo->debut;  
        ludo->debut = j;  
    }  
    else {  
        while(tete->suivant != NULL && strcmp(j->nom,tete->suivant->nom) > 0) {  
            tete = tete->suivant;  
        }  
        if (tete->suivant != NULL && strcmp(j->nom,tete->suivant->nom) == 0) {  
            return 0; // Le jeu est déjà présent (même nom)  
        }  
        // Le jeu n'est pas présent, on l'ajoute  
        j->suivant = tete->suivant;  
        tete->suivant = j;  
    }  
    return 1;  
}
```

2.2.2 Fonction ajoutée

En plus des fonctions demandées dans le sujet, nous avons ajouté la fonction

ludotheque.h

```
t_ludotheque* saisir_requete(t_ludotheque *ludo);
```

pour obtenir un code mieux organisé. Elle permet de demander à l'utilisateur d'entrer les différents paramètres de sa requête. Cette fonction appelle ensuite la fonction

ludotheque.h

```
t_ludotheque* requete_jeu(t_ludotheque *ludo, genre_jeu genre, int nbJoueurs,
    int duree);
```

demandée dans le sujet du TP.

2.3 Gestion des jeux

L'énumération des genre ainsi que la structure d'un jeu ont été définis dans le fichier *jeu.h* :

jeu.h

```
typedef enum genre {
    PLATEAU, RPG, COOPERATIF, AMBIANCE, HASARD
} genre_jeu;

typedef struct jeu {
    char *nom;
    genre_jeu genre;
    int nbJoueurMin;
    int nbJoueurMax;
    int duree;
    struct jeu *suivant;
} t_jeu;
```

2.3.1 Fonctions ajoutées

Nous avons ajouté trois fonctions destinées à éviter de répéter beaucoup de code à divers endroits :

jeu.h

```
t_jeu* saisir_jeu();

char *stringFromGenre(enum genre g);

genre_jeu genreFromString(char *str);
```

La première fonction permet de demander à un utilisateur les paramètres d'un jeu, en vue de créer ce jeu (la fonction retourne un pointeur de jeu).

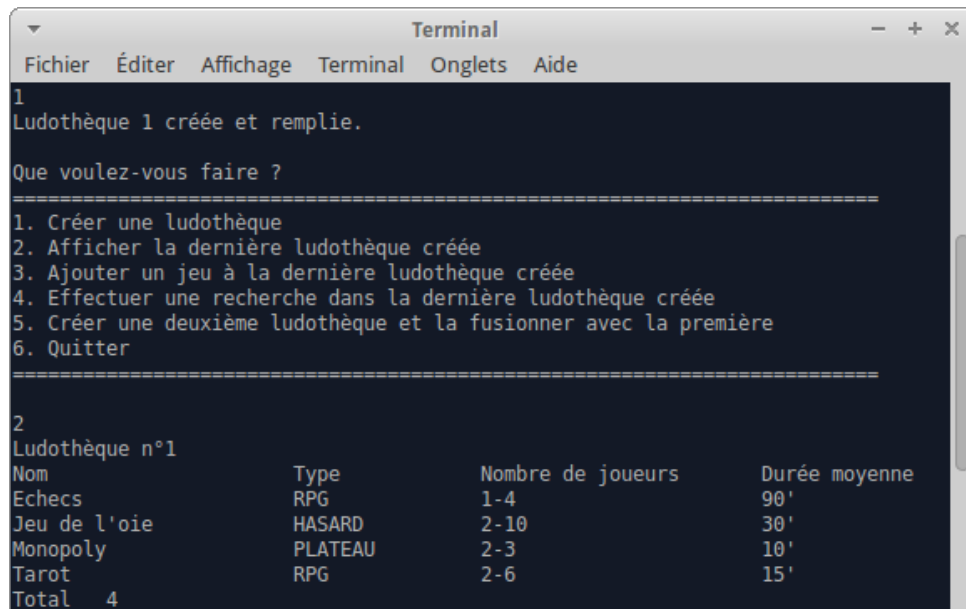
Les deux fonctions suivantes permettent de manipuler une `enum` sous forme de `string` et inversement de manière très simple, cela est plus pratique pour l'affichage sur la console.

Nous avons arbitrairement décidé pour la dernière fonction que si le paramètre `char *str` ne correspondait pas à un genre de jeu qui existe, alors nous retournerions le type `HASARD`. De

plus, nous avons utilisé la fonction `strcasemp` grâce à `#include <strings.h>` pour pouvoir comparer des `strings` sans tenir compte de la casse. Nous aurions pu utiliser la fonction `strcmp` qui, elle, tient compte de la casse.

3 Capture d'écran

Voici un aperçu de notre programme :



```
Terminal
Fichier Éditer Affichage Terminal Onglets Aide

1
Ludothèque 1 créée et remplie.

Que voulez-vous faire ?
=====
1. Créer une ludothèque
2. Afficher la dernière ludothèque créée
3. Ajouter un jeu à la dernière ludothèque créée
4. Effectuer une recherche dans la dernière ludothèque créée
5. Créer une deuxième ludothèque et la fusionner avec la première
6. Quitter
=====

2
Ludothèque n°1
Nom                Type          Nombre de joueurs  Durée moyenne
Echecs             RPG           1-4                90'
Jeu de l'oie       HASARD        2-10               30'
Monopoly            PLATEAU       2-3                10'
Tarot              RPG           2-6                15'
Total 4
```

FIGURE 3.1 – L'utilisateur a choisi 1 puis 2 dans le menu d'options