```cpp
class C_asocia_D{

    public:
        typedef map<C*, D*> CD;
        typedef map<D*, set<C*>> DC;

        void asocia(C& c_, D& d_);
        void asocia(D& d_, C& c_);

        const D& asociado_con_C(C& c_) noexcept;
        const set<C*>& asociados_con_D(D& d_) noexcept;

    private:
        CD cd;
        DC dc;

};

void C_asocia_D::asocia(C& c_, D& d_)
{
    cd[&c_] = &d_;

    //Buscamos si el objeto c_ está asociodo con otro objeto d_
    for(DC::iterator i = dc.begin(); i != dc.end(); ++i){

        auto x = i->second.find(&c_);

        if(x != i->second.end()){     //No ha llegado a su fin -> existe un objeto c = &c_

            throw "Violación de multiplicidad";       //Si lo encuetra, lanzamos excepción
            i->second.erase(&c_);                     //Lo desasociamos del antiguo d_;
        }
    }
    /* Una vez buscada la asociación de c_ con otro objeto con d_,
     * lo asociamos con el nuevo  */
    dc[&d_].insert(&c_);
}

void C_asocia_D::asocia(D& d_, C& c_)
{

    asocia(c_,d_);
}

const D& C_asocia_D::asociado_con_C(C& c_) noexcept
{
    CD::iterator i = cd.find(&c_);
    return *i->second;
}

const set<C*>& C_asocia_D::asociados_con_D(D& d_) noexcept
{
    DC::iterator i = dc.find(&d_);
    return i->second;
}

```