AUTEUR DE LA DOCUMENTATION DU CODE : VIGNON Ugo
AUTEUR DE CE COMPTE RENDU : VIGNON Ugo

Projet de Structures de données : Gestion d'une Ludothèque

Explications sur la structure des fichiers :

Fichier .txt :

- fichier adherent.txt :

  Le nombre sur la première ligne du fichier correspond au nombre total d'adhérents que possède le fichier. Sur la ligne qui suit est ensuite renseigné l'id de l'adhérent, la civilité de l'adhérent, le nom de l'adhérent. Il y a ensuite un retour à la ligne car notre programme accepte les noms de famille composés, il est donc utilisé un fgets pour le récupérer et pour arrêter un fgets il faut retourner à la ligne. Sur la ligne suivante est renseigné le prénom. Même chose que pour le nom, le programme autorise les prénoms composés et est récupéré via un fgets nous avons donc effectué un retour à la ligne. Sur la ligne qui suit est renseigné la date de l'inscription de l'adhérent : jour/mois/année. Plus généralement 3 lignes correspondent à un adhérent.

- fichier jeux.txt :

  est renseigné en premier l'id du jeu, le nom du jeu. Notre programme autorise les noms de jeu composés, le nom est donc récupéré avec un fgets il est donc effectué un retour à la ligne pour arrêter le fgets. Sur la ligne suivante est renseigné le type du jeu ainsi que le nombre d'exemplaires total.

- fichier emprunts.txt :

  Le nombre sur la première ligne correspond au nombre total d'emprunts que possède le fichier. Ensuite sur chaque ligne est renseigné les informations d'un emprunt, dans l'ordre : id de l'emprunt, id de l'adhérent, id du jeu et date de l'emprunt (jour/mois/année)

- fichier reservations.txt :
  Chaque ligne du fichier représente une réservation différente. Est renseigné dans l'ordre : id de la réservation, id de l'adhérent, id du jeu et date de la réservation (jour/mois/année).

fichiers binaire :

- fichier adherent.bin :
  Sur la première ligne est renseigné le nombre total d'adhérents du fichier suivi de tous les adhérents écrits en binaire.

- fichier jeux.bin :
  Le fichier contient des jeux écrits en binaire.

- fichier emprunts.bin :
  Sur la première ligne est renseigné le nombre total d'emprunts du fichier suivi de tous les emprunts écrits en binaire.

- fichier reservations.bin :
  Le fichier contient des réservations écrites en binaire.

Explication sur les structures mises en place du .h:

- structure Date :

```
typedef struct{
        int jour;
        int mois;
        int annee;
}Date;
```

Pour structurer le programme j'ai opté de stocker toutes les dates utilisées dans un type "Date". Chaque variable initialisé avec le type "Date" contiendra donc un jour, un mois et une année.

- structure Jeux :

```
typedef struct{
        char idJeux[6];
        char nom[32];
        char type[13];
        int nb;
}Jeux;
```

La structure Jeux sert à contenir toutes les informations relatives à un jeu. Chaque variable initialisé avec le type "Jeux" contiendra donc un id de jeu, un nom, un type et un nombre d'exemplaires. Un id de jeu est représenté sur 5 cases, la 6ème case contient le '\0'. Le nom du jeu est représenté sur 30 cases maximum + 2 cases pour le '\n' et le '\0'. Le type du jeu est représenté sur 12 cases maximum (construction étant le type avec le plus de caractères) + 1 case pour le '\0'.

- structure Adherent :

```
typedef struct{
        char idAdherent[6];
        char civilite[4];
        char nom[32];
        char prenom[32];
        Date dateInscri;
}Adherent;
```

La structure Adherent sert à contenir toutes les informations relatives à un adhérent. Chaque variable initialisé avec le type “Adherent ” contiendra donc un id d’adhérent, une civilité, un nom de famille, un prénom et une date d'inscription initialisée avec le type “Date”. Un id d’adhérent est représenté sur 5 cases + 1 case pour le ‘\0’. La civilité est représentée sur 3 cases (Mme étant le plus long) + 1 case pour le ‘\0’. Le nom de famille de l’adhérent est représenté sur 30 cases maximum + 2 cases pour le ‘\n’ et le ‘\0’. Le prénom de l’adhérent est représenté sur 30 cases maximum + 2 cases pour le ‘\n’ et le ‘\0.

- structure Emprunt :

```
typedef struct{
        char idEmprunt[6];
        char idAdherent[6];
        char idJeux[6];
        Date dateEmp;
}Emprunt;
```

La structure Emprunt sert à contenir toutes les informations relatives à un emprunt. Chaque variable initialisé avec le type “Emprunt” contiendra donc un id d’emprunt, un id d’adhérent, un id de jeu et une date d’emprunt initialisée avec le type “Date”. Un id d’emprunt est représenté sur 5 cases + 1 pour le ‘\0’. Un id d’adhérent est représenté sur 5 cases + 1 pour le ‘\0’. Un id de jeu est représenté sur 5 cases + 1 pour le ‘\0’.

- structure de la liste des réservations :
    - structure Maillon :

```
typedef struct maill{
        char idResa[6];
        char idAdherent[6];
        char idJeux[6];
        Date dateRes;
        struct maill*suiv;
}Maillon;
```

La structure Maillon sert à contenir toutes les informations relatives à une réservation. Chaque Maillon contiendra donc un id de réservation, un id d'adhérent, un id de jeu, une date de réservation initialisée avec le type "Date". et un pointeur 'suiv' qui pointera sur un autre Maillon ou sur NULL s'il est le dernier.

- définition d'une liste :

```
typedef Maillon*Reservation;
```

Cette ligne définit une liste qui a pour nom Reservation étant un pointeur sur le type Maillon.

Explication sur les structures mises en place du .c:

Globalement nous avons essayé de montrer un maximum de compétences en variant les outils utilisés : tableau,tableaux de pointeurs,liste.
Nous avons également varié l'écriture de nos fichiers txt par exemple pour le fichier des emprunts et des adhérents il y a un première ligne indiquant le nombre total d'adhérent et d'emprunt dans leur fichier respectif. Cela permet de varier nos fonctions de chargement ainsi que nos fonctions relatives aux questions du sujet.

- tableau de pointeurs sur structure :

Nous avons décidé d'utiliser un tableau de pointeurs pour le tableau des jeux ainsi que pour le tableau des adhérents. Pour ce qui est de l'initialisation, ils sont initialisés de manière à être chargés dynamiquement :

- Jeux **tabJeux
- Adherent **tabAdherents

- tableau de structure :

Utilisation d'un tableau simple pour le tableau des emprunts. Pour ce qui est de l'initialisation, il est initialisé de manière à être chargé dynamiquement :

- Emprunt *tabEmprunts

- liste :

Utilisation d'une liste pour les réservations :

- Reservation l

Liste commentée des fonctionnalités proposés :

Toutes les fonctionnalités fonctionnent

1) chargement des quatre fichiers en mémoire (DEBUSSY Lucas) :

   le chargements est fait à partir de 4 fichiers .txt

   - chargement du tableau des emprunts dynamiquement (Lucas)
   - chargement des 2 tableaux de pointeurs (Jeux et Adherents) dynamiquement (Ugo et Lucas)
   - chargement de la liste des réservations (Lucas)

2) affichage de la liste des jeux disponibles triée par type de Jeux, puis par ordre alphabétique de leur nom (DEBUSSY Lucas et VIGNON Ugo) :

   Pour cette fonctionnalité, il a tout d'abord été important de comprendre que d'appliquer 1 tri à la suite d'un autre tri n'aura que pour seul effet de n'appliquer que le dernier tri effectué. En effet, le 2ème tri ne ferait que simplement un autre tri depuis le début du tableau.

   Réflexion utilisée → trier une première fois le tableau en fonction des types de jeu pour obtenir plusieurs parties qui devront être pas la suite triés selon l'ordre alphabétique. Par la suite, nous allons utiliser 2 variables "index" nous permettant de stocker à partir de quelle case devra commencer le second tri et à partir de quelle case il devra se terminer. De ce fait, dans le parcours du tableau, dès que le type va changer l'index2 prendra l'index du tableau. Il a fallu ensuite créer une fonction tri qui prendra en plus de la taille logique ce fameux index. Lors de l'appel la fonction de tri prend le tableau ensuite, à la place de la taille logique est passé l'index qui a été initialisé à zéro et enfin en dernier paramètre l'index2. De ce fait le tri est effectué en commençant à la case index et se terminant à la case index2 (bouclage en étant inférieur à index). En sortant du second tri l'index est affecté à la case précédemment initialisé à l'index2 et ainsi de suite le bouclage continu dans la fonction principale.

3) <u>affichage de la liste des emprunts en cours en mentionnant : le nom du jeu, l'identité de l'emprunteur ainsi que la date de l'emprunt</u> (DEPORTE Jeremy) :

Pour cette fonctionnalité, afficher le tableau des emprunts n'était pas suffisant car les convertions suivantes devaient être effectués :

id de l'adhérent → nom de l'emprunteur (adhérent)
id du jeu→ nom du jeu

Réflexion utilisée → Utilisation d'une fonction qui retourne la position pour un id de jeu recherché. Utilisation d'une fonction qui retourne un adhérent pour un id d'adhérent recherché (Pourquoi elle ne retourne pas une position aussi ? car cette fonction est déjà utilisée dans une autre fonctionnalité et a été ré-utilisée). De ce fait en parcourant le tableau, est effectué à chaque fois les deux recherches en question, permettant d'afficher les informations exigées.

4) <u>affichage de la liste des réservations pour un jeu donné</u> (DEPORTE Jeremy et VIGNON Ugo) :

Pour cette fonctionnalité il a fallu afficher les informations de la réservation que lorsqu'elle concordait avec l'id du jeu voulu.

Réflexion utilisée → Tout d'abord est effectuée la saisie du nom du jeu voulu, ainsi qu'une fonction de recherche retournant la position du jeu selon l'id du jeu qui lui est donné en paramètre. Ensuite est utilisée une fonction récursive pour parcourir la liste. Dans le parcours de la liste, est comparé l'id du jeu saisie au préalable avec l'id du jeu du Maillon. Si l'id du jeu est celui qui est recherché, l'affichage de la réservation en question est déclenché.

5) <u>saisie et enregistrement d'un nouvel emprunt ou d'une réservation</u> (VIGNON Ugo) :

- création de l'adhérent si nouveau (avec contrôle de saisie) :

Pour cette fonctionnalité il a fallu enregistré l'adhérent si le nom ainsi que le prénom renseigné ne correspondait à aucun adhérent du tableau (C'est a première vu bizarre car des personnes peuvent posséder un nom et prénom identiques, seulement dans le cadre du projet il n'y avait pas assez d'éléments permettant de différencier des adhérents entre eux). Dans le cas où l'adhérent est connu pour son nom et prénom dans les adhérents, la création n'est pas déclenchée. Pour ce qui est de la création de l'adhérent, l'id adhérent qui lui est assigné c'est simplement l'id du dernier adhérent qui se trouve dans le

tableau + 1 à la partie des chiffres. L'id des adhérents respectent cette convention d'écriture : AXXXX ('X' étant un chiffre) .

Réflexion utilisée → Saisie du nom et prénom, si l'adhérent n'est pas trouvé, est déclenché la création de l'adhérent. La création de l'id de l'adhérent est automatisée. L'id de l'adhérent à la taille logique - 1 du tableau est copié et est ensuite effectué plusieurs if, else imbriquées permettant d'ajouter 1 à l'identifiant. Pour cette fonctionnalité, est utilisé le code ASCII des caractères. l'imbrication des if, else est utilisé pour tester si une case à pour valeure 9 (57 en ASCII). Si c'est le cas, la case qui précède est alors testée pour y faire +1. Dans le cas contraire, la case est simplement ajoutée à 1 en valeur numérique. Ainsi de suite jusqu'à arriver à la première case de poid fort pour la partie entière. Si la case est à 9 il n'est donc plus possible de générer de nouveaux identifiants qui ne sont pas déjà utilisés. Il y a donc en tout 9999 identifiants possibles, un adhérent ayant un identifiant unique.

- enregistrement de l'emprunt ou de la réservation si possible :

  Pour cette fonctionnalité, l'utilisateur saisit le nom du jeu voulu. Tant que le nom n'est pas valide, un nom de jeu valide lui est demandé. Une fonction est ensuite déclenchée pour savoir s'il reste des exemplaires en stock par rapport aux nombres d'emprunt pour le jeu en question. S'il reste au minimum un exemplaire, un emprunt est déclenché sinon une réservation est déclenchée. Dans le cas ou un emprunt est déclenché, est testé avant l'insertion si l'adhérent remplies les conditions suivantes :
  - durée de son inscription < 365 jours par rapport au jour de l'emprunt
  - nombre d'emprunt de l'adhérent < 3
  - l'adhérent n'a pas déjà effectué l'emprunt du jeu en question

  Dans le cas où l'emprunteur est validé pour emprunter l'emprunt est créé et inséré. L'id de l'emprunt à la taille logique - 1 du tableau est copié et est ensuite effectué plusieurs if, else imbriquées permettant d'ajouter 1 à l'identifiant. Pour cette fonctionnalité, est utilisé le code ASCII des caractères.

  Réflexion utilisée → Le nombre d'exemplaires actuellement empruntés est calculé en faisant la différence entre le nombre d'exemplaires du jeu total avec le nombre d'emprunts qu'il y a pour le jeu en question (parcours du tableau des emprunts en incrémentant un compteur si l'id du jeu de la case est le même que l'id du jeu voulu). La création d'un emprunt ou d'une réservation est ensuite décidée en fonction de la valeur de retour de la fonction comptant le nombre d'exemplaires.

Dans le cas de la création d'un emprunt, est testé avant la création et l'insertion la 'validité' de l'emprunteur. Premièrement, pour savoir si l'adhérent n'est pas arrivé ou n'a pas dépassé la date d'anniversaire de son inscription, est converti en jours la date de l'emprunt ainsi que la date de l'inscription. Le résultat est validé dans le cas d'une différence entre les deux dates inférieur à 365 jours (Le programme ne prend pas en compte les années bissextiles). Deuxièmement, pour savoir si l'adhérent n'a pas déjà 3 emprunts en cours, est compté le nombre d'emprunts en cours pour le jeu en incrémentant un compteur suite au parcours du tableau des emprunts. Troisièmement, pour savoir si l'adhérent n'a pas déjà effectué le même emprunt, est utilisé le parcours du tableau des emprunts en comparant l'id du jeu ainsi que l'id de l'adhérent.
Si au minimum l'un des 3 cas n'est pas valide, l'adhérent ne pourra pas emprunter. Pour ce qui est de la création de l'id de l'emprunt c'est le même raisonnement que pour l'id adhérent. Convention d'écriture : EXXXX ('X' étant un chiffre) .
Dans le cas où il ne reste plus d'exemplaires, est alors testée la validité de de son adhésion en fonction de la date. Si son adhésion est validée est alors créé et inséré dans la liste sa réservation. Pour ce qui est de la création de l'id de la réservation c'est le même raisonnement que pour l'id adhérent. Convention d'écriture : RXXXX ('X' étant un chiffre). Les dernières insertions sont insérées à la fin du tableau permettant de créer un ordre de priorité sur les réservations (premier enregistré, premier servi).

6) <u>retour d'un jeu. Le retour déclenche l'examen des réservations pour peut être transformer une réservation en emprunt et faire parvenir le jeu à l'adhérent l'ayant réservé</u> (VIGNON Ugo) :

- retour d'un emprunt :

  le retour d'un jeu déclenche la suppression de l'emprunt suivi d'une transformation de la première réservation trouvée pour le jeu en un emprunt (Dans le cas ou il y a une réservation)

  Réflexion utilisée → Saisie du nom et prénom, si l'adhérent n'a pas été trouvé, re-demande de saisie tant que le nom et prénom ne sont pas valides. Ensuite recherche de l'adhérent pour obtenir ses informations via sa position. Saisie du nom du jeu qui est retourné, si le jeu n'a pas été trouvé, re-demande de saisie tant que le nom du jeu n'est pas valide. Ensuite recherche du jeu pour obtenir ses informations via sa position en donnant en lui donnant l'id du jeu ainsi que l'id de l'adhérent. S'il ne s'agit pas d'une erreur, copie de l'id de l'emprunt qui suit l'emprunt qui va être supprimé. Cela permet de savoir à partir de

quel id d'emprunt il va falloir actualiser les id d'emprunts (Pour que les id d'emprunts se suivent et pour ne pas perdre des combinaisons d'id (9999 combinaisons d'id au total). Suppression de l'emprunt en décalant à gauche jusqu'à la position. Actualisation des id des emprunts à partir de l'id copié au préalable dans un tableau de caractères. Pour effectuer -1 à l'id des emprunts, est effectué une imbrication de if, en partant de la taille logique -1 les tests vont être de savoir si une case est égale à 0 (48 en ASCII) la case prendra la valeur 9 (57 en ASCII) et la taille qui suit (de droite à gauche) sera décrémenter sauf si la case vaut 0 et ainsi de suite.

- examen des réservations :

  Si une réservation pour un jeu en question a été trouvée, ajout d'un emprunt pour l'adhérent ayant réservé le jeu. Suppression de la réservation si l'ajout de l'emprunt s'est bien déroulé.

  Réflexion utilisée → Si une réservation pour un jeu en question a été trouvée, obtention de l'adhérent via une fonction de recherche d'un adhérent selon son id. Stockage de la taille logique du tableau des emprunts avant l'ajout d'un emprunt dans une variable "corbeille". Si la taille de la variable a été changée, il n'y a donc pas eu de problème lors de l'ajout d'un emprunt. Il est donc maintenant possible de supprimer la réservation. Lors de la suppression de la réservation est alors copié l'id de la réservation à partir de laquelle devra être effectué une actualisation de l'id des réservations. Suppression du Maillon devant être supprimé et actualisation de l'id des réservations à partir de l'id de la réservation stockée au préalable dans un tableau de caractères. Pour effectuer -1 à l'id des réservations est utilisé la même méthode que pour l'id des emprunts.

7) <u>annulation d'une réservation (l'adhérent a changé d'avis , il ne désire plus réserver le jeux)</u> (VIGNON Ugo) :

   recherche de la réservation pour un id d'adhérent ainsi qu'un id de jeu. Suppression si la réservation est trouvée et actualisation des id des réservations . Si ce n'est pas le cas, l'actualisation n'a pas lieu.

   Réflexion utilisée → saisie du nom et prénom de l'adhérent ainsi que du nom du jeu. Bouclage dans des boucles while tant que leurs positions respectives dans leur tableau n'ont pas été trouvées. Si la suppression de la réservation a lieu, copie de l'id de la réservation qui suit permettant de savoir à partir de quelle réservation aura lieu l'actualisation des id de réservations. Actualisation. Pour savoir si la suppression de la réservation a eu lieu, on test la première case du tableau de caractères (idResa) créé dans la fonction test pour savoir si elle vaut '\0'. Dans le cas où la suppression n'a pas lieu, aucun id de réservation n'est actualisé.

8) sauvegarde de l'ensemble des données dans les quatre fichiers (VIGNON Ugo) :

sauvegarde des informations des tableaux et listes dans plusieurs fichiers binaires. les fichiers de départ ne sont pas des fichiers binaires pour garder une certaine lisibilité pour nos tests, ainsi que pour la bonne compréhension d'une personne extérieure évaluant notre projet. (la fonctionnalité suivante permettra de tester la sauvegarde des fichiers binaires)

Réflexion utilisée → la sauvegarde est effectuée quand le choix du menu est de quitter le programme. Dans chaque fonction de sauvegarde, est ouvert leur fichier de sauvegarde respectif.

- tableau de pointeurs : parcours du tableau en écrivant via fwrite le contenu de la zone mémoire pointé par le pointeur.
- tableau simple (1 dimension) :sauvegarde du tableau via un seul fwrite
- liste : parcours de la liste via une fonction récursive en utilisant à chaque fois fwrite. Cela permet de sauvegarder le contenu de chaque Maillon de la liste.

9) chargement des données sauvegardées en binaires [BONUS] (VIGNON Ugo) :

chargement des tableaux et listes à partir des fichiers binaires .don . Nécessite au préalable d'avoir fait au minimum une sauvegarde en binaire.

Réflexion utilisée → Ouverture des fichiers binaires en lecture :

- tableau de pointeurs : allocation et réallocation de la première dimension quand la taille physique est égale à la taille logique. Allocation de la zone mémoire pointée par le pointeur et lecture d'un type structuré du fichier binaire via fread permettant de récupérer les informations ainsi que de faire pointer le pointeur de la case du la zone mémoire.
- tableau simple des emprunts (1 dimension) : allocation dynamique de la première dimension et lecture et remplissage du tableau via un seul fread.
- liste : allocation d'un de la zone mémoire d'un Maillon et lecture d'un Maillon via fread. bouclage tant que la lecture du fichier n'est pas terminée. Utilisation d'un variable p (pointeur sur Maillon/ liste) permettant d'insérer toutes les informations d'une réservation dans un Maillon de la liste. Utilisation de deux listes car à la fin de la lecture la liste p pointe sur le dernier Maillon de la liste alors que la liste l pointe toujours sur le premier Maillon de la liste.

10) libération des zones mémoires allouées dans le programme [BONUS] (VIGNON Ugo) :

Pour les tableaux de pointeurs il a fallu libérer la zone mémoire pointée par chaque pointeur avant de libérer la première dimension du tableau. Pour la liste il a fallu commencer de libérer du dernier Maillon jusqu'au premier Maillon car notre structure de Maillon n'admet pas de pointeur sur le Maillon précédent.

Réflexion utilisée → outil utilisé : free(.....)

- tableau de pointeurs : parcours du tableau en libérant la zone mémoire pointée par le pointeur suivi de la libération du tableau.
- tableau à 1 dimension : libération du tableau
- liste : appel récursif de la fonction. Lorsque le cas d'arrêt est déclenché, dépilement de la liste et libération de chaque Maillon alloué.

11) Affichage des Adhérents et des réservations [BONUS] (VIGNON Ugo) :

Affichage simple du tableau de pointeurs sur structure Adherent. Affichage simple de chaque réservation (Maillon) de la liste via une fonction récursive.

Cette fonctionnalité supplémentaire sert surtout pour faciliter la correction de notre projet (voir si les chargements ont bien fonctionné). Ça a également facilité la compréhension lors du développement du programme.