```
#include <iostream>
using namespace std;
//clase complejo representacion cartesiana
class complejo{
  float re,im;
public:
  complejo(void);
  complejo(float);
  complejo(float, float);
  complejo(const complejo &);
  complejo operator+(const complejo &) const;
  complejo operator*(const complejo &) const;
  void operator=(const complejo &);
  friend ostream & operator<<(ostream &,const complejo &);
  friend istream & operator>>(istream &,complejo &);
  friend complejo operator+(const float &, const complejo &
);
  friend complejo operator*(const float &, const complejo &
);
};

//constructor por defecto
complejo::complejo()
{
  re=0;
  im=0;
}

//constructor a partir de un real
//se utiliza de forma implicita
//en complejo op real
//convierte el real a complejo
complejo::complejo(float x)
{
  re=x;
  im=0;
}

//constructor a partir de dos reales
complejo::complejo(float a,float b)
{
  re=a;
  im=b;
}

//constructor de copia, si no se usa new no hace falta
complejo::complejo(const complejo &c)
{
  re=c.re;
  im=c.im;
}

//operador de asignacion, si no se usa new no hace falta
void complejo::operator=(const complejo &c)
{
  re=c.re;
  im=c.im;
}

//+ de dos complejos
complejo complejo::operator+(const complejo &c) const
{
  return complejo(c.re+re,c.im+im);
}

//* de dos complejos
complejo complejo::operator*(const complejo &c) const
{
  return complejo(re*c.re-im*c.im,re*c.im+im*c.re);
}

//operador de insercion
//accede a re e im, luego tiene que ser declarado friend
//en la clase
ostream & operator<<(ostream &salida, const complejo &c)
{
  salida<<c.re;
  if (c.im<0)
    salida<<c.im<<'i';
  else
    salida<<'+'<<c.im<<'i';
  return salida;
}

//operador de extraccion
//dual del anterior, mismas consideraciones
istream & operator>>(istream &entrada,complejo &c)
{
  entrada>>c.re>>c.im;
  return entrada;
}

//real+complejo
//en esta version, este operador accede a re e im,
//luego tiene que ser friend
//complejo operator+(const float &x, const complejo &c)
//{
//  return complejo(x+c.re,c.im);
//}

//en esta version, se convierte explicitamente
//el real a complejo y se llama a la suma de complejos
//como no se accede a los atributos de c, no haria falta
//declararlo friend en la clase
complejo operator+(const float &x, const complejo &c)
{
  return complejo(x)+c;
}

//similar al anterior
```

```
complejo operator*(const float &x, const complejo &c)
{
  return complejo(x)*c;
}

//no hace falta escribir los operadores
//complejo op real porque el real se convierte
//a complejo con el constructor a partir de real

int main()
{
  complejo a,b,c;
  //lectura de complejos
  cin>>a>>b;
  //complejo+complejo
  c=a+b;
  //complejo*complejo y muestra complejos
  cout<<c<<endl<<a*b<<endl;
  float x;
  cin>>x;
  //(1)
  //complejo+real, el real se convierte a complejo con comp
lejo(float);
  //despues se llama a complejo+complejo
  cout<<c+x<<endl;
  //(2)
  //real+complejo se llama al operador declarado friend
  cout<<x+c<<endl;
  //idem que (1) para el producto
  cout<<c*x<<endl;
  //idem que (2) para el producto
  cout<<x*c<<endl;
}
```

```
#include <iostream>
#include<cmath>
using namespace std;
//constante PI
const float PI=acos(-1.0);
//clase complejo representacion polar
//observar que la parte publica no cambia
//frente a la representacion en coordenadas polares
class complejo{
  float mod,arg;
public:
  complejo(void);
  complejo(float);
  complejo(float, float);
  complejo(const complejo &);
  complejo operator+(const complejo &) const;
  complejo operator*(const complejo &) const;
  void operator=(const complejo &);
  friend ostream & operator<<(ostream &,const complejo &);
  friend istream & operator>>(istream &,complejo &);
};
//arco tangente de cuatro cuadrantes
//se necesita para convertir de cartesianas a polares
float atan4c(float a, float b)
{
  if (b>0)
    if (a>0)
      return asin(b/sqrt(a*a+b*b));
    else
      return PI-asin(b/sqrt(a*a+b*b));
  else
    if (a>0)
      return 2*PI-asin(-b/sqrt(a*a+b*b));
    else
      return PI+asin(-b/sqrt(a*a+b*b));
}

complejo::complejo(void)
{
  mod=0;
  arg=0;
}

complejo::complejo(float x)
{
  if (x>0)
    {
      mod=x;
      arg=0;
    }
  else
    {
      mod=-x;
      arg=PI;
    }
}

//constructor a partir de dos reales,
//calcula las componentes de la representacion
//cartesiana
complejo::complejo(float a,float b)
{
  mod=sqrt(a*a+b*b);
  arg=atan4c(a,b);
}

//si no se usa new no es necesario
complejo::complejo(const complejo &c)
{
  mod=c.mod;
  arg=c.arg;
}

//si no se usa new no es necesario
void complejo::operator=(const complejo &c)
{
  mod=c.mod;
  arg=c.arg;
}

//suma en polares, se calculan y suman las
//componentes en cartesianas, despues se
//devuelve el complejo construido a partir
//de ellas
complejo complejo::operator+(const complejo &c) const
{
  float re,im;
  re=mod*cos(arg)+c.mod*cos(c.arg);
  im=mod*sin(arg)+c.mod*sin(c.arg);
  return complejo(re,im);
}

//el producto en polares es mas facil
complejo complejo::operator*(const complejo &c) const
{
  complejo res;
  res.mod=mod*c.mod;
  res.arg=arg+c.arg;
  return res;
}

//aunque internamente se usa la representacion en polares
//se muestran las componentes en coordenadas cartesianas
ostream & operator<<(ostream &salida, const complejo &c)
{
  salida<<c.mod*cos(c.arg);
  if (sin(c.arg)<0)
    salida<<c.mod*sin(c.arg)<<'i';
  else
    salida<<'+'<<c.mod*sin(c.arg)<<'i';
```

```
  return salida;
}

//aunque internamente se usa la representacion en polares
//se piden las componentes en coordenadas cartesianas
istream & operator>>(istream &entrada,complejo &c)
{
  float a,b;
  entrada>>a>>b;
  c.mod=sqrt(a*a+b*b);
  c.arg=atan4c(a,b);
  return entrada;
}

//real+complejo, se convierte explicitamente
//el real a complejo y se llama a la suma de complejos
complejo operator+(const float &x, const complejo &c)
{
  return complejo(x)+c;
}

//similar al anterior
complejo operator*(const float &x, const complejo &c)
{
  return complejo(x)*c;
}

//no hace falta escribir los operadores
//complejo op real porque el real se convierte
//a complejo con el constructor a partir de real

int main()
{
  complejo a,b,c;
  cin>>a>>b;
  c=a+b;
  cout<<c<<endl<<a*b<<endl;
  float x;
  cin>>x;
  cout<<c+x<<endl;
  cout<<x+c<<endl;
  cout<<c*x<<' '<<x*c<<endl;
}
```