```cpp
// testA.C

#include <iostream>
using namespace std;

const int MAX_NB_VAL = 100;

class ListeVal
{
public :
        ListeVal ();
        void ajouter (int valeur);
        int getVal (int indice) const;
        int getNbVal () const          { return nbVal; }
        int nbOccurrences (int valeur) const;
        void afficher () const;

private :
        int val [MAX_NB_VAL];
        int nbVal;
};

ListeVal::ListeVal ()
{
    nbVal = 0;
}

void ListeVal::ajouter (int valeur)
{
    // on vérifie qu'on n'a pas atteint le nombre maximal de valeurs
    if (nbVal < MAX_NB_VAL)
    {
        val[nbVal] = valeur;
        nbVal++;
    }
}

int ListeVal::getVal (int indice) const
{
    // en toute rigueur il faudrait vérifier la validité de l'indice
    return val [indice];
}

int ListeVal::nbOccurrences (int valeur) const
{
    int nb = 0;

    for (int i = 0; i < nbVal; i++)
    {
        if (val[i] == valeur)
            nb++;
    }

    return nb;
}

void ListeVal::afficher () const
{
    for (int i = 0; i < nbVal; i++)
        cout << " " << val[i];

    cout << endl;
}

class ListeSD : public ListeVal
{
public :
    ListeSD ();
    void ajouter (int valeur);
    bool contient (int valeur) const;

private :
```

```
};

// dans la liste d'initialisation, appel du constructeur de la classe "mère"
// ListeVal qui initialise la liste à vide
ListeSD::ListeSD ()
    : ListeVal()
{
}

void ListeSD::ajouter (int valeur)
{
    // on ajoute la valeur seulement si elle n'est pas déjà dans l'ensemble;
    // pour ajouter on utilise la fonction ajouter de ListeVal dont on écrit
    // le nom complet pour la différencier de la fonction ajouter de ListeSD
    if (!contient(valeur))
        ListeVal::ajouter (valeur);
}

bool ListeSD::contient (int valeur) const
{
    return nbOccurrences(valeur) != 0;
}

class EnsMath : public ListeSD
{
public :
    EnsMath ();
    EnsMath reunion (const EnsMath& ens2) const;
    EnsMath inter (const EnsMath& ens2) const;
    bool inclusDans (const EnsMath& ens2) const;
    EnsMath operator+ (const EnsMath& ens2) const;
    EnsMath operator- (const EnsMath& ens2) const;
    bool operator== (const EnsMath& ens2) const;

private :
};

// dans la liste d'initialisation, appel du constructeur de la classe "mère"
// ListeSD qui initialise l'ensemble à vide
EnsMath::EnsMath ()
    : ListeSD()
{
}

EnsMath EnsMath::reunion (const EnsMath& ens2) const
{
    EnsMath ensReunion;

    int nbval = getNbVal();
    int nbval2 = ens2.getNbVal();

    // on ajoute à l'ensemble résultat les valeurs de deux ensembles;
    // la fonction ajouter se charge de refuser les doublons

    for (int i = 0; i < nbval; i++)
        ensReunion.ajouter (getVal(i));

    for (int i = 0; i < nbval2; i++)
        ensReunion.ajouter (ens2.getVal(i));

    return ensReunion;
}

EnsMath EnsMath::inter (const EnsMath& ens2) const
{
    EnsMath ensInter;

    int nbval = getNbVal();

    // chaque valeur de l'ensemble est ajoutée au résultat si elle appartient
    // au deuxième ensemble
    for (int i = 0; i < nbval; i++)
```

```
    {
        int v = getVal(i);
        if (ens2.contient(v))
            ensInter.ajouter (v);
    }

    return ensInter;
}

bool EnsMath::inclusDans (const EnsMath& ens2) const
{
    int nbval = getNbVal();

    // test de l'appartenance de chaque élément au deuxième ensemble
    bool estInclus = true;
        for (int i = 0; i < nbval && estInclus == true; i++)
    {
        if ( !ens2.contient(getVal(i)) )
            estInclus = false;
    }

    return estInclus;
}

EnsMath EnsMath::operator+ (const EnsMath& ens2) const
{
    // le résultat est la réunion de l'ensemble courant et de ens2
    return reunion(ens2);
}

EnsMath EnsMath::operator- (const EnsMath& ens2) const
{
    EnsMath ensDiff;

    int nbval = getNbVal();

    // chaque valeur de l'ensemble est ajoutée au résultat si elle n'appartient
    // pas au deuxième ensemble
    for (int i = 0; i < nbval; i++)
    {
        int v = getVal(i);
        if (!ens2.contient(v))
            ensDiff.ajouter (v);
    }

    return ensDiff;
}

bool EnsMath::operator== (const EnsMath& ens2) const
{
    // les deux ensembles sont égaux s'ils ont même nombre d'éléments et si
    // l'un est inclus dans l'autre
    return getNbVal() == ens2.getNbVal() && inclusDans(ens2);
}

int main ()
{
        // test ListeVal
        cout << "TEST LISTE_VAL" << endl;
        ListeVal lv;
        lv.ajouter (5);
        lv.ajouter (7);
        lv.ajouter (5);
        lv.afficher();

        cout << "valeur a indice 1 : " << lv.getVal(1)
                << ", nb val : " << lv.getNbVal()
                << ", nb occurrences de 5 : " << lv.nbOccurrences(5) << endl;

        // test ListeSD
        cout << "\nTEST LISTE_SD" << endl;
        ListeSD lsd;
```

```
        lsd.ajouter (5);
        lsd.ajouter (7);
        lsd.ajouter (5);
        lsd.afficher();
        if (lsd.contient(7) && !lsd.contient(6))
                cout << "contient ok" << endl;

        // test EnsMath
        cout << "\nTEST ENS_MATH" << endl;
        EnsMath ens1, ens2, ens3, ens4;
        for (int i = 1; i <= 10; i++) ens1.ajouter (i);
        for (int i = 7; i <= 15; i++) ens2.ajouter (i);
        for (int i = 1; i <= 5; i++) ens3.ajouter (i);
        for (int i = 10; i >= 1; i--) ens4.ajouter (i);

        cout << "E1 : "; ens1.afficher();
        cout << "E2 : "; ens2.afficher();
        cout << "E3 : "; ens3.afficher();
        cout << "E4 : "; ens4.afficher();

        EnsMath ensInt = ens1.inter (ens2);
        EnsMath ensReu = ens1.reunion (ens2);
        cout << "\nreunion E1 E2 : "; ensReu.afficher();
        cout << "intersection E1 E2 : "; ensInt.afficher();

        if (ens3.inclusDans(ens1) && !ens2.inclusDans(ens1))
                cout << "\ninclusDans ok" << endl;

        EnsMath ensSomme = ens1 + ens2;
        cout << "\nE1 + E2 : "; ensSomme.afficher();
        EnsMath ensDiff = ens1 - ens2;
        cout << "E1 - E2 : "; ensDiff.afficher();

        if (ens1 == ens4 && !(ens1 == ens3))
                cout << "\noperateur == ok" << endl;

        return 0;
}
```