BERQUEZ Léa

ROSSMANN Justin

Rapport du projet

Le sujet de notre projet est de créer un programme en langage C permettant de gérer un système de comptes bancaires. On doit chercher à manipuler des structures assez importantes ayant des connections entre elles et de créer différents menus, un pour l'administrateur de la banque, un pour les clients et un pour les nouveaux clients.

Un autre de nos objectifs est d'apprendre à travailler en équipe.

Table des matières

I. Les Fonctions 2
1) Structures 2
a) Fonctions d'initialisation 3
b) Fonctions qui vont gérer la liste chaînée 3
2) Administrateur 4
a) Fonctions pour la banque 4
b) Fonctions pour les comptes 4
c) Fonctions pour les titulaires de compte 4
d) Fonctions pour un compte 5
3) Client_Administrateur 5
a) Fonctions pour les opérations 6
b) Fonction pour le Titulaire_Compte 6
c) Fonctions pour le mot de passe 7
4) Client 7
a) Fonctions pour le titulaire 7
b) Fonctions pour les opérations 8
5) Entrées 8
6) Menus 8
a) Fonctions pour le menu 9
7) Fichiers 10
a) Fonctions pour le JSON 10
b) Fonctions pour l'exportation en format CSV Operations 10
II. Schéma du programme 10
III. Bilan du travail 12

I. Les Fonctions

Pour ce projet nous avons séparé l'ensemble des fonctions en plusieurs catégories ou librairies.

Les différentes librairies :

- Structures
- Administrateur
- Client_Administrateur
- Client
- Entrées
- Menus
- Fichier

Cela nous a permis de nous répartir le travail et d'avancer plus rapidement.

1) Structures

Cette librairie contient les structures du programme ainsi que les fonctions d'initialisation de toutes ces structures. Il y a aussi les fonctions fondamentales de gestion de la liste chaînée utilisée à divers endroits du programme.

Ce programme utilise un total de 7 structures qui sont réparties en 2 groupes:

- Groupe Liste: la structure élément et la structure liste. Elles permettent de pouvoir créer des listes chaînées dans les structures du groupe banque. La structure élément à un contenu variable en fonction du type de liste dans lequel elle est utilisée: Opération, Compte, Numéro de Compte ou Titulaire de Compte (utilisation d'une union). La structure liste contient des éléments de type élément.
- Groupe Banque: elle contient 5 structures:
  Titulaire_Compte : elle représente un client de la banque,
  Compte : elle représente un compte bancaire
  Comptes : elle est un ensemble de 4 listes de Compte répartis suivant le type du compte (Livrets A, PEL, Comptes Courants, Comptes Joints)
  Opération : qui représente un virement ou un changement d'informations c'est à dire toute modification d'un compte
  Banque : elle contient une liste de Titulaire_Compte, une structure Compte pour l'ensemble des comptes et le mot de passe de l'administrateur (crypté).

a) Fonctions d'initialisation

Les différentes fonctions :

- Element initialiserElement();
- Liste initialiserListe(int type);
- Titulaire_Compte initialiserTitulaire();
- Operation initialiserOperation();
- Compte initialiserCompte();
- Comptes initialiserComptes();
- Banque initialiserBanque();

Elles vont créer un objet du type correspondant avec les mallocs nécessaires, et affecter une valeur initiale aux différents éléments de la structure pour finalement le renvoyer. Cela permet de faire une allocation de mémoire.

b) Fonctions qui vont gérer la liste chaînée

void afficherListe(Banque, Liste);
La liste possède un attribut "type", on parcourt la liste et en fonction de son type on appelle les différentes fonctions d'affichage pour des opérations, des titulaires ou des comptes.

void ajoutListe(Liste liste, Element elem);
On ajoute l'élément elem à la fin de la liste.

int rechercherElement(Liste liste, int identifiant);
Suivant le type de la liste, l'identifiant va signifier quelque chose de différent:

- Pour une liste de comptes ou de numéro de compte, c'est le numéro du compte
- Pour une liste de Titulaires c'est l'identifiant du titulaire.

Cette fonction va donc parcourir la liste, chercher l'identifiant et renvoyer la position de l'élément (la liste est indexée à partir de 0), et renvoyer -1 si l'élément correspondant à cet identifiant n'est pas dans la liste.

Element retournerElement(Liste liste, int position);
Renvoie un élément à la position donnée dans la liste

int supprimerElement(Liste liste, int position);
Supprime un élément à la position donnée.

2) Administrateur

Cette librairie contient les fonctions que seul l'administrateur utilise. Il s'agit de fonctions permettant par exemple de modifier le solde d'un compte, ce qu'un simple client ne devrait pas être capable de faire. A priori l'administrateur a tous les droits, sauf celui de modifier le mot de passe d'un client.

a) Fonctions pour la banque

void afficherBanque(Banque);
Elle permet d'afficher des informations basiques sur la banque telles que le nombre de clients et le nombre de comptes dans la banque.

void modifierMotDePasseAdmin(Banque);
Elle permet de modifier le mot de passe de l'administrateur, elle utilise la fonction modifierMotDePasseAdmin(Banque) (fonction suivante).

int demanderAncienMDPAdmin(Banque);
Elle demande l'ancien mot de passe, s'il n'est pas le même que celui sauvegardé dans la banque, la procédure de changement de mot de passe s'arrête.

b) Fonctions pour les comptes

void afficherListeComptes(Banque);
Elle affiche tous les comptes de la banque dans l'ordre Comptes Courants, Comptes Joints, Livrets A, PEL.

void afficherListeNumerosCompte(Banque);
Elle affiche seulement les numéros de tous les comptes, dans le même ordre que précédemment.

c) Fonctions pour les titulaires de compte

Titulaire_Compte choisirTitulaire(Banque);
Elle appelle afficherListeIdTitulaires afin d'afficher, de pouvoir choisir un des numéros et de renvoyer le titulaire correspondant.

void afficherListeTitulaires(Banque);
Elle affiche la liste de tous les clients de la banque par ordre d'ajout.

void supprimerClient(Banque, Titulaire_Compte);
Elle supprime un client et va aussi supprimer chez les comptes de ce client son numéro en plus de réduire le nombre de leurs titulaires de 1 et de les désactiver si jamais ce chiffre atteint 0.

void afficherListeIdTitulaires(Banque);
Elle affiche la liste des numéros des titulaires avec leur noms.

d) Fonctions pour un compte

Compte choisirCompte(Banque B) ;
Elle appelle AfficherListeNumérosCompte pour afficher tous les numéros et pouvoir en choisir un et renvoyer le compte correspondant.

Void afficherTitulairesCompte(Banque, Compte) ;
Elle affiche le titulaire d'un compte (ou les titulaire s'il s'agit d'un compte joint).

Void modifierCompte(Banque, Compte) ;
Elle permet d'accéder à un menu où on peut choisir différentes modifications à effectuer sur un compte.

Void modifierTitulairesCompte(Banque, Compte) ;
Elle permet d'accéder à un menu ou on peut choisir quel titulaire d'un compte modifier ou supprimer

Void modifierEtatCompte(Compte);
Elle passe un compte en INACTIF s'il est ACTIF et l'inverse.

Void modifierSolde(Compte) ;
Elle permet de changer le solde d'un compte.

Void supprimerCompte(Banque, Compte) ;
Elle supprime le compte et va supprimer chez ses titulaires son numéro dans la liste de leurs comptes.

Void ajouterArgentCompte(Compte) ;
Elle permet d'ajouter de l'argent au compte (montant positif uniquement).

Void creerCompte(Banque);
Elle permet de créer un compte bancaire, de choisir son type et ses titulaires.

3) Client_Administrateur

Elle contient les fonctions utilisées à la fois par l'administrateur et les clients. Principalement des fonctions comme créer des comptes clients, des comptes bancaires et les modifier dans une certaine mesure.

Int getNumC(Compte) ;
Elle renvoie le numéro de compte du compte en paramètre.

Void afficherCompte(Banque, Compte) ;
Elle affiche un compte.

Int rechercherCompte(Banque, int) ;
Elle cherche un compte grâce à son numéro dans l'ensemble de la banque (les 4 listes de comptes) et renvoie son type.

Compte retournerCompte(Banque, int) ;
Elle renvoie un compte à partir de son numéro donnée en paramètre.

Int creerNumero(Comptes, int);
Elle crée un numéro de comptes différents de tous les autres existants. Les numéros étant dans l'ordre dans les différentes listes, va dans la liste correspondant au type donné en paramètre, et retourne le numéro du dernier compte +1. Les numéros de comptes sont définis de la façon suivante:
1000-1999 Comptes Courants / 2000-2999 Comptes Joints / 3000-3999 Livrets A / 4000-4999 PEL

Void modifierTitulaireCompte(Banque, Compte, int);
Elle permet de modifier un des titulaires d'un compte ou d'en ajouter un.

Void modifierType(Banque, Compte) ;
Elle permet d'initialiser le type d'un compte en proposant les 4 différents à l'utilisateur.

a) Fonctions pour les opérations

Operation creerOperation(Compte , Compte , int) ;
Elle permet de créer une opération pour un virement entre 2 comptes d'un certain montant et de la renvoyer.

Void afficherOperation(Operation) ;
Elle affiche une opération.

Void afficherListeOpPeriode(Compte, int, int) ;
Elle affiche la liste des opérations d'un compte sur une période commençant par le premier int en paramètre et se terminant par le deuxième. Les dates sont définies de la façon AAAAMMJJ et sont automatiquement ajoutées dans l'opération lors de la fonction initialiserOperation() avec la date du jour.

b) Fonction pour le Titulaire_Compte

Void afficherListeComptesClient(Banque, Titulaire_Compte) ;
Elle affiche la liste des comptes bancaires du titulaire entré en paramètre.

Int rechercherTitulaire(Banque, int) ;
Elle cherche si le titulaire avec cet identifiant existe dans la banque (si oui renvoie sa position dans B->ListeClients, sinon renvoie -1).

Titulaire_Compte retournerTitulaire(Banque, int) ;
Elle renvoie le titulaire avec ce numéro dans la banque.

Void modifierTitulaireNom(Titulaire_Compte) ;
Elle permet de modifier le nom-prénom du titulaire

Void modifierTitulaireMail(Titulaire_Compte) ;
Elle permet de modifier le mail du titulaire.

Void modifierTitulaireTel(Titulaire_Compte) ;
Elle permet de modifier le numéro de téléphone du titulaire.

Void afficherClient(Titulaire_Compte) ;
Elle permet d'afficher un titulaire.

Void modifierClientMessage(Banque, Titulaire_Compte);
Permet de modifier le message dans du titulaire (cela permet de faire des demandes à l'administrateur).

Void modifierTitulaire(Banque, Titulaire_Compte) ;
Elle affiche un menu pour choisir quelles informations d'un titulaire modifier.

Void creerTitulaire(Banque) ;
Elle permet de créer un titulaire en entrant son nom, son mail, son numéro de téléphone.

c) Fonctions pour le mot de passe

int demanderNouveauMDP() ;
Elle demande un nouveau mot de passe, une confirmation et le renvoie si tout s'est bien déroulé.

int crypter(int);
Elle crypte un entier

int decrypter(int);
Elle décrypte un entier

4) Client

Elle contient les fonctions utilisées seulement par le client, par exemple le fait de faire un virement que l'administrateur n'utilise pas (l'administrateur n'a pas de propre compte associé).

a) Fonctions pour le titulaire

Void changerMotDePasse(Titulaire_Compte);
Elle permet de changer le mot de passe d'un titulaire.

Int demanderAncienMDP(Titulaire_Compte) ;
Elle permet de demander le mot de passe actuel d'un titulaire.

b) Fonctions pour les opérations

Void virement(Compte, Compte);
Elle permet de faire un virement entre deux comptes (création en même temps d'une opération).

Void demandeCreationCompte(Banque, Titulaire_Compte);
Elle permet de faire une demande de création de compte à l'administrateur.

Void demandeSuppressionCompte(Banque, Titulaire_Compte);
Elle permet de faire une demande de suppression de compte à l'administrateur.

5) Entrées

Librairie récupérée sur le projet S3 et étendue contenant des ensembles de fonctions facilitant la saisie d'informations au clavier ainsi que des tests sur la conformité des informations entrées. Elle contient aussi fonctions permettant d'entrer le mot de passe sans qu'il s'affiche à l'écran.

int lire_fin_ligne();
Elle permet de voir s'il n'y a pas eu d'erreur de saisie après un scanf.

Int lire_format(char* , void*) ;
Elle appelle lire_decimal, lire_entier ou lire_string suivant le paramètre entré.

Int lire_decimal(float *) ;
Elle lit un flottant et le stocke à l'adresse donnée.

int lire_entier(int *);
Elle lit un entier et le stocke à l'adresse donnée.

Int lire_string(char**) ;
Elle lit un string et le stocke à l'adresse donnée.

Int convertCharInInt(char *) ;
Elle transforme un char en entier correspondant.

Int lire_MDP() ;
Elle permet d'entrer un mot de passe à l'écran en affichant des étoiles à la place. (Il n'est possible d'entrer que des entiers).

6) Menus

Elle contient les fonctions gérant les menus ainsi que les déplacements entre eux.

a) Fonctions pour le menu

Void menuPrincipal(Banque);
Elle implémente le menu laissant un choix vers quel type de session se connecter (client, nouveau client, admin ou se déconnecter).

Void ConnectionAdministrateur(Banque) ;
Elle gère la connexion d'un administrateur avec saisie du mot de passe et vérification de sa conformité.

Void menuAdministrateur(Banque) ;
Elle affiche tout d'abord la liste des demandes des clients et les fait traiter par l'administrateur avant de passer au menu administrateur.

Void menuGestionDesComptes(Banque);
Elle propose plusieurs choix de gestion de comptes.

Void menuGestionDesClients(Banque) ;
Elle propose plusieurs choix de gestion de clients.

Void menuAdministrationAdmin(Banque) ;
Elle permet de changer le mot de passe administrateur.

Void menuDesactiver_SupprimerCompte(Banque) ;
Elle met en place le menu laissant un choix pour désactiver ou supprimer un compte, un compte désactivé ou inactif ne peut plus faire de virements.

Void ConnectionClients(Banque) ;
Elle gère la connexion d'un administrateur avec saisie de l'identifiant, du mot de passe et vérification de leur conformité.

Void menuClients(Banque, Titulaire_Compte) ;
Elle affiche le menu des clients.

Void menuGestionDesComptesClient(Banque, Titulaire_Compte);
Elle met en place un menu avec différentes fonctions de modification d'un compte du client.

Void menuVirement(Banque, Titulaire_Compte);
Elle permet de faire un virement entre 2 comptes.

Void menuAdministrationClient(Banque, Titulaire_Compte);
Elle met en place différentes options comme faire une demande à l'administrateur, consulter des opérations…

Void menuNouveauClient(Banque) ;
Elle permet à un nouveau client de la Banque de créer un compte titulaire et ensuite un premier compte bancaire sans devoir passer par une demande auprès de l'administrateur.

7) Fichiers

Elle contient les fonctions permettant d'écrire et de le lire dans un fichier .json. (Utilise la librairie parson téléchargée sur internet à l'adresse https://github.com/kgabis/parson)

a) Fonctions pour le JSON

void chargementMDPBanque(Banque);
Elle met le mot de passe (crypté) de la banque depuis le JSON mot de passe administrateur dans la structure banque initialisée dans le main.

Void remplirTitulaireStructurePremierePhase (Banque) ;
Elle crée des Titulaires dans la banque et les remplit avec les informations du JSON des clients.

Void remplirComptesStructureDeuxiemePhase(Banque) ;
Elle crée des Comptes dans la banque et les remplit avec les informations du JSON des comptes.

int chargementJSON(Banque);
Elle appelle les 3 précédentes.

void ecrireComptes(Banque, Liste, FILE*);
Elle écris dans le fichier JSON une liste de comptes.

void fermetureTitulaires(Banque);
Elle écrase le fichier JSON précédent et écrit dedans la nouvelle liste des titulaires.

void fermetureComptes(Banque);
Elle écrase le fichier JSON précédent et écrit dedans les 4 nouvelles liste des comptes.

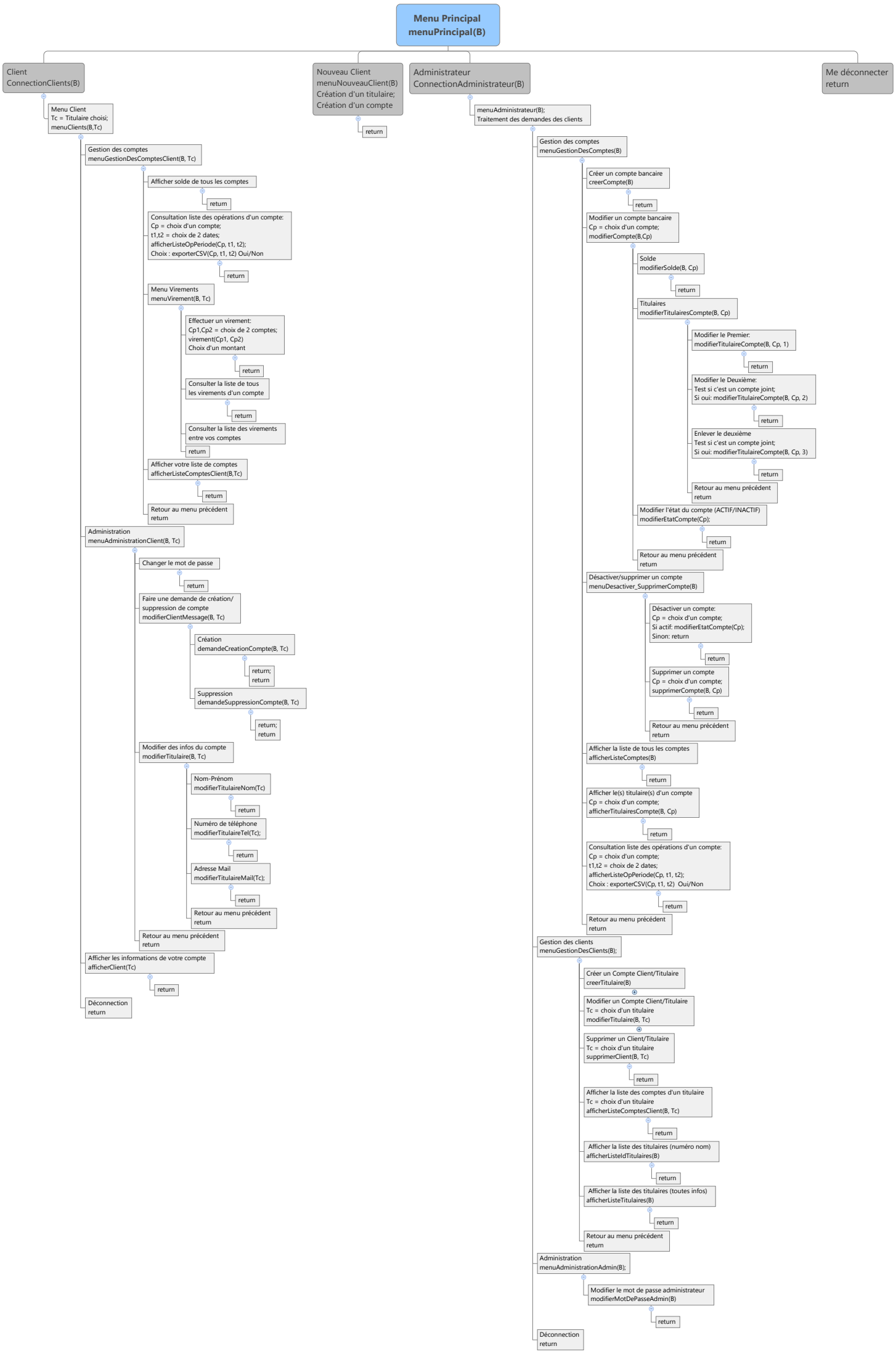
Void fermetureMDP(Banque) ;
Elle écrase le mot de passe administrateur précédent et sauvegarde le nouveau (crypté).

Void fermetureJSON(Banque) ;
Elle appelle les 3 précédentes.

b) Fonctions pour l'exportation en format CSV Operations

void exporterCSV(Compte, int, int) ;
Elle exporte au format CSV dans un fichier la liste des opérations d'un compte sur une certaine période donnée par les deux entiers en paramètre.

II. Schéma du programme


Menu Principal
menuPrincipal(B)
Client
ConnectionClients(B)
Menu Client
Tc = Titulaire choisi;
menuClients(B,Tc)
Gestion des comptes
menuGestionDesComptesClient(B, Tc)
Afficher solde de tous les comptes
return
Consultation liste des opérations d'un compte:
Cp = choix d'un compte;
t1,t2 = choix de 2 dates;
afficherListeOpPeriode(Cp, t1, t2);
Choix : exporterCSV(Cp, t1, t2) Oui/Non
return
Menu Virements
menuVirement(B, Tc)
Effectuer un virement:
Cp1,Cp2 = choix de 2 comptes;
virement(Cp1, Cp2)
Choix d'un montant
return
Consulter la liste de tous
les virements d'un compte
return
Consulter la liste des virements
entre vos comptes
return
Afficher votre liste de comptes
afficherListeComptesClient(B,Tc)
return
Retour au menu précédent
return
Administration
menuAdministrationClient(B, Tc)
Changer le mot de passe
return
Faire une demande de création/
suppression de compte
modifierClientMessage(B, Tc)
Création
demandeCreationCompte(B, Tc)
return;
return
Suppression
demandeSuppressionCompte(B, Tc)
return;
return
Modifier des infos du compte
modifierTitulaire(B, Tc)
Nom-Prénom
modifierTitulaireNom(Tc)
return
Numéro de téléphone
modifierTitulaireTel(Tc);
return
Adresse Mail
modifierTitulaireMail(Tc);
return
Retour au menu précédent
return
Retour au menu précédent
return
Afficher les informations de votre compte
afficherClient(Tc)
return
Déconnection
return
Nouveau Client
menuNouveauClient(B)
Création d'un titulaire;
Création d'un compte
return
Administrateur
ConnectionAdministrateur(B)
menuAdministrateur(B);
Traitement des demandes des clients
Gestion des comptes
menuGestionDesComptes(B)
Créer un compte bancaire
creerCompte(B)
return
Modifier un compte bancaire
Cp = choix d'un compte;
modifierCompte(B,Cp)
Solde
modifierSolde(B, Cp)
return
Titulaires
modifierTitulairesCompte(B, Cp)
Modifier le Premier:
modifierTitulaireCompte(B, Cp, 1)
return
Modifier le Deuxième:
Test si c'est un compte joint;
Si oui: modifierTitulaireCompte(B, Cp, 2)
return
Enlever le deuxième
Test si c'est un compte joint;
Si oui: modifierTitulaireCompte(B, Cp, 3)
return
Retour au menu précédent
return
Modifier l'état du compte (ACTIF/INACTIF)
modifierEtatCompte(Cp);
return
Retour au menu précédent
return
Désactiver/supprimer un compte
menuDesactiver_SupprimerCompte(B)
Désactiver un compte:
Cp = choix d'un compte;
Si actif: modifierEtatCompte(Cp);
Sinon: return
return
Supprimer un compte
Cp = choix d'un compte;
supprimerCompte(B, Cp)
return
Retour au menu précédent
return
Afficher la liste de tous les comptes
afficherListeComptes(B)
return
Afficher le(s) titulaire(s) d'un compte
Cp = choix d'un compte;
afficherTitulairesCompte(B, Cp)
return
Consultation liste des opérations d'un compte:
Cp = choix d'un compte;
t1,t2 = choix de 2 dates;
afficherListeOpPeriode(Cp, t1, t2);
Choix : exporterCSV(Cp, t1, t2) Oui/Non
return
Retour au menu précédent
return
Gestion des clients
menuGestionDesClients(B);
Créer un Compte Client/Titulaire
creerTitulaire(B)
Modifier un Compte Client/Titulaire
Tc = choix d'un titulaire
modifierTitulaire(B, Tc)
Supprimer un Client/Titulaire
Tc = choix d'un titulaire
supprimerClient(B, Tc)
return
Afficher la liste des comptes d'un titulaire
Tc = choix d'un titulaire
afficherListeComptesClient(B, Tc)
return
Afficher la liste des titulaires (numéro nom)
afficherListeIdTitulaires(B)
return
Afficher la liste des titulaires (toutes infos)
afficherListeTitulaires(B)
return
Retour au menu précédent
return
Administration
menuAdministrationAdmin(B);
Modifier le mot de passe administrateur
modifierMotDePasseAdmin(B)
return
Déconnection
return
Me déconnecter
return

III. Bilan du travail

Le travail à fournir a été bien au-delà des prévisions de notre professeur qui était parti sur une estimation de 20 heures de travail effectif. Nous nous sommes donc plutôt retrouvés à 40 voire 45 heures par personne pour l'ensemble du projet en tenant compte du rapport à rédiger, des nombreux tests à faire et de la résolution de certains problèmes plutôt étranges.

Beaucoup de temps a été consacré à la documentation pour trouver comment manipuler les informations du json par exemple ou encore pour trouver comment cacher le mot de passe en le remplaçant par des étoiles.

Notre plus gros problème au cours de ce projet était lié à l'allocation de la mémoire. Cela nous a posé de nombreux problèmes dans les structures en plus de nous prendre un temps très important pour les résoudre (au moins un quart du temps investit sur le projet). Par exemple certains champs changeaient de valeurs sans raisons apparentes et cela provoquait un nombre ahurissant de segmentation fault assez aléatoires au fur et à mesure de l'écriture du programme. Il a donc fallu que l'on trouve des alternatives.

Au final ce problème à tant bien que mal été résolu par l'utilisation de Valgrind, un outil permettant de vérifier l'ensemble des allocations mémoires et fuites de mémoire dans un programme C. Suite à son utilisation, nous avons décidé d'allouer une place bien plus importante à chaque structure, car avec les mallocs que nous avions fait, les différents éléments d'une structure empiétaient les uns sur les autres dans la mémoire et se corrompaient mutuellement.

Par exemple, au lieu de mettre Banque B = malloc(sizeof(Banque)); nous avons donc mis

Banque B = malloc(10*sizeof(Banque)); et ce pour toutes les structures (Le fait de ne jamais avoir rencontré un tel problème en cours ou en TP ne nous amené à cette résolution que très tard).

Dans l'ensemble, nous sommes plutôt satisfaits de notre travail, ayant mené à bout les idées que nous avons eu pour la version finale de ce projet.