```
#ifndef BANQUE_H
#define BANQUE_H

#include "Compte.H"
#include "ListeCompte.H"

class Banque {
 protected:
  ListeCompte* comptes;

  // Renvoie le compte dont le nom du propriétaire est login.
  // Renvoie NULL s'il n'y en a pas.
  Compte* rechercherCompte(char* login);

  // Lance la procedure d'ouverture de compte pour un proprietaire
  // de nom login, si ce nom n'est pas deja utilise.
  void ouvrirCompte(char* login);

  // Lance la consultation du compte dont le nom de propriétaire
  // est login s'il existe.
  void consulterCompte(char* login);

  // Supprime le compte dont le nom du proprietaire est login
  // s'il existe.
  void supprimerCompte(char* login);

  // Liste le nom des proprietaires des comptes.
  void listerComptes();

 public:
  // Crée une nouvelle banque.
  Banque();
  ~Banque();

  // Met la banque en service.
  void demarrer();
};

#endif /* BANQUE_H */
```

```
#ifndef COMPTE_AVEC_DECOUVERT_H
#define COMPTE_AVEC_DECOUVERT_H

#include "Compte.H"

// Compte permettant d'avoir un certain decouvert.

class CompteAvecDecouvert : public Compte {

private:
  int decouvert_maximum;

public:
  CompteAvecDecouvert(char* nom_proprietaire,int decouvert_max);

  // Retrait jusqu'au découvert autorisé
  Reponse retrait (int somme);

};

#endif /* COMPTE_AVEC_DECOUVERT_H */
```

```
#ifndef COMPTE_AVEC_JOURNAL_H
#define COMPTE_AVEC_JOURNAL_H

#include <fstream.h>

#include "Compte.H"

// Compte enregistrant les operations effectuées dans un fichier journal.

class CompteAvecJournal : public Compte{

private:
  ofstream journal;

public:
  CompteAvecJournal(char* nom_proprietaire,char* fichier_journal);

  Reponse depot (int somme);
  Reponse retrait (int somme);
  int solde_courant ();
};

#endif /* COMPTE_AVEC_JOURNAL_H */
```

```
#ifndef COMPTE_H
#define COMPTE_H

class Compte {
protected:
  int solde;

public:
  enum Reponse { rep_Ok, rep_Somme_Negative, rep_Solde_Insuffisant };
  char nom[80];

  Compte (char proprietaire[]);
  virtual ~Compte();

  virtual Reponse depot (int somme);
  virtual Reponse retrait (int somme);
  virtual int solde_courant ();
};

#endif /* COMPTE_H */
```

```
#ifndef LISTE_COMPTE_H
#define LISTE_COMPTE_H

#include "Compte.H"

class Element {
public:
  Compte* c;
  Element* suivant;

  Element(Compte* c,Element* suivant);
};

class IterateurCompte {
private:
  Element* courant;
public:
  IterateurCompte(Element* premier);

  bool aUnSuivant();
  Compte* suivant();
};

class ListeCompte {
private:
  Element* premier;
  void supprimerEnPartantDe(Compte* c,Element* &courant);
public:
  ListeCompte();
  ~ListeCompte();
  void ajouter(Compte* c);
  void supprimer(Compte* c);

  IterateurCompte* iterateur();
};

#endif /* LISTE_COMPTE_H */
```

```
#include <fstream>

#include "Banque.H"
#include "CompteAvecDecouvert.H"
#include "CompteAvecJournal.H"

Banque::Banque() {
  comptes = new ListeCompte();
}

Banque::~Banque() {
  delete comptes;
}

Compte* Banque::rechercherCompte(char* login){
  IterateurCompte* it = comptes->iterateur();
  while (it->aUnSuivant()){
    Compte* c = it->suivant();
    if (strcmp(c->nom,login) == 0)
      return c;
  }
  return NULL;
}

void Banque::ouvrirCompte(char* login){
  char commande[10];
  int decouvert_max;

  if (rechercherCompte(login) != NULL){
    cout << "ERREUR : le nom de login " << login << " est déjà utilisé !" << endl;
  } else {
    cout << "Créer un compte avec (d)écouvert ou avec (j)ournal [d,j]? : " << flush;
    cin >> commande;
    switch (commande[0]){
    case 'd': case 'D':
      cout << "Découvert maximum : " << flush;
      cin >> decouvert_max;
      if (decouvert_max < 0) {
        cout << "ERREUR : le decouvert doit être positif !" << endl;
      }
      comptes->ajouter(new CompteAvecDecouvert(login,decouvert_max));
      break;
    case 'j': case 'J':
      comptes->ajouter(new CompteAvecJournal(login,login));
      break;
    default:
      cout << "ERREUR : commande inconnue '" << commande[0] << "'" << endl;
    }
  }
}

void affiche_reponse (Compte::Reponse reponse) {
  switch (reponse) {
  case Compte::rep_Ok :
    cout << "ok" << endl;
    break;

  case Compte::rep_Somme_Negative :
    cout << "erreur: somme négative !" << endl;
    break;

  case Compte::rep_Solde_Insuffisant:
```



```
    cout << "erreur: solde insuffisant !" << endl;
    break;

  default:
    cout << "erreur interne: code de retour inconnu (" << (int)reponse << ")" << endl;
    break;
  }
}

void Banque::consulterCompte(char* login){
  Compte* c = rechercherCompte(login);
  if (c == NULL){
    cout << "ERREUR : il n'y a pas de client avec le nom : " << login << endl;
  } else {
    char operation;
    int somme;
    bool termine = false;

    cout << "Vous travaillez sur le compte de " << c->nom << endl;
    cout << "Tapez : " << endl;
    cout << "s  pour connaître le solde courant," << endl;
    cout << "d  pour effectuer un dépôt," << endl;
    cout << "r  pour effectuer un retrait, et" << endl;
    cout << "t  pour terminer." << endl << endl;

    while (! termine) {
      cout << "Commande: " << flush;
      cin >> operation;
      switch (operation) {
      case 'd' : case 'D' :
        cout << "Somme à déposer: " << flush;
        cin >> somme;
        affiche_reponse (c->depot (somme));
        break;

      case 'r': case 'R':
        cout << "Somme à retirer: " << flush;
        cin >> somme;
        affiche_reponse (c->retrait (somme));
        break;

      case 's': case 'S':
        cout << "Solde courant: " << c->solde_courant () << endl;
        break;

      case 't': case 'T':
        cout << "Au revoir !" << endl;
        termine = true;
        break;

      default:
        cout << "Commande inconnue `" << operation << "'" << endl;
        break;
      }
    }
  }
}

void Banque::supprimerCompte(char* login){
  Compte* c = rechercherCompte(login);
  if (c == NULL){
    cout << "ERREUR : il n'y a pas de client avec le nom : " << login << endl;
```

```
    return;
  }
  else {
    comptes->supprimer(c);
    cout << "OK : compte de " << login << " supprime !" << endl;
  }
}

void Banque::listerComptes(){
  cout << endl;
  cout << "LISTE DES COMPTES" << endl;
  cout << "-----------------" << endl;
  IterateurCompte* it = comptes->iterateur();
  while (it->aUnSuivant())
    cout << "  " << it->suivant()->nom << endl;
}

void Banque::demarrer(){
  char commande;
  char login[80];

  while (true){
    cout << endl;
    cout << "------- BANQUE -------" << endl;
    cout << "1. ouvrir un compte" << endl;
    cout << "2. consulter un compte" << endl;
    cout << "3. supprimer un compte" << endl;
    cout << "4. lister les comptes" << endl;
    cout << "5. quitter la banque" << endl;
    cout << endl;
    cout << "choix [1,2,3,4]? : " << flush;

    cin >> commande;

    switch (commande){
    case '1': // creer un compte
      cout << "login : " << flush;
      cin >> login;
      ouvrirCompte(login);
      break;
    case '2': // consulter un compte
      cout << "login : " << flush;
      cin >> login;
      consulterCompte(login);
      break;
    case '3': // supprimer un compte
      cout << "login : " << flush;
      cin >> login;
      supprimerCompte(login);
      break;
    case '4': // liste les comptes
      listerComptes();
      break;
    case '5': // quitter le programme
      exit(0);
    default:
      cout << "ERREUR : commande inconnue '" << commande << "'" << endl;
    }
  }
}
```

```
#include "CompteAvecDecouvert.H"

CompteAvecDecouvert::CompteAvecDecouvert(char* nom_proprietaire,int decouvert_ma
x)
  : Compte(nom_proprietaire)
{
  this->decouvert_maximum = decouvert_max;
}

Compte::Reponse CompteAvecDecouvert::retrait (int somme) {
  if (somme < 0)
    return rep_Somme_Negative;
  else if (somme > (solde + decouvert_maximum))
    return rep_Solde_Insuffisant;
  else {
    solde -= somme;
    return rep_Ok;
  }
}
```

```
#include <fstream.h>

#include "CompteAvecJournal.H"

CompteAvecJournal::CompteAvecJournal(char* nom_proprietaire,char* fichier_journa
l)
  : Compte(nom_proprietaire)
{
 journal.open(fichier_journal,ios::out);
 if (!journal)
   cout << "ERREUR : Ouverture de " << fichier_journal << " ratée !" << endl;
}

char* chainePourReponse(Compte::Reponse reponse) {
  switch (reponse) {
  case Compte::rep_Ok: return "ok";
  case Compte::rep_Somme_Negative: return "erreur: somme negative";
  case Compte::rep_Solde_Insuffisant: return "erreur: solde insuffisant";
  default: return "REPONSE INCONNUE";
  }
}

Compte::Reponse CompteAvecJournal::depot (int somme) {
  journal << "Opération: dépôt de " << somme << " CHF" << endl;
  Reponse reponse = Compte::depot(somme);
  journal << "Résultat : " << chainePourReponse(reponse) << endl;
  journal << endl;
  return reponse;
}

Compte::Reponse CompteAvecJournal::retrait (int somme) {
  journal << "Opération: retrait de " << somme << " CHF" << endl;
  Reponse reponse = Compte::retrait(somme);
  journal << "Résultat : " << chainePourReponse(reponse) << endl;
  journal << endl;
  return reponse;
}

int CompteAvecJournal::solde_courant () {
  journal << "Opération: demande du solde" << endl;
  int solde = Compte::solde_courant();
  journal << "Résultat : " << solde  << " CHF" << endl;
  journal << endl;
  return solde;
}
```

```
#include <string.h>

#include "Compte.H"

Compte::Compte (char proprietaire[]){
  strncpy (this->nom, proprietaire, 80);
  this->nom[79] = '\0';
  this->solde = 0;
}

Compte::~Compte (){
}

Compte::Reponse Compte::depot (int somme) {
  if (somme < 0)
    return rep_Somme_Negative;
  else {
    solde += somme;
    return rep_Ok;
  }
}

Compte::Reponse Compte::retrait (int somme) {
  if (somme < 0)
    return rep_Somme_Negative;
  else if (somme > solde)
    return rep_Solde_Insuffisant;
  else {
    solde -= somme;
    return rep_Ok;
  }
}

int Compte::solde_courant () {
  return solde;
}
```

```
#include "ListeCompte.H"
#include <stdlib.h>

// Implantation de la classe Element

Element::Element(Compte* c,Element* suivant){
  this->c = c;
  this->suivant = suivant;
}

// Implantation de la classe IterateurCompte

IterateurCompte::IterateurCompte(Element* premier){
  this->courant = premier;
}

bool IterateurCompte::aUnSuivant(){
  return courant != NULL;
}

Compte* IterateurCompte::suivant(){
  if (courant == NULL)
    return NULL;
  else {
    Compte* c = courant->c;
    courant = courant->suivant;
    return c;
  }
}

// Implantation de la classe ListeCompte

ListeCompte::ListeCompte(){
  premier = NULL;
}

void ListeCompte::ajouter(Compte *c){
  Element* cont = new Element(c,premier);
  premier = cont;
}

void ListeCompte::supprimerEnPartantDe(Compte* c,Element* &courant){
  ///- A COMPLETER
  if (courant == NULL)
    return;
  else
    if (courant->c == c){
      Element* temp = courant;
      courant = courant->suivant;
      delete temp;
      return;
    }
    else
      supprimerEnPartantDe(c,courant->suivant);
  ///+
}

void ListeCompte::supprimer(Compte* c){
  ///- A COMPLETER
  supprimerEnPartantDe(c,premier);
  ///+
}
```



```
IterateurCompte* ListeCompte::iterateur(){
  return new IterateurCompte(premier);
}

ListeCompte::~ListeCompte(){
  Element* courant = premier;

  while (courant != NULL){
    Element* temp = courant;
    courant = courant->suivant;
    delete temp;
  }
}
```

```
#include "Banque.H"

int main(){
  Banque* banque = new Banque();
  banque->demarrer();
  delete banque;
}
```