Maxime Churin Robin Bria

NF16 - TP 3 – Listes chaînées

Ce premier TP noté de NF16 nous demandait d'implémenter un programme pour la gestion d'une bibliothèque. Après avoir réalisé une démonstration à notre chargé de TD nous allons ici justifier nos choix de programmation pour les différentes structures et fonctions. Enfin nous détailleront les complexités de toutes nos fonctions.

Fonction 9 : *rechercherLivres()*

Pour mettre en place cette fonction de recherche sur un critère de titre dans toute notre bibliothèque nous avions deux possibilités. La première était de créer des structures spécifiques et la seconde d'adapter les structures déjà définies. Nous avons choisis de simplement modifier la structure Livre en lui ajoutant le champ ***char*** *theme_rayon[MAX]*. Cela nous a paru plus pertinent car nous avions déjà toutes les fonctions implémentées pour gérer cette structure comme la création, l'initialisation, la suppression (libération de mémoire)... Nous les avons parfois adapté pour prendre en compte ce nouveau champ. De plus ce nouveau champ nous a permis de faciliter l'affichage final en particulier pour la colonne titre du rayon.

Pour implémenter cette recherche nous avons créé un rayon où nous avons rangé les livres concernés par cette recherche. Ceci au travers du parcours des rayons de la bibliothèque et des livres de chaque rayon Ces titres sont comparés au critère de titre et, en cas de succès, ajoutés au rayon grâce à la modification de la fonction *ajouterLivre()*. Elle bloque maintenant l'ajout du livre si le couple titre et thème de rayon est déjà présent dans le rayon.

Enfin pour l'affichage du résultat de cette recherche nous aurions pu modifier la fonction *afficherRayon()* et afficher pour chaque livre le nom du thème du rayon mais cela ne nous a pas paru pertinent. En effet cette affichage ne se justifiait que pour la fonction *rechercherLivres()* alors qu'à l'affichage d'un quelconque rayon de la bibliothèque nous aurions eu l'information redondante du thème du rayon car tous les livres appartiennent à ce même rayon.

Comme vu lors de notre démonstration nous avions omis de triée la liste sur le titre du livre c'est maintenant quelque chose que nous faisons.

Fonction 10 : *traiterListeEmprunts()*

Pour mettre en place cette fonction de traitement d'une liste d'emprunts nous avons réutilisé de nombreuses fonctions préalablement implémentées. C'est exactement la même méthode qu'avec la fonction *rechercherLivres()* à savoir un rayon où nous avons rangé les livres concernés par cette liste d'emprunts. Le tout à l'intérieur d'une boucle *do{}while()* pour gérer la poursuite ou non d'emprunt.

Enfin pour l'affichage du résultat de cette liste d'emprunts nous n'avons pas utiliser *afficherRayon()* mais seulement trois champs : titre du livre, thème du rayon et sa disponibilité.

Listes des fonctions supplémentaires et raison de ces choix :

*T_Livre *initialisationLivre()* : il nous a paru plus propre de créer une fonction pour demander à l'utilisateur les différents champs du livre que de le faire directement dans le main.

*int rechercheLivre(T_Rayon *rayon, char* titre), T_Rayon *rechercheRayon(T_Biblio *biblio, char* theme_rayon), int rechercheBiblio(T_Biblio *biblio)* : pour sécuriser les entrées dans notre code nous avons créé 3 fonctions de recherche soit une pour chaque structure. Elles nous permettent de vérifier l'existence ou non de l'objet lors d'une saisie de l'utilisateur par exemple.

*void lire(char *chaine, int longueur)* : il s'agit d'un scanf plus sécurisé. On utilise dans cette fonction un *fgets()* pour lire la chaîne, puis on recherche le retour chariot que l'on remplace par le caractère de fin de chaine. Cette fonction nous permet de redemander la saisie par l'utilisateur tant que la chaine lu ne comporte que des caractères de type retour chariot et/ou espace.

int testEntree(char choix)* : nous vérifions simplement que le premier caractère de la chaîne passée en paramètre est bien un chiffre. Nous l'utilisons pour sécuriser le choix de notre menu.

Complexité de nos fonctions :

creerLivre() : cette fonction ne contient pas de boucle mais elle réalise 4 appels de la fonction *strncpy()* qui une fonction bornée par une constante réelle (longeur maximale de la chaîne) donc en O(1). Elle fait aussi appel à la fonction *malloc()* pour l'allocation mémoire qui est aussi bornée par une constante réelle (le nombre total d'espace mémoire). Les autres opérations sont aussi de simples instructions en O(1). Finalement la fonction *creerLivre()* est en **O(1).**

creerRayon() : cette fonction est similaire à la fonction *creerLivre().* En effet elle fait un appel à la fonction *strncpy()* et à la fonction *malloc().* Elle est donc en **O(1)**.

creerBiblio() : cette fonction est similaire à la fonction *creerLivre().* En effet elle fait un appel à la fonction *strncpy()* et à la fonction *malloc().* Elle est donc en **O(1)**.

ajouterLivre() : cette fonction commence par des opérations élémentaires d'affectations en O(1). Puis elle réalise une boucle *while{}* qui va comparer tous les éléments de la liste chainée des livres du rayon à un livre passé en paramètre. Dans le pire des cas, cette boucle s'arrête lorsque que l'on est au bout de la liste chainée. De plus à chaque boucle, on a une comparaison par *strcpm()* sur les titres des livres afin de pouvoir faire l'ajout en gardant un ordre alphabétique. La fonction *strcmp()* est en O(1) car elle est bornée par la taille maximale de notre titre. Ainsi, cette boucle est en O(nombre livres du rayon). Ensuite la fonction réalise différentes étapes afin d'ajouter ce livre dont un appel de *strcpm()* de complexité O(1). Au final la fonction *ajouterLivre()* est en **O(nombre de livres du rayon)**.

ajouterRayon() : cette fonction est similaire à la fonction *ajouterLivre().* En effet elle contient une boucle *while{}* qui va comparer tous les éléments de la liste chainée des rayons de la bibliothèque à

un rayon passé en paramètre. La fonction *ajouterRayon()* est en **O(nombre de rayons de la bibliothèque)**.

afficherBiblio() : cette fonction a pour but d'afficher tous les rayons de la bibliothèque. En plus d'opérations élémentaires en O(1), elle contient une boucle *while{}* qui va parcourir tous les éléments de la liste chainée jusqu'à la fin. Cette boucle fait appel à un *printf()* de complexité O(1) car elle est bornée par la constante réelle de la taille maximale de la chaîne. Elle est donc en **O(nombre de rayons de la bibliothèque)**.

afficherRayon() : elle fonctionne de manière similaire à la fonction *afficherbiblio()*. Elle est donc en **O(nombre de livres du rayon)**.

emprunterLivre() : cette fonction a pour but de trouver un livre et de changer sa disponibilité. On utilise une boucle *while{}* identique à celle de la fonction *ajouterLivre()*. La suite étant de simple opérations élémentaires en O(1), la complexité de cette fonction est en **O(nombre de livres du rayon).**

supprimerLivre() : pour supprimer un livre on utilise la fonction *free()* qui va chercher en mémoire l'espace où est stocké le livre pour le libérer. Il s'agit donc d'une boucle en O(1) car bornée par la constante du maximum d'éléments dans la mémoire. De plus, on utilise une boucle *while{}* identique à celle de la fonction *ajouterLivre()* de complexité O(nombre de livres du rayon). La fonction est donc en **O(nombre de livres du rayon)**.

supprimerRayon() : on procède comme pour la fonction *supprimerLivre().* En plus d'opérations élémentaires en O(1), on utilise une boucle *while{}* identique à celle de la fonction *ajouterLivre()* de complexité O(nombre de rayons de la bibliothèque) pour rechercher le rayon. Puis on supprime tous les livres du rayon par une boucle *while{}* de parcours des livres du rayon appelant la fonction *supprimerLivre().* Cependant, le rayon va passer à chaque fois sa tête de liste, la boucle de recherche va donc sortir pour le premier élément. On se retrouve avec une fonction *supprimerLivre()* en O(1). Enfin, après avoir supprimé tous les livres du rayon, la fonction supprime le rayon via un *free()* en O(1). La fonction a donc une complexité en **O(max(nombre de rayons de la bibliothèque, nombre de livres du rayon)).**

rechercherLivres() : ici on va parcourir toute la bibliothèque (chaque rayon avec ses propres livres). Pour cela, en plus d'opérations élémentaires en O(1), on utilise deux boucles *while{}* de parcours en O(nombre de rayons de la bibliothèque) et O(nombre de livres du rayon) qu'on majore par le maximum de livres trouvés dans un rayon. Puis on réalise un test avec la fonction *strncmp()* et *strlen*() en O(1) car bornée par la constante du maximum de la chaîne. En cas de succès on fait appel au fonction *creerLivre()* et *ajouterLivre()* de complexité O(1) et O(nombre de livres du rayon). Pour finir on réalise un affichage de notre rayon de complexité O(nombre de livres du rayon). Cette dernière est linéaire par rapport à la boucle précédente d'ordre 2. Au final la complexité de cette fonction est en **O(nombre de rayons de la bibliothèque*nombre de livres du rayon).**

traiterListeEmprunts() : on réalise n emprunts. La boucle pour chaque itération/emprunt fait appel, en plus d'opérations élémentaires en O(1), aux fonctions *lire(), fflush(), rechercheRayon(), rechercheLivre(), creerLivre(), emprunterLivre()* et *ajouterLivre()*. Cependant *creerLivre()* à un coût constant d'où sa complexité en O(1). On prendra finalement pour chaque itération de la boucle la complexité max(k, m, nombre total de rayons de la bibliothèque, nombre maximum de livres possible dans un rayon). Pour finir on réalise un affichage de notre rayon en O(nombre de livres du rayon). Cette dernière est linéaire par rapport à notre boucle d'emprunt d'ordre 2. Au final la complexité de cette fonction est en **O(n*max(max(k1,k2), m, nombre total de rayons de la bibliothèque, nombre maximum de livres possible dans un rayon)).**

lire() : cette fonction fait appel à une boucle *do{}while{}* qui s'exécute tant la chaine lu ne comporte que des caractères de type retour chariot et/ou espace, sa complexité est donc en O(k) avec k le nombre de saisie successives jusqu'à arriver à une entrée valide. De plus à chaque itération de cette boucle on réalise un appel de *fgets()* et *strchr()* qui sont de complexité O(1) car majoré par la constante du maximum de la taille de la chaîne. Pour finir on fait un appel de *fflsush()* de complexité O(m), donc la fonction est en **O(max(k,m)).**

initialisationLivre() : ici, en plus d'opérations élémentaires en O(1) dont le *printf()* (cout constant) qui est défini par le programmeur, on fait 3 appels à la fonction *lire()* et 3 appels à la fonction *fflush()* de complexité O(m) avec m le nombre de caractère restant dans le buffer à la fin de la lecture. On a une complexité finale en **O(max(k,m)).**

rechercheLivre() : cette fonction est similaire à la fonction *ajouterLivre()* sa complexité est donc en **O(nombre de livres du rayon)**.

rechercheRayon () : cette fonction est similaire à la fonction *ajouterLivre()* sa complexité est donc en **O(nombre de rayons de la bibliothèque)**.

rechercheBiblio() : cette fonction est un test qui ne comporte que des opérations élémentaires, sa complexité est donc en **O(1)**.

testEntree() : cette fonction est un test qui ne comporte que des opérations élémentaires, sa complexité est donc en **O(1)**.

Ce TP nous a été très formateur. Nous avons appris une réelle méthode de travail pour la réalisation de ce programme. Nous l'avons réalisé étape par étape, en commençant par une version brut du code mais fonctionnel pour arriver à un code épurer avec réutilisation et adaptation de nos fonctions de départ. Enfin nous avons décidé de procéder à de nombreuses sécurisation de notre code d'où la présence de ces quelques fonctions supplémentaires.