Durée : 3 h

1) Ecrire la classe Collection pour représenter une collection de valeurs entières.

```
const int NB_MAX = 100;

class Collection
{
public :
    constructeur construisant une collection vide

    fonction ajoutant une valeur à la collection

    fonction retournant le nombre de valeurs

    fonction retournant la valeur d'indice i

    fonction vidant la collection

    opérateur << d'affichage

private :
    int val [NB_MAX];    // les valeurs de la collection
    int nb;              // nombre de valeurs ajoutées à la collection (varie de 0 à NB_MAX)
};
```

Les questions 2 et 3 sont indépendantes l'une de l'autre et peuvent être réalisées dans n'importe quel ordre.

2) Ecrire la classe CollectionOrdonnee héritant de Collection : une collection ordonnée est une collection dont les valeurs sont ordonnées (par ordre croissant par exemple).

class CollectionOrdonnee **hérite de Collection**
{
public :
constructeur construisant une collection ordonnée vide

redéfinition de la fonction d'**ajout** de Collection (même nom et prototype) afin d'ordonner les valeurs

private :
};

3) Ecrire la classe Ensemble héritant de Collection : un ensemble est une collection dans laquelle il n'y a pas de doublon (chaque valeur est unique) et qu'on munit des opérations sur ensembles.

class Ensemble **hérite de Collection**
{
public :

Constructeurs :
- constructeur construisant un ensemble **vide**
- constructeur initialisant l'ensemble avec les **valeurs d'un tableau d'entiers**; ex d'utilisation :
  int tab [ ] = { 3, 9, 2, 8, 4, 3, 12 };
  Ensemble ens (tab, 7); en paramètres : adresse du tableau et nombre de valeurs; noter qu'il y a plusieurs fois la valeur 3, mais l'ensemble ne doit la stocker qu'une fois

Ajout d'élément :
redéfinition de la fonction d'**ajout** de Collection (même nom et prototype) afin de ne pas avoir de doublon

Tests et comparaisons :
- fonction qui teste si l'ensemble **contient un élément** passé en paramètre
- fonction qui teste si l'ensemble **est un sur-ensemble** d'un d'autre ensemble passé en paramètre
- fonction qui teste si l'ensemble **est inclus** dans un autre ensemble passé en paramètre
- fonction qui teste si l'ensemble **est vide**
- opérateurs == et != de **comparaison** de deux ensembles

Opérations sur ensembles :
- opérateur + : **réunion** de deux ensembles
- opérateur ^ : **intersection** de deux ensembles
- opérateur - : **différence** de deux ensembles; le résultat est un ensemble contenant les éléments du premier qui ne sont pas dans le deuxième

private :
};

```
// exam2004.C

#include <iostream>
using namespace std;

const int NB_MAX = 100;

class Collection
{
public :
    Collection ();

    void ajouter (int valeur);

    int getNbVal () const;
    int getVal (int indice) const;

    void vider ();

    friend ostream& operator<< (ostream& flot, const Collection& col);

private :
    int val [NB_MAX];
    int nb;
};

Collection::Collection ()
{
    nb = 0;
}

void Collection::ajouter (int valeur)
{
    // ajout de la valeur si la liste n'est pas pleine
    if (nb < NB_MAX)
    {
        val[nb] = valeur;
        nb++;
    }
}

int Collection::getNbVal () const
{
    return nb;
}

int Collection::getVal (int indice) const
{
    int valeur = 0;

    if (indice < nb)    // protection contre indice trop grand
```

```
        valeur = val [indice];
    return valeur;
}

void Collection::vider ()
{
    nb = 0;
}

// affichage sous la forme {3,6,14,3,7}
ostream& operator<< (ostream& flot, const Collection& col)
{
    flot << "{";
    for (int i = 0; i < col.nb; i++)
    {
        flot << col.val[i];
        if (i < col.nb - 1)
            flot << ",";
    }
    flot << "}";

    return flot;
}

class CollectionOrdonnee : public Collection
{
public :
    CollectionOrdonnee ();

    void ajouter (int valeur);

private :
};

/* appel du constructeur de la classe mère Collection qui initialise à la
 * collection vide
 */
CollectionOrdonnee::CollectionOrdonnee ()
    : Collection()
{
}

void CollectionOrdonnee::ajouter (int valeur)
{
    // on recupere dans le tableau les_val les valeurs de la collection
    int les_val [NB_MAX];
    int nbval = getNbVal();
    for (int i = 0; i < nbval; i++)
        les_val[i] = getVal(i);

    // on vide la collection
```

40000	45000	50000

```
    vider();

    // on ajoute les valeurs inferieures à la nouvelle valeur
    int j;
    for (j = 0; j < nbval && les_val[j] <= valeur; j++)
        Collection::ajouter (les_val[j]);

    // on ajoute la nouvelle valeur
    Collection::ajouter (valeur);

    // on ajoute les valeurs superieures à la nouvelle valeur
    for (; j < nbval; j++)
        Collection::ajouter (les_val[j]);
}

class Ensemble : public Collection
{
public :
    Ensemble ();
    Ensemble (const int* tab, int n);

    void ajouter (int elem);

    bool contient (int elem) const;

    bool estSurEnsemble (const Ensemble& ens2) const;
    bool estInclus (const Ensemble& ens2) const;

    bool estVide () const;

    bool operator== (const Ensemble& ens2) const;
    bool operator!= (const Ensemble& ens2) const;

    Ensemble operator+ (const Ensemble& ens2) const;
    Ensemble operator^ (const Ensemble& ens2) const;
    Ensemble operator- (const Ensemble& ens2) const;
};

Ensemble::Ensemble ()
    : Collection()
{
}

Ensemble::Ensemble (const int* tab, int n)
    : Collection()
{
    /* pour ajouter : appel de la fonction ajouter de Ensemble qui teste
les
     * valeurs multiples
     */
    for (int i = 0; i < n; i++)
```

40000	45000	50000

```
        ajouter (tab[i]);
    }

    void Ensemble::ajouter (int elem)
    {
        /* l'element est ajoute seulement si l'ensemble ne le contient pas
    deja
         */
        if (!contient(elem))
            Collection::ajouter (elem);
    }

    bool Ensemble::contient (int elem) const
    {
        /* recherche de l'element dans les valeurs de l'ensemble
         */
        int nbVal = getNbVal();
        int i = 0;
        while (i < nbVal && getVal(i) != elem)
            i++;

        // si i < nbVal : on a trouve l'element
        return (i < nbVal);
    }

    bool Ensemble::estSurEnsemble (const Ensemble& ens2) const
    {
        /* on teste si tous les elements de ens2 sont contenus dans
    l'ensemble
         */
        int nbVal2 = ens2.getNbVal();
        int i = 0;
        while ( i < nbVal2  &&  contient(ens2.getVal(i)) )
            i++;

        /* si i == nbVal2 : on n'a trouve aucun element de ens2 non contenu
    dans
         * l'ensemble
         */
        return (i == nbVal2);
    }

    bool Ensemble::estInclus (const Ensemble& ens2) const
    {
        /* l'ensemble courant (*this) est inclus dans ens2 si ens2 en est un
         * sur-ensemble
         */
        return ens2.estSurEnsemble(*this);
    }

    bool Ensemble::estVide () const
```

```
{
    return (getNbVal() == 0);
}

bool Ensemble::operator== (const Ensemble& ens2) const
{
    /* les deux ensembles sont egaux s'ils ont memes nombres d'elements
et si
     * l'un est inclus dans l'autre
     */
    return (getNbVal() == ens2.getNbVal() && estInclus(ens2));
}

bool Ensemble::operator!= (const Ensemble& ens2) const
{
    // appel de l'operateur == precedent
    return !(*this == ens2);
}

Ensemble Ensemble::operator+ (const Ensemble& ens2) const
{
    /* initialisation de l'ensemble somme avec le 1er ensemble */
    Ensemble ensSomme = *this;

    /* ajout a l'ensemble somme des elements de ens2, la fonction
ajouter se
     * chargeant de tester les valeurs multiples
     */
    int nbVal2 = ens2.getNbVal();
    for (int i = 0; i < nbVal2; i++)
        ensSomme.ajouter (ens2.getVal(i));

    return ensSomme;
}

Ensemble Ensemble::operator^ (const Ensemble& ens2) const
{
    Ensemble ensInter;  // ensemble resultat vide

    /* parcours des elements du 1er ensemble et ajout a l'intersection
s'ils
     * sont contenus dans le 2eme
     */
    int nbVal = getNbVal();
    for (int i = 0; i < nbVal; i++)
    {
        int val = getVal(i);
        if (ens2.contient(val))
            ensInter.ajouter (val);
    }
```

```
    return ensInter;
}

Ensemble Ensemble::operator- (const Ensemble& ens2) const
{
    Ensemble ensDiff;   // ensemble resultat vide

    /* parcours des elements du 1er ensemble et ajout au resultat s'ils
     * ne sont pas contenus dans le 2eme
     */
    int nbVal = getNbVal();
    for (int i = 0; i < nbVal; i++)
    {
        int val  = getVal(i);
        if (!ens2.contient(val))
            ensDiff.ajouter (val);
    }

    return ensDiff;
}

int main ()
{
    Collection c1;
    c1.ajouter (2);
    c1.ajouter (5);
    c1.ajouter (2);
    c1.ajouter (7);
    cout << c1 << endl;
    cout << "nb val=" << c1.getNbVal()
            << ", 1ere val=" << c1.getVal(0)
            << ", derniere val=" << c1.getVal(c1.getNbVal()-1) << endl;
    c1.vider();
    cout << c1 << endl;

    CollectionOrdonnee co1;
    co1.ajouter (4);
    co1.ajouter (8);
    co1.ajouter (4);
    co1.ajouter (9);
    co1.ajouter (6);
    cout << co1 << endl;

    Ensemble e1;

    int t2[] = { 3, 9, 2, 8, 4, 3, 12};
    Ensemble e2 (t2, sizeof(t2)/sizeof(int));

    cout << "e1 vide=" << e1 << ", e2=" << e2 << endl;

    Ensemble e3 = e2;
```

```
    e3.ajouter (20);

    int t4[] = { 3, 9, 15};
    Ensemble e4 (t4, sizeof(t4)/sizeof(int));

    if (e2.contient(8) && !e2.contient(10))
        cout << "contient element ok" << endl;

    if (e3.estSurEnsemble(e2) && !e3.estSurEnsemble(e4))
        cout << "sur-ensemble ok" << endl;

    if (e2.estInclus(e3) && !e4.estInclus(e3))
        cout << "inclus ok" << endl;

    if (e1.estVide())
        cout << "estVide ok" << endl;

    Ensemble e5 = e2;
    e5.ajouter (3);
    e5.ajouter (9);

    int t6[] = { 3, 9, 2, 8, 4, 3, 16};
    Ensemble e6 (t6, sizeof(t6)/sizeof(int));

    if (e5 == e2 && e6 != e2)
        cout << "== et != ok" << endl;

    int t7[] = { 5, 2, 9, 10 };
    Ensemble e7 (t7, sizeof(t7)/sizeof(int));
    cout << "e7 : " << e7 << endl;

    cout << "e2 + e7 : " << (e2 + e7) << endl;
    cout << "e2 ^ e7 : " << (e2 ^ e7) << endl;
    cout << "e2 - e7 : " << (e2 - e7) << endl;

    return 0;
}
```

1) Ecrire les classes Depl et Point : Point sert à représenter les points du plan, Depl sert simplement à exprimer un déplacement (dx,dy) à ajouter à un point.

```
class Depl
{
public :
    un constructeur

    deux fonctions retournant
    respectivement dx et dy

private :
    int dx, dy;
};
```

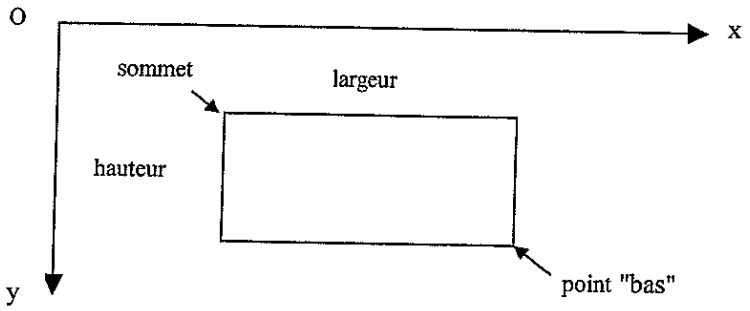
```
class Point
{
public :
    un constructeur sans paramètre initialisant à (0,0)
    un constructeur recevant les coordonnées
    deux fonctions retournant respectivement x et y

    opérateur + ajoutant un déplacement (un objet Depl) à
    un point, le résultat étant un point

    opérateur – faisant la différence entre 2 points, le
    résultat étant un déplacement (un objet Depl)

private :
    int x, y;
};
```

2) Ecrire la classe Rect pour représenter des rectangles de cotés parallèles aux axes de coordonnées.




```
class Rect
{
public :
        un constructeur recevant 4 entiers (les coordonnées du sommet, la largeur et la hauteur)
        un constructeur recevant 2 objets Point (le sommet et le bas)

        une fonction simulant le dessin en affichant les coordonnées du sommet, la largeur et la hauteur

        une fonction qui reçoit un point et qui teste si le point est à l'intérieur du rectangle

        une fonction qui permet de déplacer le rectangle (sans modifier ses dimensions) : elle reçoit deux entiers
        (dx et dy) représentant le déplacement que doit subir le sommet

        une fonction permettant de redimensionner le rectangle, c'est à dire modifier sa largeur et sa hauteur

private :
        Point sommet;
        int largeur, hauteur;
};
```

3) Ecrire la classe Fenetre pour représenter des fenêtres d'écran : une fenêtre est un rectangle possédant un titre.

class Fenetre hérite de Rect
{
public :
un constructeur : il reçoit en paramètre seulement le titre, toute fenêtre devant être créée avec un sommet en (200,200), une largeur de 100 et une hauteur de 50

redéfinition de la fonction de dessin (même nom et prototype que celle de Rect) pour dessiner le rectangle de la fenêtre et afficher son titre

private :
char titre[50];
};

4) Ecrire la classe BoiteMsg pour représenter des boites d'affichage de message, une boite de message étant une fenêtre avec un message.

class BoiteMsg hérite de Fenetre
{
public :
un constructeur recevant en paramètres le titre de la boite et le texte de son message;
il faut donner à la boite des dimensions initiales de 40x20

redéfinition de la fonction de dessin (même nom et prototype que celle de Fenetre) pour dessiner la fenêtre de la boite et afficher son message

private :
char msg[100]; // texte du message
};

```
//*****************
//Vincent Hélie 2A
//*****************

//Declaration des en tetes
#include<iostream.h>
#include <string.h>

//using namespace std;


//Question 1

class Depl
{
public:

  Depl(int DX,int DY); //Constructeur

  //Fonction(s) membre(s)

        int getValX();
        int getValY();

private:

  //variables,Constantes et prototypes

        int dx,dy;

};

Depl::Depl(int DX,int DY)
{
        dx=DX;
        dy=DY;
}
int Depl::getValX()
{
        return dx;
}
int Depl::getValY()
{
        return dy;
}

class Point
{
public:

        Point();
        Point(int X,int Y);

        int getValX();
        int getValY();

        Point operator+ (const Depl& D1)const;
        Depl operator- (const Point& P1)const;

private:
                int x,y;
};

Point::Point()
{
        x=0;
        y=0;
```

```
}
Point::Point(int X,int Y)
{
        x=X;
        y=Y;
}
int Point::getValX()
{
        return x;
}
int Point::getValY()
{
        return y;
}
Point Point::operator + (const Depl& D1)const
{
        Point P1;

        P1.getValX()=D1.getValX() + getValX();
        P1.getValY()=D1.getValY() + getValY();

        return P1;

}
Depl Point::operator - (const Point& P1)const
{
//      int dx;
//      int dy;

//      dx=P1.getValX()-getValX();
//      dy=P1.getValY()-getValY();
//      Depl D1(dx,dy);

        Depl D1(P1.getValX()-getValX(),P1.getValY()-getValY());

        return D1;
}


//Question 2

class Rect
{
public:

        Rect(int sx,int sy,int larg,int haut);
        Rect(Point& Som,Point& Bas);
        void afficherObj();
        bool TestPoint(int x,int y);
        void DeplaRect(int dx,int dy);
        void ModiRect(int larg,int haut);

private:
        Point sommet;
        int largeur,hauteur;
};

Rect::Rect(int sx,int sy,int larg,int haut):sommet(sx,sy)
{
                largeur=larg;
                hauteur=haut;
}
Rect::Rect(Point& Som,Point& Bas) //calcul de la hauteur et de la largeur
{
        largeur=Bas.getValX()-Som.getValX();
        hauteur=Bas.getValY()-Som.getValY();
}
void Rect::afficherObj()
```

```
{
        cout<<"Le sommet a pour coordonnee :
"<<sommet.getValX()<<sommet.getValY()<<endl;
        cout<<"La largeur est"<<largeur<<endl;
        cout<<"La longueur est"<<hauteur<<endl;
}

bool Rect::TestPoint(int x,int y)
{

        if(x<sommet.getValX() || y>sommet.getValY())
        {
                return 0;
        }
        else if (x>largeur + sommet.getValX() || y<hauteur + sommet.getValY())
        {
                return 0;
        }
        else
                return 1;

}
/*void Rect::DeplaRect(int dx,int dy):sommet(dx,dy)
{

}*/
void Rect::ModiRect(int larg,int haut)
{
        largeur=larg;
        hauteur=haut;
}

//Question 3

class Fenetre:public Rect
{
public:
        Fenetre(char* Titre);
        void afficherObj();

private:

                char titre[50];
};


void Fenetre::afficherObj()
{
        Rect::afficherObj();
                cout<<"Le titre est: "<<titre<<endl;
}


/*Fenetre::Fenetre(char* Titre)    //Constructeur a un probleme???????? ERROR
{
        strcpy(titre,Titre);
}*/


class BoiteMsg:public Fenetre
{
public:

        BoiteMsg(char* Titre,char* Msg);
        void afficherObj();


private:
```

```
        char msg[100];
};


int main()  //pas le temps de faire le main (8 erreurs identiques!!!!)
{

//      Depl D1;
//      Point P1;
//      Fenetre F1;
//      BoiteMsg BM;


        return 0;
}
```

1) Ecrire la classe Heure pour représenter des heures et minutes :

class **Heure**
{
public :
un constructeur sans paramètre initialisant à 0h0
un constructeur recevant l'heure et la minute

un opérateur >= pour comparer deux objets Heure

un opérateur + pour ajouter deux objets Heure, ex. : 10h30 + 5h40 = 16h30
pour simplifier on considèrera que les objets Heure à additionner ne font jamais dépasser 23h59

un opérateur - pour faire la différence entre deux objets Heure, ex : 12h30 - 3h40 = 8h 50
pour simplifier on considérera que le 1er objet Heure est toujours postérieur au 2ème

un opérateur << d'affichage

private :
int h, m; // heure (0-23) et minute (0-59)
};

2) Ecrire la classe Creneau pour représenter des créneaux horaires :

class **Creneau**
{
public :
un constructeur recevant 4 entiers : heure et minute de début, heure et minute de fin

une fonction d'affichage

des fonctions getDebut et getFin retournant les Heure de début et de fin
des fonctions setDebut et setFin modifiant les Heure de début et de fin

une fonction recevant 2 entiers (heure et minute) et testant s'ils sont dans la plage horaire du créneau

private :
Heure debut, fin;
};

3) Ecrire la classe Reunion héritant de Creneau et permettant de représenter des réunions :

class **Reunion** hérite de Creneau
{
public :
un constructeur recevant le sujet et 3 entiers (heure de début, minute de début, durée) sachant que les réunions durent toujours un nombre entier d'heures

une fonction d'affichage affichant le sujet de la réunion et son horaire

une fonction *void decaler (int dh)* permettant de décaler la réunion du nombre d'heures dh, qui peut être positif ou négatif, sans changer sa durée

private :
char sujet [50]; // sujet de la réunion
};

```
#include<iostream>
#include<string.h>


// Test de rattrapage du 06 septembre 2004
// à compiler sous LINUX

using namespace std;

// question n°1

class Heure
{
public:
        Heure();
        Heure(int heure,int minute);
        bool operator>= (const Heure& heure);
        Heure operator+ (const Heure& heure);
        Heure operator- (const Heure& heure);
        friend ostream& operator<< (ostream& flot,const Heure& heure);
private:
        int h,m;
};

Heure::Heure()
{
h=0;
m=0;
}

Heure::Heure(int heure,int minute)
{
h=heure;
m=minute;
}

bool Heure::operator>= (const Heure& heure)
{
        if (h > heure.h)
                return true;
        else
                if (h<heure.h)
                        return false;

        if(h == heure.h)
                {
                        if (m>=heure.m)
                return true;
                        else
                return false;
                }
}



// je suppose pour les deux operateurs que la première heure saisie est ultérieur

Heure Heure::operator+ (const Heure& heure)
{

int heur,minu,reste;
heur = h + heure.h;
minu = m + heure.m;
```

```
if((minu > 60) && (heur < 24))
        {
                reste = minu - 60;
                heur = heur + 1;
        }
        Heure heu(heur,reste);
return heu;
}

Heure Heure::operator- (const Heure& heure)
{
int heur,minu,reste;
heur = h - heure.h;
minu = m - heure.m;
if(( minu < 0) && ( heur >0))
        {
        reste = m + (60 - heure.m);
        heur= heur - 1;
        }
        Heure heu(heur,reste);
return heu;


}

ostream& operator<< (ostream& flot,const Heure& heure)
{
flot<<heure.h<<":"<<heure.m<<"\n";
return flot;
}


// question n°2

class Creneau
{
public:
        Creneau(int heur_deb,int min_deb,int heur_fin,int min_fin);
        friend ostream& operator<< (ostream& flot,const Creneau& cre);
        Heure getDebut();
        Heure getFin();
        void setDebut(Heure& heur);
        void setFin(Heure& heur);
        bool teste(int heure,int minute);
private:
        Heure debut,fin;
};

Creneau::Creneau(int heur_deb,int min_deb,int heur_fin,int min_fin)
{
        Heure heure_debut(heur_deb,min_deb);
        Heure heure_fin(heur_fin,min_fin);
        debut=heure_debut;
        fin=heure_fin;
}

ostream& operator<<(ostream& flot,const Creneau& cre)
{
        flot<<cre.debut<<"et"<<cre.fin;
        return flot;
}


Heure Creneau::getDebut()
```

```
{
        return debut;
}

Heure Creneau::getFin()
{
        return fin;
}

void Creneau::setDebut(Heure& heur)
{
        debut=heur;
}

void Creneau::setFin(Heure& heur)
{
        fin=heur;
}

bool Creneau::teste(int heure,int minute)
{
        Heure heur(heure,minute);
        if (heur >= debut)
        "l'heure vaut :"        return true;
        else
                return false;
}

// question n°3

class Reunion:public Creneau
{
//Creneau(int heur_deb,int min_deb,int heur_fin,int min_fin);
public:
        Reunion(char* suj,int heure_deb,int min_deb,int duree);
        void affichage();
        //void decaler(int dh);
private:
        char sujet[50];
};

Reunion::Reunion(char* suj,int heure_deb,int min_deb,int duree):Creneau(heure_deb
{
        strcpy(sujet,suj);
}

void Reunion::affichage()
{

cout<<"le sujet de la reunion est : "<<sujet<<endl;
}

/*void Reunion::decaler(int dh)
{

if (dh > 0)
        debut=debut + dh;
else"l'heure vaut :"
        debut=debut - dh;

cout<<"la reunion est decalee pour :"<<debut<<"\n";
}*/
```

```
int main(void)
{

//      affichage des heures
        Heure heur1(12,25);
        cout<<"Il est actuellement :"<<" "<<heur1<<endl;
        Heure heur2(14,15),heur3(5,10);
        cout<<"heur2 + heur3 = "<<heur2+heur3<<endl;
        cout<<"heur2 - heur3 = "<<heur2-heur3<<endl;

        if (heur2>=heur3)
                cout<<" heure2 >= heure3"<<endl;
        else
                cout<<"heure2 <= heure3"<<endl;
"l'heure vaut :"
//  affichage des données du creneau
        Creneau c1(8,5,10,40);
        cout<<"c1: "<<c1<<endl;
        cout<<c1.getDebut()<<"et"<<c1.getFin()<<endl;
        c1.setDebut(heur3);
        c1.setFin(heur2);
        cout<<c1.getDebut()<<"et"<<c1.getFin()<<endl;

// teste de l'appartenance à la plage horaire du creneau
        if (c1.teste(12,20))
                cout<<"l'heure est bien dans la plage horaire du creneau"<<endl;
        else
                cout<<"l'heure est hors de la plage horaire du creneau"<<endl;

        if (c1.teste(2,50))
                cout<<"l'heure est bien dans la plage horaire du creneau"<<endl;
        else
                cout<<"l'heure est hors de la plage horaire du creneau"<<endl;

// affichage des données de la réunion
cout<<" "<<endl;
        char* s;
        s="bourse";
        Reunion r(s,14,55,2);
        r.affichage();
        cout<<"heure de debut"<< r.getDebut()<<" et heure de fin "<<r.getFin()<<e

return 1;
}
```