# Projet C

# *Groupe* : Samuel SIRVEN, Antoine DUTEYRAT, Mathys LIOSON

***Toutes les fonctions des fichiers proj.c, proj.h et mainproj.c se sont révélées fonctionnelles après le test de chacune d'entre elles***

## Structures

Jeu :

- identifiant (int idJeu),

- nom du jeu (char nom[26]),

- type du jeu (char type[14]),

- nombre d'exemplaires (int nbExp).

Adhérent :

- identifiant (int idAdherent),

- civilité (char civilite[4]),

- nom de l'adhérent (char nom[16]),

- prénom de l'adhérent (char prenom[16]),

- date d'inscription (date dateIns)

Emprunt, réservation, Emprunt/réservation (alias utilisés pour réduire le nombre de structure et donc de fonctions de chargement) :

- identifiant emprunt ou réservation (int idEmpResa),

- identifiant adhérent (int idAdherent), - identifiant jeu (int idJeu),

- date de l'emprunt ou de la réservation (date dateEmpResa)

Maillon (appelé 'node' pour des raisons de praticité) :

- Jeu stocké dans le maillon (jeu game),

- pointeur vers le maillon suivant (struct node \*next)

*En ce qui concerne la déclaration d'une Liste à la fin de la structure nodeType, j'ai préféré ne pas m'en servir et montrer dans tout le programme que les maillons étaient renvoyés par des pointeurs.*

## Menu

### Prototype menu

**int menu(void);**

Cette fonction sert uniquement à afficher le menu principal, elle renvoie un int qui est le choix de l'utilisateur.

## Date et upper/lower

### Prototype date du jour

**date dateNow(date date);**

Cette fonction prend en paramètre une chaine de caractère.

Eller permet de récupérer la date actuelle grâce au fonction **time()** et **strftime()** qui sont importer depuis la bibliothèque *time.h*.

### Prototypes chaines caractères

**void upper (char tab[]);**

Cette fonction prend en paramètre une chaine de caractères et la convertit en minuscules.

**void upperLower (char tab[]);**

Cette fonction prend en paramètre une chaine de caractères et convertit la première lettre en majuscule puis les autres lettres en minuscules.

**void formatTxt (char tab[]);**

Cette fonction prend en paramètre une chaine de caractères et convertit chaque '\_' en espace.

**void formatBin (char tab[]);**

Cette fonction prend en paramètre une chaine de caractères et convertit chaque espace en '\_'.

## Chargement des fichiers binaires

### Prototypes chargement jeux

**nodeType \*createNode(jeu game);**

Cette fonction prend un jeu en paramètre pour le stocker dans un maillon créé par *malloc (node = (nodeType\*)malloc(sizeof(nodeType)))*, on vérifie que le maillon a été alloué correctement puis on lui assigne le jeu en donnée et un pointeur vers le maillon suivant NULL pour éviter les erreurs.

**nodeType \*lastNode(nodeType \*head);**

Cette fonction prend la tête la liste en paramètre, parcourt la liste à partir de la tête et renvoie le dernier maillon rencontré *(current = current->next jusqu'à ce qu'il y ait current->next == NULL)*, si la tête n'existe pas on retourne NULL.

**void createLastNode(nodeType \*\*head, jeu game);**

Cette fonction se sert des deux fonctions au-dessus pour créer un maillon à la toute fin de la liste chaînée passée en paramètre (via sa tête), elle alloue la maillon grâce à createNode et rentre le jeu passé en paramètre.

**nodeType \*ChargementJeux(nodeType\* head, int \*nbJeux);**

Cette fonction prend en paramètre la tête de la liste chaînée et un pointeur vers le nombre de jeux du fichier (déclaré dans la fonction global).

Elle ouvre le fichier jeux.bin, teste sa viabilité *(!= NULL)*, puis lit le nombre de jeux (écrit en début de fichier), enfin, elle lit le jeu de chaque ligne jusqu'à arriver au nombre de jeux. Finalement, elle renvoie la liste à la fonction appelante.

### Prototypes chargement adhérents

**int ChargementAdherents (adherent \*tab[], int max);**

Cette fonction lis un fichier binaire puis alloue à chaque lecture de ligne un pointeur du tableau sur le quelle il y écrit le données. A la fin elle retourne le nombre total d'adherents.

### Prototypes chargement emprunt/réservation

**empResa \*lectureEmpResa (empResa tabEmpResa[], int \*nbEmpResa, FILE \*flot);**

Cette fonction lit tous les emprunts d'un fichier binaire. Elle prend en paramètre le tableau d'emprunts ou de réservations, un nombre d'emprunt ou réservation et un fichier binaire.

**emprunt \*ChargementEmprunt (int \*nbEmp);**

Cette fonction prend en paramètre un pointeur int vers le nombre d'emprunt déclaré dans la fonction global.

Elle lit en début de fichier le nombre d'emprunts, alloue le tableau dynamiquement et retourne le résultat de **lectureEmpResa()** du fichier emprunt.bin.

**réservation \*Chargementréservation (int \*nbResa);**

Cette fonctionne exactement comme la fonction **chargementEmprunt()**.

## Affichages

### prototype affichage jeux

**nodeType \*copyList(nodeType \*head);**

Cette fonction prend en paramètre la liste chaînée déjà existante puis la copie de façon récursive dans une nouvelle. A la fin elle retourne la nouvelle liste.

**void copierL(nodeType \*headbis, int nbJ, nodeType \*\*GD, nodeType \*\*S);**

Cette fonction prend en paramètre la liste chaînée de jeu, le nombre de jeux / 2, une liste chaîné G ou D et une liste sauvegarde S.

Elle permet de copier une partie de la liste sur la partie Gauche puis coupe à la *nbJ* position et sauvegarde la suite. Pour la partie droite elle copie juste la liste de sauvegarde.

**nodeType \*fusionL(nodeType \*G, nodeType \*D, int choix);**

Cette fonction prend en paramètre la liste gauche, la liste droite et le choix de tri.

Ensuite tant que les 2 listes ne sont pas *NULL* elle effectue un test pour copier chaque maillon 1 par 1 dans l'ordre dans notre liste fusionné. Puis elle relance la fonction en envoyant la valeur *next* d'une des 2 liste pour l'attribuer au maillon suivant de notre liste fusionné. Une fois qu'une des 2 listes est à nul elle copie toujours defaçon récursif le reste de la liste G ou D. Elle retourne la liste fusionée à la fin.

**nodeType \*triFusionL(nodeType \*headbis, int nbJ, int choix);**

Cette fonction prend en paramètre la liste chaînée de jeux, le nombre de jeux et le choix.

Elle permet de trier notre liste par le principe de la fusion en faisant appelle à la fonction **copierL** et **fusionL**.

**void triSwitch(nodeType \*head, int choix);**

Cette fonction prend en paramètre la liste chaînée de jeux et un choix, puis: - cas 1 : parcourt la liste en comparant avec *(strcmp())* les éléments de la liste deux à deux. Si le résultat est positif, on intervertit les deux éléments comparés (par nom), puis on redémarre du début de la liste. - cas 2 : de la même manière que pour le tri alphabétique, on parcourt la liste en intervertissant les éléments en fonction de leur type.

**int comptNbEmp(emprunt tabEmp[], int idJeu, int nbEmp);**

Cette fonction prend en paramètre le tableau d'emprunts, un identifiant de jeu et le nombre d'emprunts dans le tableau.

Elle parcourt le tableau d'emprunts en regardant chaque identifiant de jeu et itère le nombre d'emprunt du jeu si il s'agit de l'identifiant passé en paramètre. Elle renvoie ensuite le nombre d'emprunts pour le jeu en question

**void freeListe(nodeType \*head);**

Cette fonction libère l'espace alloué à chaque maillon à la fin du programme.

**void printJeu(nodeType\* head, emprunt tabEmp[], int nbEmp);**

Cette fonction prend en paramètre la liste chaînée, le tableau d'emprunts et le nombre d'emprunts dans ce tableau.

Parcourt la liste chaînée et calcule le nombre d'exemplaire (en retirant le nombre d'emprunts) du jeu actuel, si son nombre d'exemplaire est supérieur à zéro on l'affiche à l'écran, sinon, on passe au jeu suivant.

**void printAllJeu(nodeType\* head, emprunt tabEmp[], int nbEmp);**

Cette fonction prend en paramètre la liste chaînée de jeu, le tableau d'emprunt, et le nombre total d'emprunt.

Elle fait appelle à la fonction **compteNbEmp()** pour pouvoir afficher le nombre d'exemplaire restant en fonction des emprunts.

**int menuJeu(void);**

Cette fonction affiche un sous-menu qui laisse deux choix à l'utilisateur : - Afficher les jeux disponibles (triés par type et ordre alphabétique) - Afficher tous les jeux de la ludothèque Elle renvoie le choix de l'utilisateur à la fonction **global()**.

### Prototype affichage adhérents

**void printAdherent(adherent \*tabAdh[],int nbAdh);**

Cette fonction prend en paramètre le tableau d'adhérent et le nombre d'adhérents.

Elle permet d'afficher tous les adhérents trier par id

### Prototypes affichage emprunts/réservations

**void dateEmp (emprunt tabEmp[], date date, int i, int \*exp);**

Cette fonction prend en paramètre le tableau d'emprunt, la date du jours, la position dans le tableau et un pointeur exp.

Elle permet de comparé la date du jour à celle de l'emprunt pour savoir si l'emprunt en question a plus de 1 mois. Si c'est le cas elle affecte la valeur 1 à *\*exp*.

void printEmprunt (emprunt tabEmp[], date date, int max);

Cette fonction prend en paramètre le tableau d'emprunt, la date du jours et le nombre d'emprunt.

Elle permet d'afficher tout les emprunt en cours et grace à l'appelle de la fonction **dateEmp()** on choisi d'afficher ou non la mention *"Retard de l'emprunt"*.

**void printResa(réservation \*tabResa, nodeType \*head, int nbResa, int nbJeux);**

Cette fonction prend en paramètre le tableau de réservation, la liste chaînée, le nombre de réservation et le nombre de jeux.

Elle permet d'afficher les réservation en fonction du jeu choisi par l'utilisateur qui est formaté grâce à la fonction **uperLower()** et ensuite vérfié grâce à la fonction **rechercheJeu()**.

## Ajout d'un jeu

### Prototype pour un nouveau jeu

**nodeType \*saisieNouveauJeu(nodeType \*head, int \*nbJeux); (bonus)**

Cette fonction permet de créer un nouveau jeu, elle demande à l'utilisateur de saisir les informations nécessaires puis utilise **createLastNode()** pour ajouter le jeu en bout de liste.

La fonction retourne la liste modifée.

## Ajout d'un/e emprunt/réservation

### Prototypes pour nouvel emprunt/réservation

**int rechercheAdherent(adherent \*tabAdh[], char nom[], char prenom[], int nbAdh);**

Cette fonction prend en paramètre le tableau d'adhérent, un nom, un prénom et le nombres d'adhérent.

Elle permet de chercher si le nom et prénom choisi appartienne à un adhérent ou non. Retourne sa position si trouvé ou -1 si il n'existe pas.

**void nouvelAdherent(adherent \*tabAdh[], date date, int \*nbAdh, int max);**

Cette fonction prend en paramètre le tableau d'adhérents, une date d'inscription, le nombre d'adhérents dans le tableau et la taille max du tableau.

Elle crée un nouveau pointeur alloué dynamiquement, assigne un identifiant à l'adhérent puis demande les informations nécessaires à son enregistrement. On convertit le nom en majuscule et le prénom en majuscule et minuscule, enfin on insère toutes ces données dans la nouvelle case pointée puis on ajoute 1 au nombre d'adhérents.

**void dateIns (adherent \*tabAdh[], date date, int i, int \*exp);**

Cette fonction prend en paramètre le tableau d'adherent, la date du jours, la position de l'adherent et un pointeur int.

Elle permet de comparé la date d'inscription à la date actuelle pour tester la validité de l'adherent. Elle met *\*exp* à 1 si il n'est plus valide.

**int comptNbEmpAdh(emprunt tabEmp[], int idAdh, int nbEmp);**

Cette fonction prend en paramètre le tableau d'adherent, l'id d'un adherent et le nombre d'emprunt.

Elle permet de compter combien d'emprunt a passer un adherent puis return cette valeur.

**int RetardEmp(emprunt tabEmp[], char date[], int nbEmp, int idAdh);**

Cette fonction prend en paramètre le tableau d'emprunt, la date du jours, le nombre d'emprunts et l'id de l'adhérent.

Elle permet de savoir si un emprunt effectué par cet adhérent et en retard ou non en regardant si pour un emprunt contenant l'id de notre adhérent, la focntion **dateEmp** nous retourne -1.

**int rechercheJeu(nodeType\* head, char name[25], int \*trouve, int nbJeux);**

Cette fonction prend en paramètre unr liste chaînée de jeu, une chaine de caractère, un pointeur int et le nombre de jeux.

Elle permet de comparer la chaine de caractère a un jeu de la liste et retourne sa position si elle existe.

**int rechercheJeuxAct (emprunt tabEmp[], int idJeu, int idAdh, int nbEmp);**

Cette fonction prend en paramètre le tableau d'emprunt/réservation, l'id d'un jeu, l'id d'un adherent et le nombre d'emprunt.

Elle permet de savoir si l'adherent a déjà emprunté ou reservé le jeu en question en comparent son id et l'id du jeu dans le tableau.

**empResa \*ajoutEmpResa(empResa tabEmpResa[], adherent \*tabAdh[], int \*nbEmpResa, int \*nbEmpResaMax, int \*idEmpResa, int idAdh, int idJeu, char date[]);**

Cette fonction prend en paramètre le tableau d'emprunt/réservation, le tableau d'adherent, le nombre d'adherent, le nombbre max du tableau, la date, la position de l'adherent dans sont tableau et l'id du jeu.

Elle permet de réallouer le tableau avec 5 place en plus si *(\*nb == \*nbMax)* puis copie ce/tte nouvel/le emprunt/réservation dans le tableau puis ajoute +1 au nombre d'emprunt. Pour finir elle retourne le nouveau tableau

**emprunt \*saisieNouvelEmprunt(adherent \*tabAdh[], emprunt tabEmp[], nodeType\* head, date date, int \*nbAdh, int \*nbEmp, int \*nbEmpmax, int nbJeu, int max);**

Cette fonction prend en paramètre les tableaux d'adhérents et d'emprunts, la tête de la liste, la date du jour, le nombre d'adhérents, d'emprunts, la taille max du tableau d'emprunts le nombre de jeux et la taille max du tableau adhérent.

On demande à l'utilisateur de saisir un nom d'emprunteur, on effectue par la suite plusieurs tests :

- existe-t-il ? s'il n'existe pas on propose à l'utilisateur de créer l'adhérent (si oui on lui demande si l'adherent à ajouter à payé sa cotisation), (si oui lance la fonction **nouvelAdherent()**).

- a-t-il 3 emprunts ? (Grâce à **comptNbEmpAdh()**)

- a-t-il rendu tous ses emprunts a temps ? (grâce à **dateEmp()**)

- est-il arrivé a expiration ? s'il n'as pas payé sa cotisation on demande à l'utilisateur si cela a été fait (si oui on met à jour sa date d'inscription).

Si toutes les conditions sont remplies, on demande d'entrer le nom du jeu emprunté, puis on le formate grâce à **upperLower**() puis on effectue de nouveaux tests :

- si le jeu existe (Grâce à **rechercheJeu()**)

- si l'adhérent ne l'as pas déjà emprunté (Grâce à **rechercheJeuxAct()**)

Enfin, on compte le nombre d'exemplaires restant du jeu, s'il est épuisé, on conseille à l'adhérent de créer une réservation, sinon, on crée l'emprunt via **ajoutEmpResa()**. On renvoie le tableau modifié à la fonction appelante.

**réservation \*saisieNouvelResa(adherent \*tabAdh[], réservation tabResa[], emprunt tabEmp[], nodeType\* head, date date, int \*nbAdh, int \*nbResa, int \*nbResaMax, int nbEmp, int nbJeux, int max);**

Cette fonction prend en paramètre tous les tableaux, la date du jours, tout les nombres (jeux, adherent, emprunt, resa) et les taille max des tableaux adherent, emprunt et réservation.

Elle permet d'ajouter une nouvelle réservation en demandant à l'utilisateur de saisir le nom de l'adherent souhaitant réserver puis, on effecture plusieurs tests :

- existe-t-il ? s'il n'existe pas on propose à l'utilisateur de créer l'adhérent (si oui on lui demande si l'adherent à ajouter à payé sa cotisation), (si oui lance la fonction **nouvelAdherent()**).

- est il arrivé à expiration ? (grâce à **dateIns()**) s'il n'as pas payé sa cotisation on demande à l'utilisateur si cela a été fait (si oui on met à jour sa date d'inscription) Ensuite si toutes les conditions sont remplies, on lui demande d'entrer le nom du jeu qu'il souhaite réserver, puis on effectue de nouveaux plusieurs tests :

- le jeu existe-t-il ? (Grâce à **rechercheJeu()**)

- a-t-il déjà réservé ce jeu ? (Grâce à **rechercheJeuxAct()**)

- est il disponible en emprunt ? (Grâce à **comptNbEmp()**) Enfin si toutes ces conditions sont remplies on fait appel à la fonction **ajoutEmpResa()** pour ajouter la nouvelle réservation. Pour finir on retourne le tableau de réservation.

## Retour jeu/annulation réservation

### Prototypes pour retour jeu/annulation réservation

**int printEmpResa(empResa tabEmpResa[], int nbEmpResa, int id);**

Cette fonction prend en paramètre le tableau d'emprunt ou de réservation, le nombre d'emprunt ou de réservation, et un id.

Elle permet de savoir les id des emprunts ou réservation effecuté par un adhérent choisi.

**empResa \*decalerGauche(empResa tabEmpResa[], int \*nbEmpResa, int val);**

Cette fonction prend en paramètre le tableau d'emprunt ou de réservation, la taille logique du tableau et un indice auquel commencer.

Elle décale cahque case du tableau à gauche puis renvoie le tableau modifié à la fonction appelante.

**int rechercheResa(reservation tabResa[], int nbResa, int idjeu);**

Cette fonction prend en paramètre le tableau de réservation, le nombre de réservation et l'id d'un jeu.

Elle permet de trouvé la première réservation sur un jeu donné. Elle *return* sa position ou -1 si elle n'en trouve pas.

**emprunt \*RetourJeu(adherent \*tabAdh[], emprunt tabEmp[], emprunt tabResa[], int \*nbEmp, int \*nbResa, int nbAdh, date date);**

Cette fonction demande le nom d'un utilisateur qui désire rendre un jeu, vérifie qu'il existe. Elle affiche ensuite tous les emprunts en cours enregistrés pour cet adhérent s'ils existent. Puis on demande à l'utilisateur de préciser un identifiant d'emprunt, vérifie qu'il est correct et effectue le retour et décalage à gauche si c'est le cas.

Enfin, la fonction fait passer une réservation sur ce jeu en emprunt, si tenté que l'adhérent soit autorisé à emprunter (abonnement expiré ou trop d'emprunts en cours).

**réservation \*AnnulationResa(adherent \*tabAdh[], réservation tabResa[], int \*nbResa, int nbAdh);**

Cette fonction a le même fonctionnement que **RetourJeu()** sans le passage d'une réservation en emprunt.

## Sauvegardes

### Prototypes pour la sauvegarde

**void saveJeux(nodeType \*head, int nbJeux);**

Cette fonction prend en paramètre la tête de la liste de jeux et le nombre de jeux.

Elle ouvre un fichier binaire, sauvegarde chaque jeu dedans, les jeux sont précédés de leur nombre dans le fichier. Enfin, elle ferme le fichier.

**void saveAdh(adherent \*\*tabAdh, int nbAdh);**

Cette fonction prend en paramètre le tableau de pointeurs d'adhérents et le nombre d'adhérents.

Elle ouvre un fichier binaire, sauvegarde chaque adhérent dedans, les adhérents sont précédés de leur nombre dans le fichier. Enfin, elle ferme le fichier.

**void saveEmp(emprunt \*tabEmp, int nbEmp);**

Cette fonction prend en paramètre le tableau d'emprunts et le nombre d'emprunts.

Elle ouvre un fichier binaire, sauvegarde chaque emprunt dedans, les emprunts sont précédés de leur nombre dans le fichier. Enfin, elle ferme le fichier.

**void saveResa(réservation \*tabResa, int nbResa);**

Cette fonction prend en paramètre le tableau de réservations et le nombre de réservations.

Elle ouvre un fichier binaire, sauvegarde chaque réservation dedans, les réservations sont précédées de leur nombre dans le fichier. Enfin, elle ferme le fichier.

**void sauvegarder(nodeType \*head, adherent \*\*tabAdh, emprunt \*tabEmp, réservation \*tabResa, int nbJeux, int nbAdh, int nbEmp, int nbResa);**

Cette fonction prend en paramètre les tableaux d'adhérents, d'emprunts et de réservations, on trouve aussi le nombre de jeux, d'emprunts, d'adhérents et de réservations.

Elle utilise toutes les fonctions sauvegarde binaire ci-dessus pour effectuer une sauvegarde binaire générale.

**void saveJeuxTxt(nodeType \*head, int nbJeux);**

Cette fonction prend en paramètre la tête de la liste de jeux et le nombre de jeux.

Elle ouvre un fichier txt, sauvegarde chaque jeu dedans avec une mise en page, les jeux sont précédés de leur nombre dans le fichier. Enfin, elle ferme le fichier.

**void saveAdhTxt(adherent \*\*tabAdh, int nbAdh);**

Cette fonction prend en paramètre le tableau de pointeurs d'adhérents et le nombre d'adhérents.

Elle ouvre un fichier txt, sauvegarde chaque adhérent dedans avec une mise en page, les adhérents sont précédés de leur nombre dans le fichier. Enfin, elle ferme le fichier.

**void saveEmpTxt(emprunt \*tabEmp, int nbEmp);**

Cette fonction prend en paramètre le tableau d'emprunts et le nombre d'emprunts.

Elle ouvre un fichier txt, sauvegarde chaque emprunt dedans avec une mise en page, les emprunts sont précédés de leur nombre dans le fichier. Enfin, elle ferme le fichier.

**void saveResaTxt(réservation \*tabResa, int nbResa);**

Cette fonction prend en paramètre le tableau de réservations et le nombre de réservations.

Elle ouvre un fichier txt, sauvegarde chaque réservation dedans avec une mise en page, les réservations sont précédées de leur nombre dans le fichier. Enfin, elle ferme le fichier.

**void sauvegarderTxt(nodeType \*head, adherent \*\*tabAdh, emprunt \*tabEmp, réservation \*tabResa, int nbJeux, int nbAdh, int nbEmp, int nbResa)**

Cette fonction prend en paramètre tout les tableaux et tout les nombres (jeux, adherent, emprunt, resa)

Elle demande si l'utilisateur veux changer de répertoire de sauvegarde pour les fichier txt ou non. Ce nouveau repertoir n'affectera que les fichiers txt et non les fichiers binaire. Puis fait appelle à la fonction **sauvergarder()** et toute les fonction de sauvegarde en Txt (**saveJeuxTxt()**, **saveAdhTxt()**, **saveEmpTxt()**, **saveResaTxt()**)

**int menuSave(void);**

Cette fonction permet d'afficher un sous-menu qui nous propose 2 options: - Sauvegarder en fichier binaire. - Sauvegarder en fichier binaire et texte.

## Fonction global

### Prototype fonction global

**void global(void);**

Cette fonction est la fonction globale, elle sert à : - déclarer toutes les variables et tableaux nécessaires au programme - charger les fichiers dans les tableaux et la liste - faire fonctionner les choix dans le menu et les fonctions correspondantes - libérer l'espace alloué à tous les tableaux et à la liste