```
/* exemple cours C++ : classe point (dont hériteront les vecteurs et torseurs) */
/*                                Patrick TRAU   IPST-ULP  Novembre 04        */
/*                         la suppression de la ligne précédente est illicite ! */

#include <iostream.h>
class point
 {
 private :  //la structure interne d'un vrai objet n'a pas a être publique
  float x,y,z;

 protected : //a la rigueur les héritiers peuvent accéder directement aux
             //attributs mais ne savent pas comment ils sont gérés
  void setx(float a=0) {x=a;}
  void sety(float a=0) {y=a;}
  void setz(float a=0) {z=a;}
  void set(float a=0,float b=0,float c=0) {x=a;y=b;z=c;}

 public :
//constructeur
  point(float a=0,float b=0,float c=0) {x=a;y=b;z=c;}
//accesseurs
  float getx(void) {return(x);}
  float gety(void) {return(y);}
  float getz(void) {return(z);}
  void raz(void) {x=y=z=0;}

// = ne peut être qu'une méthode, pas une fonction !
  point operator = (float f) {set(f,0,0);return *this;} //surtout prévu pour p=0

  void affiche(ostream &flux) //flux=fichier ou cout
      {flux<<"["<<x<<","<<y<<","<<z<<"]";}
  void saisie(istream &flux); //si le flux est cin on pose des questions
                              //sur cout, sinon on saisit sans question

  //on pourrait plutôt dire translater plutôt qu'additionner
  void additionner(float a,float b=0,float c=0)
      {x=a+x;y=b+y;z=c+z;}
  void additionner(point a)
      {x=x+a.x;y=y+a.y;z=z+a.z;}
  void soustraire(point a)
      {x=x-a.x;y=y-a.y;z=z-a.z;}

  //on pourrait dire dilater plutôt que multiplier
  void multiplier_par(float a) {x=a*x;y=a*y;z=a*z;}
  void multiplier_par(float a,float b,float c) {x=a*x;y=b*y;z=c*z;}
  void multiplier_par(point a) {x=a.x*x;y=a.y*y;z=a.z*z;}
 };

void point::saisie(istream &f)
 {
  if(f==cin)cout<<"entrez x : ";
  f>>x;
  if(f==cin)cout<<"entrez y : ";
  f>>y;
  if(f==cin)cout<<"entrez z : ";
  f>>z;
 }


//surcharges des opérateurs
ostream& operator << (ostream &f,point v)
 {v.affiche(f);return(f);}
istream& operator >> (istream &f,point &v)
 {v.saisie(f);return(f);}

point operator * (float f,point v)     //produit par un réel
    {point z=v;z.multiplier_par(f);return(z);}
point operator * (point v,float f)     //le prod par un float est commutatif !!!
    {return(f*v);}         //déjà défini dans l'autre sens, autant s'en servir !
point operator * (point v,point w)         //produit : est-ce utile (c'est une
           // homothétie ou l'on ne dilate pas d'autant dans chaque direction)
 {point z=v;z.multiplier_par(w);return z;}

point operator / (const point &v,float f)      //non commutatif
 {return(v*(1/f));}

point operator + (float f,const point &v)    //le float est ajoute a x seulement
 {point z=v;
  z.additionner(f);
  return(z);
 }
point operator + (point v,float f)          //commutatif !!!
 {return(f+v);}
point operator + (point v,point w)
 {point z=v;
  z.additionner(w);
  return(z);
 }
point operator - (point v,float f)       //j'accepte uniquement dans ce sens !
 {return(v+(-f));}
point operator - (point v,point w)
 {point z=v;
  z.soustraire(w);
  return(z);
 }

int operator == (point a,point b)
 {return ((a.getx()==b.getx())&&(a.gety()==b.gety())&&(a.getz()==b.getz()));}
int operator == (point a, float f) //surtout pour a==0
  {return ((a.getx()==f)&&(a.gety()==0)&&(a.getz()==0));}
int operator != (point a,point b)
 {return !(a==b);}
int operator != (point a, float b)
 {return !(a==b);}

/*exemple de programme de test ****************************
int main(void)
 {
  point v(1,2,3),w,z;
//tests *
  z=2*v;w=v*2;
  cout<<"point v : "<<v<<" *2= "<<w<<" ou "<<z<<"\n";
  cout<<"entrez vos nouvelles coordonnées :\n";
//test cin
  cin>>w;
  cout<<w<<" *(1,2,3) vaut "<<v*w<<"\n";
// test / et logique
  if((z/2==v)&&(z!=v))cout<<"tests réussis\n";
  w=1;
//additions
  cout<<v<<"+1="<<v+1<<" et 1+"<<v<<"="<<1+v<<"\n";
  cout<<"alors que"<<v<<"+"<<w<<"="<<v+w<<"\n";
  }
/*fin de « point.cpp »*************************************************/
```

```
/* la classe vecteur hérite du point ************************************/

// un vecteur est presque comme un point, mais les méthodes changent un peu
// évidement je ne réécris que les méthodes qui diffèrent de celles du point

class vecteur:public point
 {
//attributs : les trois coordonnées sont deja dans le point

/*** héritées,mises en commentaire pour se rappeler qu'elles sont accessibles
 protected :
  void setx(float a=0);
  void sety(float a=0);
  void setz(float a=0);
  void set(float a=0,float b=0,float c=0);
 public :
  float getx(void);
  float gety(void);
  float getz(void);
  void raz(void);
  void affiche(ostream &flux);
  void saisie(istream &flux);
  void additionner(float a);
  void additionner(float a,float b,float c=0);
  void additionner(point a);
  void additionner(vecteur a);
  void soustraire(point a);
  void multiplier_par(float a);
  void multiplier_par(float a,float b,float c);
  void multiplier_par(point a);
// surcharges de fonctions également héritées
ostream& operator << (ostream &f,point v)
istream& operator >> (istream &f,point &v)
// pour les autres opérations, bien que je revoie un point, ça marche encore !
point operator / (const point &v,float f);
point operator + (float f,const point &v);
point operator + (point v,float f);
point operator + (point v,point w);
point operator - (point v,float f);
point operator - (point v,point w);
int operator == (point a,point b);
int operator == (point a, float f); //surtout pour a==0
int operator != (point a,point b);
int operator != (point a, float b);
fin héritage*************************************************************/

 public :
  //le constructeur n'est pas hérité
  vecteur(float a=0,float b=0,float c=0) {set(a,b,c);}
  vecteur(point p) {set(p.getx(),p.gety(),p.getz());}
  //il est pratique de créer directement un vecteur a l'aide d'1 couple de points
  vecteur(point a,point b)
       {set(b.getx()-a.getx(),b.gety()-a.gety(),b.getz()-a.getz());}

  vecteur operator = (float f) {set(f,0,0);return *this;} //surtout pour p=0
  vecteur operator = (point p) {set(p.getx(),p.gety(),p.getz()); return *this;}

  float norme(void)
     {return(sqrt(getx()*getx()+gety()*gety()+getz()*getz()));}

  void normer(void)
     {float n=norme();setx(getx()/n);sety(gety()/n);setz(getz()/n);}

  float prodscal(vecteur v)
     {return(getx()*v.getx()+gety()*v.gety()+getz()*v.getz());}
 };

// le produit n'a pas a être surcharge, je VEUX le produit scalaire qui est
// différent de ce que faisais pour deux points !
// ce qui est bizarre est qu'il faut alors que je surcharge aussi pour la
// multiplication avec un float.
float operator * (vecteur v,vecteur w)      //produit scalaire
 {return v.prodscal(w);}
vecteur operator * (float f,vecteur v)      //produit par un reel
    {vecteur z=v;z.multiplier_par(f);return(z);}
vecteur operator * (vecteur v,float f)      // commutatif !!!
 {return(f*v);}    //je l'ai deja defini dans l'autre sens, autant s'en servir !

vecteur operator ^ (vecteur v,vecteur w)     //produit vectoriel
 {
  vecteur z(
            v.gety()*w.getz()-w.gety()*v.getz() ,
            v.getz()*w.getx()-w.getz()*v.getx() ,
            v.getx()*w.gety()-w.getx()*v.gety()
           );
  return(z);
 }

/exemple de test des vecteurs*********************************************
int main(void)
 {
  vecteur v(1,2,3),w;
  point A(1,1,1),O(0),B(1,0,0);
  vecteur z(O,A);
  cout<<"2*"<<v<<"="<<2*v<<", OA="<<z<<" (sur x:"<<z.getx()<<")\n";
  z.raz();cout<<"mis a 0 :"<<z<<"\n";
//tests *
  w=v*2;z=2*v;
  cout<<"vecteur v : "<<v<<" *2= "<<w<<" ou "<<z<<"\n";
  cout<<"entrez vos nouvelles coordonnees :\n";
//test cin
  cin>>w;
  cout<<w<<" *(1,2,3) vaut "<<v*w<<"\n";
//on affecte un vecteur par un bipoint
  w=vecteur(O,A); cout<<"le vecteur OA vaut "<<w<<" et sa norme "<<w.norme()
<<"\n";
// test / et logique
  if((z/2==v)&&(z!=v))cout<<"tests reussis\n";
  w=1;
//additions
  cout<<v<<"+1="<<v+1<<" et 1+"<<v<<"="<<1+v<<"\n";
  cout<<"alors que"<<v<<"+"<<w<<"="<<v+w<<"\n";
  cout<<"le prod scal de "<<v<<" et "<<w<<" est "<<v.prodscal(w)<<" (ou
"<<v*w<<")\n";
  z=2*(v^w);
  cout<<"le double de leur prod vect vaut "<<z<<"\n";
  }


 /*fin fichier « vecteur.cpp »*************************************/
```

```
/* la classe torseur hérite du point ************************************/

class torseur:public point  //c'est le point d'application qu'on hérite
 {
 private :  //la structure interne d'un vrai objet n'a pas a être publique
  vecteur res,mom;
   //ceux là on n'en hérite pas, on n'a donc pas accès au protected

 protected : //à la rigueur les héritiers peuvent accéder directemt aux attributs
  void setpoint(float a=0,float b=0,float c=0) {point::set(a,b,c);}
  void setpoint(point p) {point::set(p.getx(),p.gety(),p.getz());}
  void setresultante(float a=0,float b=0,float c=0) {res=vecteur(a,b,c);}
  void setresultante(vecteur v) {res=v;}
  void setmoment(float a=0,float b=0,float c=0) {mom=vecteur(a,b,c);}
  void setmoment(vecteur v) {mom=v;}
  void set(vecteur r,vecteur m, point a)
        {setresultante(r);setmoment(m);setpoint(a);}

 public :
//constructeur (plusieurs suivant les arguments)
  torseur(float ra=0,float rb=0,float rc=0,
          float ma=0,float mb=0,float mc=0,
          float  a=0,float  b=0,float  c=0)
     {setresultante(ra,rb,rc);setmoment(ma,mb,mc);point::set(a,b,c);}
     //attention en premier res puis mom puis le point d'application !
  torseur(vecteur r,vecteur m, point a)
        {setresultante(r);setmoment(m);setpoint(a);}

//accesseurs en lecture
/* hérités  :  float getx(void);float gety(void);float getz(void); */
  vecteur getpoint(void) {point p(getx(),gety(),getz());return p;}
  vecteur getresultante(void) {return res;}
  vecteur getmoment(void) {return mom;}

  void raz(void){res.raz();mom.raz();point::raz();}

// = ne peut etre qu'une methode, pas une fonction !
  vecteur operator = (float f) //uniquement prevu pour f=0
      {if(f==0)raz();else cout<<"affectation incompréhensible\n";return *this;}
  vecteur operator = (torseur t)
      {set(t.getresultante(),t.getmoment(),t.getpoint()); return *this;}

  void affiche(ostream &flux)
      {flux<<"{"<<res<<","<<mom<<" en "<<getx()<<","<<gety()<<","<<getz()<<"}";}
  void saisie(istream &f);

  void deplacer(point p)
  {
   ;//à implanter
  }
  void additionner(torseur a)
              //si au même point on additionne, sinon il faut d'abord déplacer
  {
     // {res=res+getresultante(a);mom=mom+getmoment(a);}
  }
 };

void torseur::saisie(istream &f)
 {
  if(f==cin)cout<<"resultante : ";
  f>>res;
  if(f==cin)cout<<"moment : ";
  f>>mom;
  if(f==cin)cout<<"point d'application : ";
  point::saisie(f);
 }

//surcharges opérateurs
ostream& operator << (ostream &f,torseur t)
 {t.affiche(f);return(f);}
istream& operator >> (istream &f,torseur &t)
 {t.saisie(f);return(f);}

torseur operator + (torseur v,torseur w)     //somme
 {
  torseur z=v;
  v.additionner(w);
  return(z);
 }

torseur operator - (torseur v,torseur w)     //différence vectorielle
 {return(v+((-1)*w));}

int operator == (torseur a,torseur b)
 {return ((a.getresultante()==b.getresultante())
        &&(a.getmoment()==b.getmoment())
        &&(a.getpoint()==b.getpoint()));
 }
int operator == (torseur a, float f) //uniquement pour f==0 : nul si
          //résultante et moment nuls, indépendamment du point d'application
  {return ((a.getresultante()==0)&&(a.getmoment()==0)&&(f==0));}

int operator != (torseur a,torseur b)
 {return !(a==b);}
int operator != (torseur a, float b)
 {return !(a==b);}


/*test torseurs*********************************************/
int main(void)
 {
  point O(0),X(1,0,0),Y(0,1,0),Z(0,0,1);
  vecteur v(1,1,0),w(2,-1,3),z(O,X);
  torseur t(v,w,X),u;
  cout<<"getresultante"<<t.getresultante()<<" getmoment"<<t.getmoment()
      <<" getpoint"<<t.getpoint()<<"\n";
  cin>>u;
  t=u;
  cout<<"verif:"<<t<<"\n";
 }
/*******************************************************/
```