```cpp
#include <iostream>
#include <iomanip>
#include <string.h>
using namespace std;

class Montant
{
public :
    Montant ();
    Montant (int eur, int cen);

        int getEuros () const           { return euros; }
        int getCentimes () const        { return centimes; }
        void setEuros (int eur)         { euros = eur; }
        void setCentimes (int cen)      { centimes = cen; }

    friend ostream& operator<< (ostream& flot, const Montant& mont);
    friend istream& operator>> (istream& flot, Montant& mont);

    Montant operator+ (const Montant& mont2) const;
    Montant operator- (const Montant& mont2) const;

    bool operator== (const Montant& mont2) const;
    bool operator> (const Montant& mont2) const;

private :
    int euros;
    int centimes;
};

Montant::Montant ()
{
    euros = 0;
    centimes = 0;
}

Montant::Montant (int eur, int cen)
{
    euros = eur;
    centimes = cen;
}

ostream& operator<< (ostream& flot, const Montant& mont)
{
    flot << mont.euros << "," << setfill('0') << setw(2) << mont.centimes
        << " euros";
    return flot;
}

istream& operator>> (istream& flot, Montant& mont)
{
    flot >> mont.euros >> mont.centimes;
    return flot;
}

Montant Montant::operator+ (const Montant& mont2) const
{
    int sommeCen = centimes + mont2.centimes;

    // calcul des euros et centimes du resultat, sachant que la somme des
    // centimes peut depasser 100
    int eurResult = euros + mont2.euros + (sommeCen / 100);
    int cenResult = sommeCen % 100;

    Montant montResult (eurResult, cenResult);
    return montResult;
}

Montant Montant::operator- (const Montant& mont2) const
{
    int eurResult, cenResult;
```

```
    // le 2eme Montant est plus grand que le 1er
    if (mont2 > *this)
    {
        eurResult = 0;
        cenResult = 0;
    }
    else
    {
        if (centimes >= mont2.centimes)
        {
            eurResult = euros - mont2.euros;
            cenResult = centimes - mont2.centimes;
        }
        // cas ou les centimes du 2eme sont > au centimes du 1er
        else
        {
            eurResult = euros - mont2.euros - 1;
            cenResult = centimes - mont2.centimes + 100;
        }
    }

    Montant montResult (eurResult, cenResult);
    return montResult;
}

bool Montant::operator== (const Montant& mont2) const
{
    return (euros == mont2.euros && centimes == mont2.centimes);
}

bool Montant::operator> (const Montant& mont2) const
{
    return (euros > mont2.euros)
           || (euros == mont2.euros && centimes > mont2.centimes);
}

class Compte
{
public :
    Compte (const char* n);

    virtual void crediter (const Montant& somme);
    bool debiter (const Montant& somme);

    virtual void afficher (bool retourLigne = true) const;

    Montant getSolde () const     { return solde; }

        bool comparerNom (const char* nom2) const;
        bool comparerNom (const Compte& compte2) const;

private :
    char nom[50];
    Montant solde;
};

// dans la liste d'initialisation : appel du constructeur de l'objet membre
// solde
Compte::Compte (const char* n)
    : solde(0,0)
{
    strcpy (nom, n);
}

void Compte::crediter (const Montant& somme)
{
    solde = solde + somme;
}

bool Compte::debiter (const Montant& somme)
{
    bool debitFait;
```

```
    if (somme > solde)
        debitFait = false;
    else
    {
        solde = solde - somme;
        debitFait = true;
    }

    return debitFait;
}

void Compte::afficher (bool retourLigne) const
{
    cout << "nom : " << nom << "/ solde : " << solde;
    if (retourLigne)
        cout << endl;
}

bool Compte::comparerNom (const char* nom2) const
{
        return strcmp (nom, nom2) == 0;
}

bool Compte::comparerNom (const Compte& compte2) const
{
        return comparerNom (compte2.nom);
}

class CompteLimite : public Compte
{
public :
    CompteLimite (const char* n, const Montant& lim);
    void crediter (const Montant& somme);
    void afficher (bool retourLigne = true) const;

private :
    Montant limite;
};

// dans la liste d'initialisation : appel du constructeur de la classe mere et
// de celui de l'objet membre limite (constructeur par copie)
CompteLimite::CompteLimite (const char* n, const Montant& lim)
    : Compte(n), limite(lim)
{
}

void CompteLimite::crediter (const Montant& somme)
{
    // on credite si l'operation ne fait pas depasser la limite
    if ( ! (getSolde() + somme > limite) )
                Compte::crediter (somme);
}

void CompteLimite::afficher (bool retourLigne) const
{
    // affichage des donnees heritees de Compte
    Compte::afficher(false);

    cout << "/ limite : "  << limite;

    if (retourLigne)
        cout << endl;
}

const int MAX_COMPTES = 100;

class Banque
{
public :
        Banque ();
        bool ajouter (Compte* pCompte);
```

```
        void afficher () const;
        void crediter (const char* nom, const Montant& somme);
        bool debiter (const char* nom, const Montant& somme);

private :
    // tableau de pointeurs Compte* qui peuvent pointer sur des objets Compte
    // ou des objets CompteLimite
        Compte* comptes[MAX_COMPTES];

    // nb de comptes ajoutes dans la banque
        int nbComptes;

        // cherche un compte par son nom et retourne son indice ou -1 si non trouvé
        int chercher (const char* nom) const;

    // teste si un compte a deja son nom dans la banque (retour de true)
        bool testerNomCompte (const Compte& leCompte) const;
};

Banque::Banque ()
{
        nbComptes = 0;
}

bool Banque ::ajouter (Compte* pCompte)
{
    bool ajoutFait;

    // ajout du compte s'il y a de la place et si son nom n'existe pas ds deja
        if (nbComptes < MAX_COMPTES && testerNomCompte(*pCompte) == false)
        {
                comptes[nbComptes] = pCompte;
                nbComptes++;
        ajoutFait = true;
        }
    else
        ajoutFait = false;

    return ajoutFait;
}

void Banque::afficher () const
{
    cout << "BANQUE" << endl;

        for (int i = 0; i < nbComptes; i++)

        // il faut que la fonction afficher de Compte soit virtuelle pour que
        // l'appel ci-dessous appelle la fonction afficher de Compte si on
        // pointe sur un Compte ou celle de CompteLimite si on pointe sur un
        // CompteLimite
                comptes[i]->afficher();

    cout << "****************" << endl;
}

void Banque::crediter (const char* nom, const Montant& somme)
{
        int ind;

    // on credite si on a trouve le nom dans la banque
        if ( (ind = chercher(nom)) >= 0 )

        // la fonction crediter de Compte doit etre virtuelle (idem afficher)
                comptes[ind]->crediter (somme);
}

bool Banque::debiter (const char* nom, const Montant& somme)
{
    bool debitFait = false;
        int ind;
```

```
    // on debite si on a trouve le nom dans la banque
        if ( (ind = chercher(nom)) >= 0 )
                debitFait = comptes[ind]->debiter (somme);

    return debitFait;
}

int Banque::chercher (const char* nom) const
{
        int i = 0;

        while (i < nbComptes && comptes[i]->comparerNom(nom) == false)
                i++;

        int retour;
        if (i < nbComptes)      // trouvé
                retour = i;
        else
                retour = -1;
        return retour;
}


bool Banque::testerNomCompte (const Compte& leCompte) const
{
        int i = 0;

        while (i < nbComptes && comptes[i]->comparerNom(leCompte) == false)
                i++;

        return i < nbComptes;
}

int main ()
{
    // TEST Montant
    /*
    Montant mont1, mont2;
    char oper[10], choix[10];
    do
    {
        cout << "entrer Montant operation(+-==>) Montant :" << endl;
        cin >> mont1 >> oper >> mont2;
        if (strcmp(oper,"+") == 0)
            cout << mont1 + mont2 << endl;
        else if (strcmp(oper,"-") == 0)
            cout << mont1 - mont2 << endl;
        else if (strcmp(oper,"==") == 0)
            if (mont1==mont2)cout<<"egaux"<<endl;else cout<<"non egaux"<<endl;
        else if (strcmp(oper,">") == 0)
            if (mont1>mont2) cout << ">" <<endl; else cout << "non >" <<endl;
        cout << "autre operation (o pour oui) ? ";
        cin >> choix;
    } while (strcmp(choix,"o") == 0);
    */

    // TEST COMPTE

    Compte c1 ("toto");
    Montant s1 (150,50);
    c1.crediter (s1);
    c1.afficher();

    /*
    Montant s2;
    cout << "somme a debiter : ";
    cin >> s2;
    if (c1.debiter(s2)==true) cout<<"FAIT"<<endl; else cout<<"NON FAIT"<<endl;
    c1.afficher();
    */

    cout << "solde = " << c1.getSolde() << endl;
```

```
    Compte c2("toto"), c3("titi");
    if (c1.comparerNom("toto") == true && c1.comparerNom("titi") == false
        && c1.comparerNom(c2) == true && c1.comparerNom(c3) == false)
        cout << "comparerNom ok" << endl;

    // TEST COMPTE LIMITE

    Montant lim1(1000,0), lim2(2000,0);
    CompteLimite cl1 ("titi1", lim1);
    CompteLimite cl2 ("titi2", lim2);
    Montant s3(500,70), s4(800,80), s5(1800,0);;
    cl1.crediter (s3);
    cl2.crediter (s4);
    cl1.afficher();
    cl2.afficher();
    cl2.crediter (s5);
    cl2.afficher();

    // TEST BANQUE

        Banque ban;
        ban.ajouter (&cl1);
        ban.ajouter (&c1);
        bool aj1 = ban.ajouter (&cl2);
        Compte c4 ("titi1");
        bool aj2 = ban.ajouter (&c4);
    if (aj1 == true && aj2 == false)
        cout << "retour ajouter ok" << endl;

        ban.afficher();

        ban.crediter ("titi2", Montant(100,0));
        ban.crediter ("toto", Montant(1000,0));
        ban.crediter ("titi2", Montant(1000,0));
        ban.crediter ("titi2", Montant(1000,0));
        ban.afficher();

    bool deb1 = ban.debiter ("toto", Montant(2000,0));
    bool deb2 = ban.debiter ("titi2", Montant(1000,0));
    if (deb1 == false && deb2 == true)
        cout << "retour debiter ok" << endl;

    return 0;
}
```