```
#ifndef TABLECOMPTES_H
#define TABLECOMPTES_H

#include <cstdlib>
#include <cstdio>
#include <stdlib.h>
#include <stdio.h>
#include <iostream>
#include <string>
#include <fstream>
#include <sys/types.h>
#ifdef WIN32
#include <windows.h>
#include <winsock2.h>
#include <commctrl.h>
#else
#include <sys/socket.h>
#include <pthread.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#endif
#include <unistd.h>
#include <memory.h>
#include <signal.h>
#include <sys/time.h>
#include <time.h>
#include <ctime>


using namespace std;

class table_comptes
{

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



#include "table_comptes.h"

////////////////////////////////////////
// Creation de l'objet
table_comptes::table_comptes(void)
{
    // creation du mutex pour protéger le solde
    mutex_solde = CreateMutex(NULL, FALSE, NULL); // mutex anonyme
      // Erreur de création ?
```

```
      if (!mutex_solde)
            printf("Impossible de créer le mutex\n");
}


void table_comptes::setid(string id_)
{
    id = id_;
}


void table_comptes::setnom(string nom_)
{
     nom=nom_;
}

void table_comptes::setprenom(string prenom_)
{
     prenom=prenom_;
}

void table_comptes::setmdp(string mdp_)
{
     mdp=mdp_;
}

void table_comptes::setsolde(double solde_)
{
     solde = solde_;
}

string table_comptes::getid(void)
{
     return id;
}

string table_comptes::getnom(void)
{
     return nom;
}
```

```
string table_comptes::getprenom(void)
{
     return prenom;
}

string table_comptes::getmdp(void)
{
     return mdp;
}

double table_comptes::getsolde(void)
{
     return solde;
}

double table_comptes::changesolde(double valeur)
{
     // il s'agit d'une opération critique, aussi, on protege a l'aide d'un mutex
     WaitForSingleObject( mutex_solde, INFINITE ); // acquérir le mutex
     solde = solde + valeur;
     ReleaseMutex( mutex_solde ); // libérer le mutex
     // fin de l'operation critique
     return solde;
}

double table_comptes::changesolde_controle(double valeur)
{
     // il s'agit d'une opération critique, aussi, on protege a l'aide d'un mutex
     WaitForSingleObject( mutex_solde, INFINITE ); // acquérir le mutex
     if( (solde+valeur) < 0) // on verifie que le solde sera bien positif
     {
          ReleaseMutex( mutex_solde ); // libérer le mutex
          return -1;
     }

     // si a la fin de l'operation, le solde est positif, alors on procede a l'operation
     solde = solde + valeur;
     ReleaseMutex( mutex_solde ); // libérer le mutex
     // fin de l'operation critique
     return solde;
}
```

```
void table_comptes::setnb_entrees(int nb_)
{
    nb_entrees=nb_;
}


int table_comptes::getnb_entrees(void)
{
    return nb_entrees;
}

void table_comptes::display(void)
{
    printf("id=%s - nom=%s - prenom=%s - mdp=%s -
solde=%f\n",id.c_str(),nom.c_str(),prenom.c_str(),mdp.c_str(),solde);
    return;
}



void etape1(table_comptes *table, int nb_entrees)
{

    std::ifstream fichier( "la_table_des_comptes" );
    if (fichier == NULL)
    {
        printf("Erreur de lecture de la base de donnees\n");
        exit(-1);
    }
    string ligne; // variable contenant chaque ligne lue
    getline( fichier, ligne );
    for(int i=0; i<nb_entrees;i++)
    {
       getline( fichier, ligne );
       table[i].setid(ligne);
       getline( fichier, ligne );
       table[i].setnom(ligne);
       getline( fichier, ligne );
       table[i].setprenom(ligne);
```

```
        getline( fichier, ligne );
        table[i].setmdp(ligne);
        getline( fichier, ligne );

        // conversion en char*
        size_t size = ligne.size() + 1;
        char* ligne2 = new char[ size ];
        strncpy( ligne2, ligne.c_str(), size );
        // conversion en double
        table[i].setsolde(atof(ligne2));

        table[i].display(); // affichage de la fiche
    }

}
```

```
void etape2_consultation(table_comptes *table)
{
// code pour activer l'API socket de Windows
        WSADATA         wsaData;
        int             res;
        if((res = WSAStartup(MAKEWORD(2,0), &wsaData)) != 0)       // initialisation de l'API socket sous
Windows
                printf("Impossible d'initialiser l'API Winsock 2.0\n");
// ! code pour activer l'API socket de Windows



// code pour creer un socket de type TCP
  int sock;
  if ((sock=socket(AF_INET,SOCK_STREAM,0)) == -1)
  {
    perror("Creation de socket impossible");
    return;
  }
// ! code pour creer un socket de type TCP
```

```
// configuration de l'adresse socket
  struct sockaddr_in adresse_socket;
  int longueur=sizeof(struct sockaddr_in);
  int  port=5666; // port de communication

   memset(&adresse_socket,0x0,sizeof(adresse_socket)); // on met toute la structure adresse_socket a zero

   adresse_socket.sin_family = AF_INET;
   adresse_socket.sin_port = htons(port);
   adresse_socket.sin_addr.s_addr=htonl(INADDR_ANY); // toutes les interfaces présentes
// ! configuration de l'adresse socket



// association du socket sur le port
      res=bind(sock, (struct sockaddr *)&adresse_socket,sizeof(adresse_socket)); // activation du socket
      if(res == INVALID_SOCKET)
      {
             perror("bind");
      }
// ! association du socket sur le port



// autorise l'ecoute du port pour 1 connexions simultanees
  if (listen(sock, 1) == -1) {
    perror("Listen");
    exit(1);
  }
// ! autorise l'ecoute du port pour 1 connexions


// attente d'une connexion
    DWORD threadID; // pointeur vers les threads
    int sock2;
    struct table_et_socket infostablesocket;
    while(1)      // boucle infinie
    {
```

```
//        printf("attente de communication client pour consultation\n");
        sock2=accept(sock,(struct sockaddr*) &adresse_socket, &longueur);
        if (sock2 == -1) {
          perror("Accept");
          return;
        }
        printf("communication client pour consultation etablie\n");

        infostablesocket.table=table;
        infostablesocket.socket=sock2;

//        printf("Creation d'un thread pour gerer la connexion client du service consultation\n");
        CreateThread(NULL, 0, Thread_service_consultation2, &infostablesocket, 0, &threadID);

    }
// ! attente d'une connexion




}


void etape2_consultation2(table_comptes *table, int sock2)
{
// reception des donnees
    int     receive; // donne en octets les donnees echangees
    char    nom[100];
    char    prenom[100];
    char    idcompte[100];
    char    mdp[100];

    // reception de l'id compte
    if((receive = recv(sock2,idcompte,100,0)) <= 0)
           printf("Echec de reception des donnees idcompte !\n");
    idcompte[receive] = '\0';// place à la fin du tableau le caractère nulle
    printf("id_compte=%s - ", idcompte);          // on afficher le contenu du buffer

    // reception du nom
    if((receive = recv(sock2,nom,100,0)) <= 0)
           printf("Echec de reception des donnees nom !\n");
```

```
    nom[receive] = '\0';// place à la fin du tableau le caractère nulle
    printf("nom=%s - ", nom);          // on afficher le contenu du buffer

    // reception du prenom
    if((receive = recv(sock2,prenom,100,0)) <= 0)
           printf("Echec de reception des donnees prenom !\n");
    prenom[receive] = '\0';// place à la fin du tableau le caractère nulle
    printf("prenom=%s - ", prenom);          // on afficher le contenu du buffer

    // reception du mot de passe
    if((receive = recv(sock2,mdp,100,0)) <= 0)
           printf("Echec de reception des donnees mdp !\n");
    mdp[receive] = '\0';// place à la fin du tableau le caractère nulle
    printf("mot_de_passe=%s\n", mdp);          // on afficher le contenu du buffer
// ! reception des donnees


// verification des donnees
   int verif;
   char    solde[100];
   verif = control1(idcompte, nom, prenom, mdp, solde, table);
// ! verification des donnees



//  envoie des donnees
    if( verif != 0)
    {
        // envoie de l'etat mauvais
        if(send(sock2,"1",strlen("1")+1,0) != 2)
               printf("Echec de l'envoie des donnees etat !\n");
    }
    else
    {

        char ladate[10]; // contient la date
        char lheure[10]; // contient l'heure
        _strdate(ladate);
        _strtime(lheure);

        // envoie de l'etat bon
```

```
        if(send(sock2,"2",2,0) != 2)
                printf("Echec de l'envoie des donnees etat !\n");
        // envoie de la date
        if(send(sock2,ladate,sizeof(ladate),0) != sizeof(ladate))
                printf("Echec de l'envoie des donnees ladate !\n");
        // envoie de l'heure
        if(send(sock2,lheure,sizeof(lheure),0) != sizeof(lheure))
                printf("Echec de l'envoie des donnees heure !\n");
        // envoie du solde
        if(send(sock2,solde,sizeof(solde),0) != sizeof(solde))
                printf("Echec de l'envoie des donnees solde !\n");
    }
// ! envoie des donnees



// fermeture du socket
        shutdown(sock2,2);
        close(sock2); // ferme le socket
// ! fermeture du socket

}


int control1(char *idcompte, char *nom, char *prenom, char *mdp, char *solde, table_comptes *table)
{
    int nb_entrees = 0;
    nb_entrees =  table->getnb_entrees();

    string id_;
    id_.assign(idcompte);
    string nom_;
    nom_.assign(nom);
    string prenom_;
    prenom_.assign(prenom);
    string mdp_;
    mdp_.assign(mdp);
    string solde_;

    for(int i=0; i<nb_entrees;i++)
    {
```

```
        // comparaison de l'id
        if (id_ == table[i].getid() && nom_ == table[i].getnom() && prenom_ == table[i].getprenom() && mdp_
== table[i].getmdp() )
        {
           printf("identification reussie\n");
           sprintf( solde, "%f", table[i].getsolde());
           return 0;
        }
     }


    return -1;
}




void etape2_transfert(table_comptes *table)
{
// code pour activer l'API socket de Windows
        WSADATA         wsaData;
        int             res;
        if((res = WSAStartup(MAKEWORD(2,0), &wsaData)) != 0)       // initialisation de l'API socket sous
Windows
                printf("Impossible d'initialiser l'API Winsock 2.0\n");
// ! code pour activer l'API socket de Windows



// code pour creer un socket de type TCP
  int sock;
  if ((sock=socket(AF_INET,SOCK_STREAM,0)) == -1)
  {
    perror("Creation de socket impossible");
    return;
  }
// ! code pour creer un socket de type TCP
```

```
// configuration de l'adresse socket
  struct sockaddr_in adresse_socket;
  int longueur=sizeof(struct sockaddr_in);
  int  port=5999; // port de communication
  char *adresse = "127.0.0.1"; // adresse IP du serveur

   memset(&adresse_socket,0x0,sizeof(adresse_socket)); // on met toute la structure adresse_socket a zero

   adresse_socket.sin_family = AF_INET;
   adresse_socket.sin_port = htons(port);
//   adresse_socket.sin_addr.S_un.S_addr = inet_addr(adresse);
   adresse_socket.sin_addr.s_addr=htonl(INADDR_ANY); // toutes les interfaces présentes
// ! configuration de l'adresse socket



// association du socket sur le port
      res=bind(sock, (struct sockaddr *)&adresse_socket,sizeof(adresse_socket)); // activation du socket
      if(res == INVALID_SOCKET)
      {
             perror("bind");
      }
// ! association du socket sur le port



// autorise l'ecoute du port pour 1 connexions simultanees
  if (listen(sock, 1) == -1) {
    perror("Listen");
    exit(1);
  }
// ! autorise l'ecoute du port pour 1 connexions



// attente d'une connexion
    DWORD threadID; // pointeur vers les threads
    int sock2;
    struct table_et_socket infostablesocket;
    while(1)      // boucle infinie
```

```
    {
//        printf("attente de communication client pour virement\n");
        sock2=accept(sock,(struct sockaddr*) &adresse_socket, &longueur);
        if (sock2 == -1) {
          perror("Accept");
          return;
        }
        printf("communication client pour virement etablie\n");

        infostablesocket.table=table;
        infostablesocket.socket=sock2;

//        printf("Creation d'un thread pour gerer la connexion client du service consultation\n");
        CreateThread(NULL, 0, Thread_service_virement2, &infostablesocket, 0, &threadID);

    }
// ! attente d'une connexion




}


void etape2_transfert2(table_comptes *table,int sock2)
{

// reception des donnees
    int     receive; // donne en octets les donnees echangees
    char    nomA[100],nomB[100];
    char    prenomA[100],prenomB[100];
    char    idcompteA[100],idcompteB[100];
    char    mdpA[100],mdpB[100];
    char    somme[100];

    // reception de l'id compte A
    if((receive = recv(sock2,idcompteA,sizeof(idcompteA),0)) != sizeof(idcompteA))
           printf("Echec de reception des donnees idcompteA !\n");
    idcompteA[receive] = '\0';// place à la fin du tableau le caractère nulle
    printf("id_compteA=%s - ", idcompteA);          // on afficher le contenu du buffer
```

```
// reception du nom A
if((receive = recv(sock2,nomA,sizeof(nomA),0)) != sizeof(nomA))
       printf("Echec de reception des donnees nomA !\n");
nomA[receive] = '\0';// place à la fin du tableau le caractère nulle
printf("nomA=%s - ", nomA);          // on afficher le contenu du buffer

// reception du prenom A
if((receive = recv(sock2,prenomA,sizeof(prenomA),0)) != sizeof(prenomA))
       printf("Echec de reception des donnees prenomA !\n");
prenomA[receive] = '\0';// place à la fin du tableau le caractère nulle
printf("prenomA=%s - ", prenomA);          // on afficher le contenu du buffer

// reception du mot de passe A
if((receive = recv(sock2,mdpA,sizeof(mdpA),0)) != sizeof(mdpA))
       printf("Echec de reception des donnees mdpA !\n");
mdpA[receive] = '\0';// place à la fin du tableau le caractère nulle
printf("mot_de_passeA=%s\n", mdpA);          // on afficher le contenu du buffer

// reception de l'id compte B
if((receive = recv(sock2,idcompteB,sizeof(idcompteB),0)) != sizeof(idcompteB))
       printf("Echec de reception des donnees idcompteB !\n");
idcompteB[receive] = '\0';// place à la fin du tableau le caractère nulle
printf("id_compteB=%s - ", idcompteB);          // on afficher le contenu du buffer

// reception du nom B
if((receive = recv(sock2,nomB,sizeof(nomB),0)) != sizeof(nomB))
       printf("Echec de reception des donnees nomB !\n");
nomB[receive] = '\0';// place à la fin du tableau le caractère nulle
printf("nomB=%s - ", nomB);          // on afficher le contenu du buffer

// reception du prenom B
if((receive = recv(sock2,prenomB,sizeof(prenomB),0)) != sizeof(prenomB))
       printf("Echec de reception des donnees prenomB !\n");
prenomB[receive] = '\0';// place à la fin du tableau le caractère nulle
printf("prenomB=%s - ", prenomB);          // on afficher le contenu du buffer

// reception du mot de passe B
if((receive = recv(sock2,mdpB,sizeof(mdpB),0)) != sizeof(mdpB))
       printf("Echec de reception des donnees mdpB !\n");
mdpB[receive] = '\0';// place à la fin du tableau le caractère nulle
printf("mot_de_passeB=%s\n", mdpB);          // on afficher le contenu du buffer
```

```
    // reception de la somme
    if((receive = recv(sock2,somme,sizeof(somme),0)) != sizeof(somme))
           printf("Echec de reception des donnees somme !\n");
    somme[receive] = '\0';// place à la fin du tableau le caractère nulle
    printf("somme=%s\n", somme);          // on afficher le contenu du buffer


// ! reception des donnees



// verification des donnees
   int verif=0;
    char    soldeA[100],soldeB[100];
    if( control2(idcompteA, nomA, prenomA, mdpA, soldeA, table) != 0)
    {
        // envoie de l'etat mauvais
        if(send(sock2,"1",2,0) != 2)
               printf("Echec de l'envoie des donnees verif !\n");
        verif = 1;
    }
    else
    {
        // envoie de l'etat bon
        if(send(sock2,"2",2,0) != 2)
               printf("Echec de l'envoie des donnees verif !\n");
    }


    // verification des donnees
    if( control2(idcompteB, nomB, prenomB, mdpB, soldeB, table) != 0)
    {
        // envoie de l'etat mauvais
        if(send(sock2,"1",2,0) != 2)
               printf("Echec de l'envoie des donnees verif !\n");
        verif = 1;
    }
    else
    {
        // envoie de l'etat bon
```

```
        if(send(sock2,"2",2,0) != 2)
                printf("Echec de l'envoie des donnees verif !\n");
    }
// ! verification des donnees



//  envoie des donnees
    char ladate[10]; // contient la date
    char lheure[10]; // contient l'heure
    _strdate(ladate);
    _strtime(lheure);

    if(verif == 0)
    {
         int etat;
         etat=virement(idcompteA,nomA,prenomA,mdpA,soldeA,idcompteB,nomB,prenomB,mdpB,soldeB,somme,table);

         if(etat == -1)
         {
               // envoie de l'etat mauvais
               if(send(sock2,"1",2,0) != 2)
                     printf("Echec de l'envoie des donnees verif !\n");
         }
         else
         {
               // envoie de l'etat bon
               if(send(sock2,"2",2,0) != 2)
                     printf("Echec de l'envoie des donnees verif !\n");
         }

        // envoie de la date
        if(send(sock2,ladate,sizeof(ladate),0) != sizeof(ladate))
               printf("Echec de l'envoie des donnees ladate !\n");
        // envoie de l'heure
        if(send(sock2,lheure,sizeof(lheure),0) != sizeof(lheure))
               printf("Echec de l'envoie des donnees lheure !\n");
        // envoie du solde A
        if(send(sock2,soldeA,sizeof(soldeA),0) != sizeof(soldeA))
               printf("Echec de l'envoie des donnees soldeA !\n");
        // envoie du solde B
```

```
        if(send(sock2,soldeB,sizeof(soldeB),0) != sizeof(soldeB))
                printf("Echec de l'envoie des donnees soldeB !\n");
    }
// !  envoie des donnees



// fermeture du socket
        shutdown(sock2,2);
        close(sock2); // ferme le socket
// ! fermeture du 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)
{

    int nb_entrees = 0;
    nb_entrees =  table->getnb_entrees();

    string idA_;
    idA_.assign(idcompteA);
    string nomA_;
    nomA_.assign(nomA);
    string prenomA_;
    prenomA_.assign(prenomA);
    string mdpA_;
    mdpA_.assign(mdpA);
    string soldeA_;

    string idB_;
    idB_.assign(idcompteB);
    string nomB_;
    nomB_.assign(nomB);
    string prenomB_;
    prenomB_.assign(prenomB);
    string mdpB_;
    mdpB_.assign(mdpB);
    string soldeB_;
```

```
    double somme_ = atof(somme);
    double soldedA, soldedB;

    int verif=0;

    for(int i=0; i<nb_entrees;i++)
    {

       // comparaison de l'id A
       if (idA_ == table[i].getid() && nomA_ == table[i].getnom() && prenomA_ == table[i].getprenom() &&
mdpA_ == table[i].getmdp() )
       {
//           printf("identification reussie id A\n");
           verif = table[i].changesolde_controle(-somme_);
           if(verif != -1) // on verifie que le solde A est bien positif
           {
                   soldedA = verif;
                   sprintf( soldeA, "%f", soldedA);
           }
//           printf("solde A = %s\n",soldeA);
       }
    }

    if(verif != -1)
    {
        for(int i=0; i<nb_entrees;i++)
        {
           // comparaison de l'id B
           if (idB_ == table[i].getid() && nomB_ == table[i].getnom() && prenomB_ == table[i].getprenom() &&
mdpB_ == table[i].getmdp() )
           {
    //           printf("identification reussie id B\n");
              soldedB = table[i].changesolde(somme_);
              sprintf( soldeB, "%f", soldedB);
    //           printf("solde B = %s\n",soldeB);
           }
        }
    }

    if(verif != -1)
        return 0;
```

```
    else
        return -1;

}


int control2(char *idcompte, char *nom, char *prenom, char *mdp, char *solde, table_comptes *table)
{
    int nb_entrees = 0;
    nb_entrees =  table->getnb_entrees();

    string id_;
    id_.assign(idcompte);
    string nom_;
    nom_.assign(nom);
    string prenom_;
    prenom_.assign(prenom);
    string mdp_;
    mdp_.assign(mdp);
    string solde_;

    for(int i=0; i<nb_entrees;i++)
    {
       // comparaison de l'id
       if (id_ == table[i].getid() && nom_ == table[i].getnom() && prenom_ == table[i].getprenom() && mdp_
== table[i].getmdp() )
       {
          printf("identification reussie\n");
          sprintf( solde, "%f", table[i].getsolde());
//          printf("solde = %s\n",solde);

          return 0;
       }
    }


    return -1;
}
```

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#define CLE 666

struct msg
{
    char    nom[100];
    char    prenom[100];
    char    idcompte[100];
    char    somme[100];
};


void etape3(table_comptes *table)
{
    struct msg message; // creation de la structure de messages qui seront envoyes

// La primitive msgget () permet de créer ou de retrouver une file de message à partir de sa clé.
    int res; // sert de test
    int qid_file; // file de messages
    qid_file = msgget((key_t)CLE, 0700 | IPC_CREAT);
    if (qid_file == -1)
    {
             perror("msgget");
             return (EXIT_FAILURE);
    }

    struct msqid_ds buffer;
    int nb_messages;

    int nb_entrees = 0; // nb de fiches dans la base de donnees

    string id_;
    string nom_;
    string prenom_;
    double somme_;
```

```
char ladate1[10],ladate2[10]; // contient la date
 string date1,date2;

_strdate(ladate1);
_strdate(ladate2);
 date1.assign(ladate1);
 date2.assign(ladate2);

 while(1)
 {
        // delai d'attente de 5 secondes pour ne pas prendre trop de ressources
        Sleep(5000);

        // recuperation de la date
        _strdate(ladate2);
        date2.assign(ladate2);

        if(date1 != date2) // changement de jour
        {
               _strdate(ladate1);
               date1.assign(ladate1);

               // on recupere le nb de messages dans la file de messages
               nb_messages = msgctl(qid_file, IPC_STAT, &buffer);

               // on recupere les messages
               for(int i=0; i<nb_messages;i++)
               {
                      res=msgrcv(qid_file, &message, sizeof(struct msg)-4, 0, 0);
                      if(res == -1)
                             printf("erreur\n");

                      nb_entrees =  table->getnb_entrees();
                      id_.assign(buffer.idcompte);
                      nom_.assign(buffer.nom);
                      prenom_.assign(buffer.prenom);
                      string solde_;
                      somme_= atof(buffer.somme);

                      // on recherche dans la base de donnees la fiche qui nous interesse
                      for(int i=0; i<nb_entrees;i++)
```

```
                        {
                                // comparaison de l'id
                                if (id_ == table[i].getid() && nom_ == table[i].getnom() && prenom_ ==
table[i].getprenom() )
                                {
                                     // on effectue le changement du solde
                                     table[i].changesolde(somme_);
                                }
                        }
                } // fin du if
            }
     } // fin du while

}



#include "table_comptes.h"


// 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
{
```

```
        table_comptes *table;       // adresse table
        int    socket;   // adresse socket
};


DWORD WINAPI Thread_service_consultation(LPVOID lpParameter)
{
/* Partie 2 - Activation du service de consultation */
   table_comptes *table;
   table= (table_comptes *)lpParameter;
   etape2_consultation(table);
/* 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 */
   table_et_socket* infos;
   infos = (table_et_socket*) basedonnees_et_socketclient;
   table_comptes *table;
   table= infos->table;
   int socket2;
   socket2 = infos->socket;
   etape2_consultation2(table,socket2);
/* Fin de la partie 2 de consultation avec connexion client */
}

DWORD WINAPI Thread_service_virement(LPVOID lpParameter)
{
/* Partie 2 - transfert argent */
   table_comptes *table;
   table= (table_comptes *)lpParameter;
   etape2_transfert(table);
/* 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 */
   table_et_socket* infos;
   infos = (table_et_socket*) basedonnees_et_socketclient;
```

```
   table_comptes *table;
   table= infos->table;
   int socket2;
   socket2 = infos->socket;
   etape2_transfert2(table,socket2);
/* Fin de la partie 2 de virement avec connexion client */
}


int main(int argc, char* argv[])
{
    // pointeur vers les threads
    DWORD threadID;


/* Partie 1 - On charge en memoire la table des comptes */
    printf("Demarrage du sereur bancaire\n\n");
    printf("On commence par charger en memoire la table des comptes\n");
    table_comptes *table = NULL;
    int nb_entrees=0;
    FILE *f = NULL;
    f = fopen ( "la_table_des_comptes" , "r" );
    if (f == NULL)
    {
        printf("Erreur de lecture de la base de donnees\n");
        exit(-1);
    }
    fscanf(f, "%d", &nb_entrees);
    table = new table_comptes[nb_entrees];
    table->setnb_entrees(nb_entrees);
    fclose(f);
    etape1(table, nb_entrees);
printf("Fin du chargement en memoire la table des comptes\n\n");
/* Fin de la partie 1 */


printf("Creation du thread service consultation\n");
CreateThread(NULL, 0, Thread_service_consultation, table, 0, &threadID);

printf("Creation du thread service virement\n");
CreateThread(NULL, 0, Thread_service_virement, table, 0, &threadID);
```

```
/* Partie 3 - analyse periodique */
   printf("Demarrage du service periodique\n");
   etape3(table);
/* Fin de la partie 3 analyse periodique */


    system("PAUSE");
    delete table; // suppression des donnees en memoire
    return 0; // EXIT_SUCCESS
}


#include "service_creation_base_de_donnees.cpp"
#include "service_consultation.cpp"
#include "service_transfert.cpp"
#include "service_periodique.cpp"
```