NF16 - TP3
Gestion d'une ludothèque

Automne 2014 – 17 novembre 2014

Enseignant : Mohamed Akheraz

Thomas COUTANT **GI01**
Eric GOURLAOUEN **GI01**

Sommaire

Introduction

L'objectif de ce TP était de manipuler des listes chaînées, à travers la gestion de ludothèques de jeux de société. Chaque ludothèque était une liste chaînée de jeux de sociétés ; il nous a fallu, dans le cadre de ce projet, apprendre à ajouter des objets à des listes chaînées, apprendre à les trier, apprendre à afficher tout le contenu d'une liste, ou à faire une recherche dans une liste chaînée.

Structures conçues

Dans le contexte de ce TP, l'objectif était de concevoir deux structures : des **jeux de société**, d'un côté, et des **ludothèques**. Les jeux de société sont représentées sous formes de listes chaînées ; chaque jeu d'une ludothèque est représenté par son nom, son genre (parmi une liste de genre énumérés *genre_jeu*), son nombre de joueur minimum et maximum, la durée moyenne d'une partie, et l'adresse mémoire du jeu suivant. Cette structure *struct jeu* a été renommée *t_jeu*.

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

Les ludothèques étaient représentées sous formes de structures avec à la fois le nombre de jeux dans la ludothèque, et l'adresse mémoire du premier jeu de la liste. Cette structure *struct ludotheque* a été renommée *t_ludotheque*.

```
typedef struct ludotheque {
	int nb_jeu;
	t_jeu *debut;
} t_ludotheque;
```

Fonctions du programme

creer_ludotheque

creer_ludotheque() ne prend aucun argument et rend une adresse mémoire. Si l'allocation mémoire pour obtenir un emplacement mémoire vers une ludothèque a réussi, le programme rend l'adresse de la ludothèque ; sinon, le programme rend NULL.

```
t_ludotheque* creer_ludotheque() {
	t_ludotheque *ludo = malloc(sizeof(t_ludotheque));

	if (ludo != NULL) {
		ludo->nb_jeu = 0;
		ludo->debut = NULL;
		return ludo;
	}
	return NULL; // Si la ludothèque n'a pas été correctement initialisée
}
```

creer_jeu

creer_jeu(char *nom, int nbJoueurMin, int nbJoueurMax, genre_jeu genre, int duree) rend une adresse mémoire. De la même façon que pour la création d'une ludothèque, si l'allocation mémoire ne réussit pas, il rend NULL ; sinon, il rend l'adresse mémoire du jeu créé, en y ajoutant les paramètres insérés.

```
t_jeu* creer_jeu(char *nom, int nbJoueurMin, int nbJoueurMax, genre_jeu genre, int
duree) {
	t_jeu *jeu = malloc(sizeof(t_jeu));

	if(jeu != NULL) {
		jeu->nom = nom;
		jeu->genre = genre;
		jeu->nbJoueurMin = nbJoueurMin;
		jeu->nbJoueurMax = nbJoueurMax;
		jeu->duree = duree;
		jeu->suivant = NULL;
		return jeu;
	}
	return NULL; // Si le jeu n'a pas été correctement créé
}
```

ajouter_jeu

ajouter_jeu(t_ludotheque *ludo, t_jeu *j) prend en paramètres l'adresse d'une ludothèque et celle d'un jeu ; il ajoute le jeu à la ludothèque donnée, tout en respectant l'ordre lexicographique des noms de jeux. Ainsi, au début, si la ludothèque est vide ou que le premier jeu de la liste est situé après dans l'ordre lexicographique, il ajoute le jeu en tant que tête de liste directement (et dans le deuxième cas, rajoute l'ancien premier jeu en tant que jeu suivant). Sinon, il parcourt tous les éléments de la liste tant que le jeu qu'on ajoute est situé après l'élément étudié dans l'ordre lexicographique ou qu'on arrive à la fin de la liste ; on ajoute alors le jeu à la liste.

La comparaison des noms se fait à l'aide d'une fonction triAlphabetique, qui rend un entier en fonction de la comparaison (0 si les chaînes de caractères sont identiques, 1 si la première chaîne est avant la seconde, 2 si la seconde chaîne est avant la première)

```
int ajouter_jeu(t_ludotheque *ludo, t_jeu *j) {
	/* Ajoute un jeu à une ludothèque */
	if (ludo->debut == NULL) {
		// premier cas possible : il n'y a aucun jeu dans la ludothèque
		ludo->debut = j;
		ludo->nb_jeu++;
		return 1;
	} else if (triAlphabetique(j->nom,(ludo->debut)->nom) == 1) {
		// second cas possible : le premier jeu de la liste est situé après le
nouveau dans l'ordre lexicographique
		j->suivant = ludo->debut;
		ludo->debut = j;
		ludo->nb_jeu++;
		return 1;
	} else {
		t_jeu *jTemp = ludo->debut;

		while ((jTemp->suivant != NULL) && (triAlphabetique(j->nom,
(jTemp->suivant)->nom) == 2)) {
			jTemp = jTemp->suivant;

		}
		if (jTemp->suivant == NULL) {
			jTemp->suivant = j;
			j->suivant=NULL;
			ludo->nb_jeu++;
			return 1;
		} else {
			j->suivant = jTemp->suivant;
			jTemp->suivant = j;
			ludo->nb_jeu++;
			return 1;
		}
```

```
		return 0;
	}
}
```

retirer_jeu

retirer_jeu(t_ludotheque *ludo, char *nom) est une fonction qui retire la première occurrence dans la ludotheque ludo du jeu ayant comme nom la chaîne de caractère donnée. Elle passe en revue tous les éléments de la liste jusqu'à arriver à la fin de la liste ; s'il y a bien un élément qui a été supprimée, elle retourne 1. Sinon, elle retourne 0.

```
int retirer_jeu(t_ludotheque *ludo, char *nom) {
	t_jeu *iter, *temp;
	iter = ludo->debut;
	if (iter != NULL && iter->nom == nom){
		temp = iter;
		ludo->debut = temp->suivant;
		ludo->nb_jeu--;
		free(temp);
		return 1;
	}
	while(iter != NULL && iter->suivant != NULL && (iter->suivant)->nom != nom)
		iter = iter->suivant;
	if ((iter->suivant)->nom == nom) {
		temp = iter->suivant;
		iter->suivant = temp->suivant;
		ludo->nb_jeu--;
		free(temp);
		return 1;
	} else
		return 0;
}
```

supprimer_ludotheque

supprimer_ludotheque(t_ludotheque *ludo) prend en paramètre une ludothèque, et en supprime tous les éléments jusqu'à ce qu'il n'y ait plus d'éléments dans la ludothèque ; elle vide ensuite l'espace mémoire occupé par la ludothèque.

```
void supprimer_ludotheque(t_ludotheque *ludo) {
	t_jeu *iter = ludo->debut;
	t_jeu *iterTemp;
	if (iter != NULL) {
		while(iter->suivant != NULL) {
			iterTemp = iter;
			iter = iterTemp->suivant;
```

```
                        free(iterTemp);
                }
                free(iter);
                free(ludo);
        } else
                free(ludo);
                return;
}
```

affiche_ludotheque

affiche_ludotheque(t_ludotheque *ludo) est une fonction qui prend en paramètre une ludothèque, et en affiche les éléments un par un. Si la ludothèque est vide, la fonction affiche juste “Aucun jeu dans la ludothèque.” avant de s’arrêter. Sinon, elle affiche d’abord la première ligne (d’annonce des colonnes), avant d’afficher chaque élément. Le nombre de caractères correspondant à chaque composante de chaque jeu est prévu pour ne pas dépasser le nombre de caractères prévus.

```
void affiche_ludotheque(t_ludotheque *ludo) {
        int i;
        char *tabTypes[5]={"PLATEAU", "RPG", "COOPERATIF", "AMBIANCE", "HASARD"};

        if(ludo->debut == NULL) {
                printf("Aucun jeu dans la ludotheque.\n");
                return;
        }

        t_jeu *iter = ludo->debut;
        printf("Nom             Type         Nombre de joueurs  Duree Moyenne    \n");

        for(i=0; i<ludo->nb_jeu; i++) {
                printf("%-16s%-13s%8d-%-10d%-17d\n", iter->nom, tabTypes[iter->genre],
iter->nbJoueurMin, iter->nbJoueurMax, iter->duree);
                iter = iter->suivant;
        }

        printf("Total\t\t%d\n",ludo->nb_jeu);
}
```

requete_jeu

requete_jeu(t_ludotheque *ludo, genre_jeu genre, int nbJoueurs, int duree) prend en paramètres une ludothèque, un genre, un nombre de joueurs et une durée et rend une ludothèque qui comprend des jeux de la ludothèque donnée qui correspondent aux arguments donnés. Les arguments du nombre de joueurs et de la durée de la partie peuvent être ignorés en mettant ces arguments à -1.

```
t_ludotheque* requete_jeu(t_ludotheque *ludo, genre_jeu genre, int nbJoueurs, int
duree) {
	t_ludotheque* nLudo;
	t_jeu *jTemp, *iter;
	nLudo = malloc(sizeof(t_ludotheque));
	nLudo->nb_jeu = 0;
	nLudo->debut=NULL;
	int i;

	for (i=0; i<(ludo->nb_jeu); i++) {
		iter = (i==0) ? ludo->debut : iter->suivant;

		if (nbJoueurs == -1 && duree == -1) {
			if(genre == iter->genre)

ajouter_jeu(nLudo,creer_jeu(iter->nom,iter->nbJoueurMin,iter->nbJoueurMax,iter->genre,
iter->duree));
		} else if (nbJoueurs == -1) {
			if((genre == iter->genre) && ((duree > 0.90*iter->duree) && (duree
< 1.10*iter->duree)))

ajouter_jeu(nLudo,creer_jeu(iter->nom,iter->nbJoueurMin,iter->nbJoueurMax,iter->genre,
iter->duree));
		} else if (duree == -1) {
			if((genre == iter->genre) && ((nbJoueurs > iter->nbJoueurMin) &&
(nbJoueurs < iter->nbJoueurMax)))

ajouter_jeu(nLudo,creer_jeu(iter->nom,iter->nbJoueurMin,iter->nbJoueurMax,iter->genre,
iter->duree));
		} else {
			if((genre == iter->genre) && ((nbJoueurs > iter->nbJoueurMin) &&
(nbJoueurs < iter->nbJoueurMax)) && ((duree > 0.90*iter->duree) && (duree <
1.10*iter->duree)))

ajouter_jeu(nLudo,creer_jeu(iter->nom,iter->nbJoueurMin,iter->nbJoueurMax,iter->genre,
iter->duree));
		}
	}
	return nLudo;
}
```

fusion

fusion(t_ludotheque *ludo1, t_ludotheque *ludo2) est une fonction qui prend en argument deux ludothèques, et qui rend l'adresse d'une nouvelle ludothèque qui fusionne la première et la seconde ludothèque. La fonction crée une nouvelle ludothèque, intègre tous les éléments de la première ludothèque à la nouvelle, puis intègre tous les éléments de la seconde ludothèque à la nouvelle ludothèque, mais seulement si ils ne sont pas déjà dans la ludothèque (chose vérifiée avec la fonction doublons, présente sur le fichier tp3.c, qui rend 1 si le jeu est déjà présent et 0 sinon).

```
t_ludotheque* fusion(t_ludotheque *ludo1, t_ludotheque *ludo2)
{
	t_ludotheque *ludo=creer_ludotheque();
	t_jeu *jeu, *temp=ludo1->debut;
	while(temp!=NULL)
	{

ajouter_jeu(ludo,creer_jeu(temp->nom,temp->nbJoueurMin,temp->nbJoueurMax,temp->genre,t
emp->duree));
		temp=temp->suivant;
	}
	//On passe aux jeux de la 2e ludothèque
	temp=ludo2->debut;
	while(temp!=NULL)
	{

jeu=creer_jeu(temp->nom,temp->nbJoueurMin,temp->nbJoueurMax,temp->genre,temp->duree);
		if(doublons(jeu,ludo) == 1)
			temp=temp->suivant;
		else{
		  ajouter_jeu(ludo,jeu);
		  temp=temp->suivant;
		}
	}
	return ludo;
}
```

Programme

Nous avons un programme qui tourne autour de 2 ludothèques ; l'utilisateur peut interagir avec la première ou la 2e ludothèque. Ceci permet à l'utilisateur de tester le processus de fusion. L'utilisateur doit creer une ludotheque avant de l'utiliser.

Le programme est une boucle qui pose encore et encore la même question à l'utilisateur, pour savoir ce qu'il veut faire. Il y a 6 options differentes : Créer une ludothèque, l'afficher, ajouter un jeu à une ludothèque, effectuer une recherche de jeu, faire une fusion des 2 ludothèques créées, et quitter le programme.

Création de ludothèque

Lorsqu'on choisit de créer une ludothèque, le programme demande quelle ludothèque on veut créer (si on veut créer la ludothèque 1 ou la ludothèque 2). A la ludothèque sélectionnée, on attribue une nouvelle adresse mémoire, générée par creer_ludotheque. On enregistre également qu'on a créé la ludothèque voulue ; ainsi, si une autre fonction veut se servir d'une ludothèque alors qu'elle n'a pas été créée, elle aura un message d'erreur.

Affichage de ludothèque

Lorsqu'on choisit d'afficher une ludothèque, le programme demande quelle ludothèque on veut afficher ; et si la ludothèque existe, elle appelle la fonction affiche_ludotheque.

Ajout de jeu

Quand on veut ajouter un jeu à une ludothèque, le programme demande quelle ludothèque ; il demande ensuite à l'utilisateur d'entrer les paramètres du jeu. Notamment, si l'utilisateur entre un nombre de joueurs maximum inférieur au nombre de joueurs minimum, il lui redemandera. Il ajoute ensuite ce jeu dans la ludothèque voulue à l'aide de la fonction ajouter_jeu.

Recherche d'éléments

Quand on veut lancer une recherche de jeux à l'aide d'un éléments, le programme demande à l'utilisateur de rentrer des paramètres correspondant au jeu recherché et la ludothèque de recherche. Il affiche ensuite la ludothèque récupérée grâce à la fonction requete_jeu.

Fusion de ludothèques

Quand l'utilisateur veut lancer une fusion de ludothèques, le programme vérifie d'abord que les 2 ludothèques ont été initialisées. Si elles ont bien été initialisées, le programme affiche les 2 ludothèques initiales, avant d'afficher la ludothèque créée.

Suppression d'élément

Le chargé de TP lors de la démonstration nous a demandé de rajouter une fonction de suppression d'élément dans une ludothèque. Pour cela, nous avons rajouté un cas dans le menu où l'utilisateur peut supprimer un élément d'une ludothèque en indiquant la ludothèque et le nom du jeu. Le programme affiche ensuite à l'utilisateur si la suppression a réussi ou a échoué.

Difficultés rencontrées

Nous avons rencontré des difficultés avec la fonction de fusion au début. En effet, ce que faisait la fonction de fusion au début était de simplement rajouter les jeux à d'autres ludothèques, en coupant ainsi les liens des listes chaînées de ludothèques. Ainsi, lorsqu'une fusion était réalisée, les liens entre les éléments des autres ludothèques étaient coupés.

Pour pallier à ce problème, nous avons changé la fonction de façon à ce qu'au lieu qu'au lieu d'ajouter simplement l'élément à la nouvelle ludothèque, la fonction créait un autre élément qui avait les mêmes caractéristiques, et l'ajoute ensuite à la ludothèque de fusion.

Ce problème s'est également répercuté lors de la démonstration car la fonction de recherche de jeu avait le même problème lors de sa création de ludothèque, et nous n'avions pas résolu le problème dans la fonction requete_jeu. C'est à présent fait dans le code.

Conclusion

Ce projet nous a permis d'améliorer nos compétences en manipulation de listes chaînées et de structures de données. Il nous a demandé d'être rigoureux dans la conception de nos algorithmes pour ne pas mélanger les types d'objets que nous manipulions entre eux.

Une amélioration possible que nous aurions pu réaliser serait une gestion des ludothèques ; on aurait pu gérer des listes chaînées de ludothèques, pour que l'utilisateur du programme puisse manipuler beaucoup plus de ludothèques. Cependant, au vu du cahier des charges du projet et du temps que ceci aurait pris, nous avons préféré nous contenter de la gestion de 2 ludothèques différentes et nous concentrer sur la création des fonctions.