```
class table_comptes
{
/*
la classe table_comptes va servir a stocker l'ensemble des comptes de la banque.
Elle contient donc des variables pour le nom, le prenom, l'identifiant, le mot de passe et le solde.
Elle a aussi des methodes pour exploiter les variables
*/

public:
    table_comptes(void);

    void setid(string id_);
    void setnom(string nom_);
    void setprenom(string prenom_);
    void setmdp(string mdp_);
    void setsolde(double solde_);
    string getid(void);
    string getnom(void);
    string getprenom(void);
    string getmdp(void);
    double getsolde(void);
    double changesolde(double value);
    double changesolde_controle(double value);

    void setnb_entrees(int nb_);
    int getnb_entrees(void);

    void display(void);

// variable pour le mutex
   HANDLE mutex_solde;

private:
    string id;
    string nom;
    string prenom;
    string mdp;
    double solde;

    int nb_entrees;
};
```

```
#endif



////////////////////////////////////////
// Creation de l'objet
table_comptes::table_comptes(void)
{
/*
Nous allons ici creer un mutex anonyme qui nous permettra par la suite de securiser l'acces a la variable
solde
*/
}


void table_comptes::setid(string id_)
{
/*
Permet de changer l'identifiant
*/
}


void table_comptes::setnom(string nom_)
{
/*
Permet de changer le nom
*/
}

void table_comptes::setprenom(string prenom_)
{
/*
Permet de changer le prenom
*/
}

void table_comptes::setmdp(string mdp_)
{
/*
```

```
Permet de changer le mot de passe
*/
}

void table_comptes::setsolde(double solde_)
{
/*
Permet de changer le solde
*/
}

string table_comptes::getid(void)
{
/*
Renvoie l'identifiant
*/
}

string table_comptes::getnom(void)
{
/*
Renvoie le nom
*/
}

string table_comptes::getprenom(void)
{
/*
Renvoie le prenom
*/
}

string table_comptes::getmdp(void)
{
/*
Renvoie le mot de passe
*/
}

double table_comptes::getsolde(void)
{
```

```
/*
Renvoie le solde
*/
}

double table_comptes::changesolde(double valeur)
{
/*
Cette methode permet d'ajouter ou de soustraire une certaine somme au solde.
Cette methode est critique car il ne faut pas qu'il y ait plusieurs operations de changement du solde en
meme temps.
Pour proteger l'operation de changement du solde, nous utiliserons un mutex
*/
}

double table_comptes::changesolde_controle(double valeur)
{
/*
Cette methode permet d'ajouter ou de soustraire une certaine somme au solde.
Cette methode est critique car il ne faut pas qu'il y ait plusieurs operations de changement du solde en
meme temps.
Pour proteger l'operation de changement du solde, nous utiliserons un mutex

Avant de changer le solde, nous effectuerons un controle.
Il s'agit de verifier qu'en cas de modification du solde, le solde soit positif.
Si le solde devait ne pas etre positif, alors on ne procede pas au changement du solde.
*/
}

void table_comptes::setnb_entrees(int nb_)
{
/*
Permet de changer le nombre de fiches contenus dans la base de donnees
*/
}


int table_comptes::getnb_entrees(void)
{
/*
Renvoie le nombre de fiches contenus dans la base de donnees
```

```
*/
}

void table_comptes::display(void)
{
/*
Affiche l'ensemble des informations d'une fiche (nom, prenom, mot de passe et identifiant
*/
}



void etape1(table_comptes *table, int nb_entrees)
{
/*
Nous ouvrons le fichier"la_table_des_comptes"

Nous savons combien il y a de fiches a creer.
Pour chaque fiche, nous allons recuperer le nom, le prenom, l'identifiant, le mot de passe et le solde

Le solde est de la forme chaine de caracteres et il faut donc la convertir en double.

Une fois que la fiche est complete, on demande son affichage a l'aide de la methode display().
*/
}



void etape2_consultation(table_comptes *table)
{
/*
On commence par creer un socket de type TCP.
On configure ensuite le socket en lui donnant un port et une adresse.

On associe le socket au port grace a la commande bind().

On ecoute ensuite le socket a l'aide de la fonction listen().
On demarre ensuite l'attente de connexions de la part de clients.
```

```
Des qu'une connexion avec un client est etablie (avec la fonction accept()), on cree un nouveau thread ( le
nom du thread est Thread_service_consultation2) et on lui transmet l'adresse de la base de donnees, ainsi
que l'adresse de la socket.
*/
}


void etape2_consultation2(table_comptes *table, int sock2)
{
/*
Nous sommes en communication avec le client.

Le client va nous transmettre un nom, un prenom, un mot de passe et l'identifiant d'un compte.
La reception se fait grace a la fonction recv().

Nous allons verifier que ces donnees correspondent bien a un compte qui existe.
Pour cela, nous lancons la fonction control1() qui a pour but de parcourir toutes les fiches de la base de
donnees pour voir s'il y en a une qui correspond aux informations en notre possession.

Si la verification n'a rien trouve, alors on envoie au client un message pour lui dire que nous ne pouvons
pas lui donner ce qu'il veut.
Si la verification est bon, on envoie au client un message pour dire que nous avons trouve la fiche qui
l'interesse.

Nous transmettons alors au client le solde du compte, ainsi que l'heure et la date a laquelle nous avons
regarde son solde.

Il ne reste plus qu'a fermer le socket
*/
}


int control1(char *idcompte, char *nom, char *prenom, char *mdp, char *solde, table_comptes *table)
{
/*
On commence par demander combien il y a de fiches dans la base de donnees.

Nous allons ensuite parcourir une a une toutes les fiches.
Pour chaque fiche, nous comparons son identifiant, son nom, son prenom et son mot de passe avec les donnees
que nous a transmis le client.
```

```
Si on trouve une fiche qui correspond, alors on note le solde du compte et en quitte la fonction avec la
valeur 0.
Si on ne trouve aucune fiche correspondante, alors on quitte la fonction avec la valeur -1
*/
}



void etape2_transfert(table_comptes *table)
{
/*
On commence par creer un socket de type TCP.
On configure ensuite le socket en lui donnant un port et une adresse.

On associe le socket au port grace a la commande bind().

On ecoute ensuite le socket a l'aide de la fonction listen().
On demarre ensuite l'attente de connexions de la part de clients.

Des qu'une connexion avec un client est etablie (avec la fonction accept()), on cree un nouveau thread ( le
nom du thread est Thread_service_virement2) et on lui transmet l'adresse de la base de donnees, ainsi que
l'adresse de la socket.
*/
}


void etape2_transfert2(table_comptes *table,int sock2)
{
/*
Nous sommes en communication avec le client.

Le client va nous transmettre un nom, un prenom, un mot de passe et l'identifiant de deux comptes. Il
transmet aussi une somme qui sera a retirer du compte A pour la mettre sur le compte B.
La reception se fait grace a la fonction recv().

Nous allons verifier que ces donnees correspondent bien a deux comptes qui existent.
Pour cela, nous lancons la fonction control2() qui a pour but de parcourir toutes les fiches de la base de
donnees pour voir s'il y en a une qui correspond aux informations en notre possession.
```

```
Si la verification n'a rien trouve, alors on envoie au client un message pour lui dire que nous ne pouvons
pas lui donner ce qu'il veut.
Si la verification est bon, on envoie au client un message pour dire que nous avons trouve la fiche qui
l'interesse.

Nous demandons ensuite a se que le virement se fasse grace a la fonction virement().
Nous envoyons ensuite un message au client pour lui dire que le virement a bien ete effectue, ou bien que le
virement n'a pas ete effectue car le solde A aurait ete negatif.
Enfin, nous transmettons au client l'actuel solde des comptes.

Il ne reste plus qu'a fermer le socket
*/
}


int virement(char *idcompteA, char *nomA, char *prenomA, char *mdpA, char *soldeA, char *idcompteB, char
*nomB, char *prenomB, char *mdpB, char *soldeB, char *somme, table_comptes *table)
{
/*
On commence par demander combien il y a de fiches dans la base de donnees.

Nous allons ensuite parcourir une a une toutes les fiches a la recherche de l'identifiant A.
Nous allons demander que soit retire du solde A une certaine somme a la condition que le solde A reste
positif.
Si le solde A est negatif, le retrait d'argent ne se fait pas et on quitte la fonction avec la valeur -1.

Si le solde A reste positif, alors on retire la somme.
On recherche l'identifiant B et on lui ajoute la somme.
On quitte la fonction avec la valeur de retour 0 pour dire que le virement s'est bien fait.
*/
}


int control2(char *idcompte, char *nom, char *prenom, char *mdp, char *solde, table_comptes *table)
{
/*
On commence par demander combien il y a de fiches dans la base de donnees.

Nous allons ensuite parcourir une a une toutes les fiches.
Pour chaque fiche, nous comparons son identifiant, son nom, son prenom et son mot de passe avec les donnees
que nous a transmis le client.
```

```
Si on trouve une fiche qui correspond, alors on note le solde du compte et en quitte la fonction avec la
valeur 0.
Si on ne trouve aucune fiche correspondante, alors on quitte la fonction avec la valeur -1
*/
}



// definition d'une cle pour la file de messages
#define CLE 666

struct msg
{
/*
Cette structure servira a stocker le nom, le prenom et l'identifiant du compte, ainsi que la somme a
crediter ou a debiter dessus
*/
};


void etape3(table_comptes *table)
{
/*
A l'aide d'une CLE, on cherche une file de messages. Si on ne trouve pas la file de messages, alors on la
cree.

On stocke la date actuelle.
Toutes les 5 secondes, on regarde si le jour a change. Si le jour est le meme, alors on ne fait rien. Si le
jour a change, alors on declenche une action.

On consulte a file de messages pour savoir combien de messages elle contient.
On recupere ensuite l'ensemble de ces messages.

Chaque message recu est stocke dans la structure msg.
On recherche dans la base de donnees, l'identifiant du compte et on lui ajoute ou enleve une certaine somme.
*/
}
```

```
// creation de la base de donnees
void etape1(table_comptes *table, int nb_entrees);

// service consultation
void etape2_consultation(table_comptes *table);
void etape2_consultation2(table_comptes *table,int socket2);
int control1(char *idcompte, char *nom, char *prenom, char *mdp, char *solde, table_comptes *table);

// service virement
void etape2_transfert(table_comptes *table);
void etape2_transfert2(table_comptes *table,int socket2);
int control2(char *idcompte, char *nom, char *prenom, char *mdp, char *solde, table_comptes *table);
void virement(char *idcompteA, char *nomA, char *prenomA, char *mdpA, char *soldeA, char *idcompteB, char
*nomB, char *prenomB, char *mdpB, char *soldeB, char *somme, table_comptes *table);

// service periodique
void etape3(table_comptes *table);


struct table_et_socket
{
/*
cette structure va contenir un pointeur vers un objet appartenant a la classe table_comptes et l'adresse
d'une socket
*/
};


DWORD WINAPI Thread_service_consultation(LPVOID lpParameter)
{
/* Partie 2 - Activation du service de consultation */
/*
le thread se contente de lancer la fonction etape2_consultation(), en mettant en parametre l'adresse de la
table de donnees et l'adresse du socket
*/
/* Fin de la partie 2 de consultation */
}

DWORD WINAPI Thread_service_consultation2(LPVOID basedonnees_et_socketclient)
{
/* Partie 2 - Traitement en thread d'une connexion client */
```

```
/*
le thread se contente de lancer la fonction etape2_consultation2(), en mettant en parametre l'adresse de la
table de donnees et l'adresse du socket
*/
/* Fin de la partie 2 de consultation avec connexion client */
}

DWORD WINAPI Thread_service_virement(LPVOID lpParameter)
{
/* Partie 2 - transfert argent */
/*
le thread se contente de lancer la fonction etape2_transfert(), en mettant en parametre l'adresse de la
table de donnees et l'adresse du socket
*/
/* Fin de la partie 2 de virement */
}

DWORD WINAPI Thread_service_virement2(LPVOID basedonnees_et_socketclient)
{
/* Partie 2 - Traitement en thread d'une connexion client */
/*
le thread se contente de lancer la fonction etape2_transfert2(), en mettant en parametre l'adresse de la
table de donnees et l'adresse du socket
*/
/* Fin de la partie 2 de virement avec connexion client */
}


int main(int argc, char* argv[])
{
/*
Au demarrage du serveur, on commence par mettre en memoire tous les comptes de la banque.
Pour cela, on recupere les donnees se trouvant dans le fichier "la_table_des_comptes" et on les stocke a
l'aide de la classe table_comptes.

Pour commencer, on ouvre le fichier "la_table_des_comptes" et on lit la premiere ligne pour savoir combien
il y a de fiches a entrer en memoire.
Ensuite, on lance la fonction etape1() en lui donnant en parametre l'adresse de la base de donnees, ainsi
que le nb de fiches a enregistrer dedans.
```

```
Une fois que la base de donnees des comptes est cree, on peut activer les services du service.
En tout, il y a 3 services qui doivent tourner en parallele, ce qui signifie la presence de 3 processus.
Il va donc etre necessaire de creer 2 nouveaux processus. On choisit de creer des processus legers, c'est a
dire des threads.

On cree un thread pour le service de consultation (son nom etant Thread_service_consultation).

On cree un thread pour le service de consultation (son nom etant Thread_service_virement).

On met ensuite en place l'analyse peridique.
Pour cela, on lance la fonction etape3().
*/
}
```