```
//TAD resistencia
#include<iostream>
using namespace std;
class resistencia{
  float valor;
  float pmax;
  float tol;
  //funciones privadas, se usan solo desde
  //los operadores + y *
  float mayor(float a,float b) const;
  float menor(float a, float b) const;
public:
  resistencia (void);
  resistencia(float a);
  resistencia(float a,float b, float c);
  resistencia(const resistencia &a);
  void operator=(const resistencia &a);
  resistencia operator+(const resistencia &a) const;
  resistencia operator*(const resistencia &a) const;
};

float resistencia::mayor(float a,float b) const
{
  if (a>b) return a;
  return b;
}

float resistencia::menor(float a, float b) const
{
  if (a<b) return a;
  return b;
}

resistencia::resistencia (void){};

resistencia::resistencia(float a)
{
  valor=a;
  pmax=1;
  tol=5;
}

resistencia::resistencia(float a,float b, float c)
{
  valor=a;
  pmax=b;
  tol=c;
}

resistencia::resistencia(const resistencia &a)
{
  valor=a.valor;
  pmax=a.pmax;
  tol=a.tol;
}

void resistencia::operator=(const resistencia &a)
{
  valor=a.valor;
  pmax=a.pmax;
  tol=a.tol;
}

resistencia resistencia::operator+(const resistencia &a) const
{
  resistencia r;
  r.valor=valor+a.valor;
  r.pmax=r.valor*menor(pmax/valor,a.pmax/a.valor);
  r.tol=mayor(tol,a.tol);
  return r;
}

resistencia resistencia::operator*(const resistencia &a) const
{
  resistencia r;
  r.valor=valor*a.valor/(valor+a.valor);
  r.pmax=menor(pmax*valor,a.pmax*a.valor)/r.valor;
  r.tol=mayor(tol,a.tol);
  return r;
}

int main()
{
  resistencia r1(7,2,2),r2(3,1,2),r3(40),r4(10,2,5),r;
  r=r3*r4*(r1+r2);
}

#include <iostream>
using namespace std;
class cuaternio{
  float a,b,c,d;
public:
  cuaternio();
  cuaternio(float, float, float, float);
  //convierte real a cuaternio, se usa
  //en cuaternio op real de forma implicita
  //y en real op cuaternio de forma explicita
  cuaternio(float);
  cuaternio operator+(const cuaternio &) const;
  cuaternio operator-(const cuaternio &) const;
  cuaternio operator*(const cuaternio &) const;
  cuaternio operator/(const cuaternio &) const;
  cuaternio conjugado() const;
  float norma() const;
  friend cuaternio operator+(const float &, const cuaternio &);
  friend cuaternio operator-(const float &, const cuaternio &);
  friend cuaternio operator*(const float &, const cuaternio &);
  friend cuaternio operator/(const float &, const cuaternio &);
  friend ostream & operator<<(ostream &, const cuaternio &);
  friend istream & operator>>(istream &, cuaternio &);
};
```

```
cuaternio::cuaternio(){}

cuaternio::cuaternio(float u, float v, float w, float x)
{
  a=u;
  b=v;
  c=w;
  d=x;
}

//convierte real a cuaternio, se usa
//en cuaternio op real de forma implicita
//y en real op cuaternio de forma explicita
cuaternio::cuaternio(float u)
{
  a=u;
  b=0;
  c=0;
  d=0;
}

cuaternio cuaternio::operator+(const cuaternio &x) const
{
  return cuaternio(a+x.a,b+x.b,c+x.c,d+x.c);
}

cuaternio cuaternio::operator-(const cuaternio &x) const
{
  return cuaternio(a-x.a,b-x.b,c-x.c,d-x.c);
}

cuaternio cuaternio::operator*(const cuaternio &x) const
{
  return cuaternio(a*x.a-b*x.b-c*x.c-d*x.d,
                   a*x.b+b*x.a+c*x.d-d*x.c,
                   a*x.c-b*x.d+c*x.a+d*x.b,
                   a*x.d+b*x.c-c*x.b+d*x.a);
}

float cuaternio::norma() const
{
  return (a*a+b*b+c*c+d*d);
}

cuaternio cuaternio::conjugado() const
{
  return cuaternio(a,-b,-c,-d);
}



cuaternio cuaternio::operator/(const cuaternio &x) const
{
  return 1/x.norma()*x.conjugado()*cuaternio(a,b,c,d);
}

cuaternio operator+(const float &a, const cuaternio &x)
{
  return cuaternio(a)+x;
}

cuaternio operator-(const float &a, const cuaternio &x)
{
  return cuaternio(a)-x;
}

cuaternio operator*(const float &a, const cuaternio &x)
{
  return cuaternio(a)*x;
}

cuaternio operator/(const float &a, const cuaternio &x)
{
  return cuaternio(a)/x;
}

ostream & operator<<(ostream &os, const cuaternio &x)
{
  os<<x.a<<'+'<<x.b<<"i+"<<x.c<<"j+"<<x.d<<"k";
  return os;
}

istream & operator>>(istream &is, cuaternio &x)
{
  is>>x.a>>x.b>>x.c>>x.d;
  return is;
}

int main()
{
  cuaternio a,b;
  cout<<"Introduce un cuaternio:";
  cin>>a;
  cout<<"Introduce un cuaternio:";
  cin>>b;
  cout<<"+,-,*,/:"<<endl<<a+b<<endl<<a-b<<endl<<a*b<<endl<<a/b<<endl;
  float c;
  cout<<"Introduce un real:";
  cin>>c;
  //el real se convierte a cuaternio llamando a cuaternio(float)
  //despues se llama a cuaternio op cuaternio
  cout<<"Operaciones con el real por la derecha"<<endl;
  cout<<a+c<<endl<<a-c<<endl<<a*c<<endl<<a/c<<endl;
  //operadores friend
  cout<<"Operaciones con el real por la izqda"<<endl;
  cout<<c+a<<endl<<c-a<<endl<<c*a<<endl<<c/a<<endl;
}
```

```
#include<iostream>
#include<cmath>
class recta;
using namespace std;

//clase punto en dos dimensiones
class punto{
  float x,y;
public:
  punto(void);
  punto(float,float);
  //distancia entre dos puntos
  float distancia(const punto&)const;
  //recta puede acceder a los atributos de punto
  friend class recta;
  friend ostream& operator<<(ostream&,const punto&);
};

punto::punto(){}

punto::punto(float a,float b)
{
  x=a;
  y=b;
}

float punto::distancia(const punto &p) const
{
  return sqrt((x-p.x)*(x-p.x)+(y-p.y)*(y-p.y));
}

ostream&operator<<(ostream&o,const punto&p)
{
  o<<p.x<<' '<<p.y;
  return o;
}



//clase recta de la forma y=mx+n
class recta{
  float m,n;
public:
  recta(void);
  recta(float,float);
  //interseccion de dos rectas, devuelve un punto
  punto operator*(const recta &) const;
  //perpendicular a una recta por un punto
  //accede a los atributos de punto, por eso
  //recta es friend de punto
  recta perpendicular(const punto &) const;
  //distancia entre un punto y una recta
  //como distancia entre ese punto y la distancia
  //a la interseccion con la otra recta de la perpendicular
  //que pasa por ese punto
  //tambien se accede a los atributos de punto
  float distancia(const punto &) const;
  friend ostream&operator<<(ostream&,const recta&);
};

recta::recta(void){}
recta::recta(float pend,float ord)
{
  m=pend;
  n=ord;
}

punto recta::operator*(const recta &r) const
{
  float x;
  x=(r.n-n)/(m-r.m);
  return punto(x,r.m*x+r.n);
}

recta recta::perpendicular(const punto &p) const
{
  return recta(-1/m,p.y+1/m*p.x);
}

float recta::distancia(const punto &p) const
{
  if (m==0)
    return p.y-n;
  //si no se quiere usar this
  recta esta=recta(m,n);
  return p.distancia(esta.perpendicular(p)*esta);
  //usando this
  //return p.distancia(*this*this->perpendicular(p));
}

ostream&operator<<(ostream&o,const recta&r)
{
  o<<r.m<<' '<<r.n;
  return o;
}

main()
{
  recta r1(0,2),r2(1,1),r3(1,0);
  punto p;
  p=r1*r2;
  cout<<r1<<'*'<<r2<<'='<<p<<' '<<r3.distancia(p);
}
```