Ejercicio 4 - Haskell

```
maximo ::  [Int] -> Int
maximo [x] = x
maximo (x:xs)
    | x > maximo xs = x
    | otherwise = maximo xs

media :: (Fractional a) => [a] -> a
media [x] = x
media (x:xs) = sum (x:xs) / fromIntegral (length (x:xs))

minimo :: (Ord a) => [a] -> a
minimo [x] = x
minimo (x:xs)
    | x < minimo xs = x
    | otherwise = minimo xs

suma :: (Num a) => [a] -> a
suma [x] = x
suma (x:xs) = x + suma xs

iguales :: (Ord a) => [a] -> Bool
iguales [] = False
iguales [_] = True
iguales (x:xs)
    | x == head xs && iguales xs = True
    | otherwise = False

iguales' :: (Ord a) => [a] -> Bool
iguales' [] = True
iguales' [_] = True
iguales' (x:y:xs)
    | x /= y = False
    | otherwise = iguales' xs

producto :: (Num a) => [a] -> a
producto [x] = x
producto (x:xs) = x * producto xs
```

Ejercicio 4 - C++

```
template <typename T>
T maximo(list<T> l) {
    T aux = T(0);
    for (auto i : l) {
        if (i > aux)aux = i;
    }
    return aux;
}
template <typename T>
T media(list<T> l) {
    T avg = T(0);
    for (auto i : l) {
        avg += i;
    }
    return avg/l.size();
}
template <typename T>
T minimo(list<T> l) {
    T aux = T(0);
    for (auto i : l) {
        if (i < aux)aux = i;
    }
    return aux;
}
template <typename T>
T suma(list<T> l) {
    T suma = T(0);
    for (auto i : l){
        suma += i;
    }
    return suma;
}
template <typename T>
T iguales(list<T> l){
    for (auto i = l.begin(); i < l.end(); i++) {
        for (auto j = l.next(i, 1); j < l.end(); j++) {
            if (*i != *j)  return false;
        }
    }
    return true;
}
```

```
template <typename T>
T producto(list<T> l) {
    T prod = T(0);
    for (auto i : l) {
        prod *= i;
    }
    return prod;
}
```

Ejercicio 5

```
type Coordenada = (Double,Double)
type Polilinea = [Coordenada]

esPoligono :: Polilinea -> Bool
esPoligono xs = head xs == last xs

esEquilatero :: Polilinea -> Bool
esEquilatero [] = True
esEquilatero [x,y] = True
esEquilatero (x:xs)
     | sqrt ((x1-x0)**2 + (y1-y0)**2)  /= sqrt ((x2-x1)**2 +
(y2-y1)**2)  = False
     | otherwise = esEquilatero xs
        where
            [(x0,y0),(x1,y1),(x2,y2)] = take 3 (x:xs)

esEquiangulo :: Polilinea -> Bool
esEquiangulo [] = True
esEquiangulo [x,y] = True
esEquiangulo (x:xs)
    | ang1 /= ang2 = False
    | otherwise = esEquiangulo xs
        where
            ang1 = acos (ladoIzq1 / ladoDer1)
            ang2 = acos (ladoIzq2 / ladoDer2)
            ladoIzq1 = x01x12+y01y12
            ladoDer1 = a01*a12
            ladoIzq2 = x12x23+y12y23
            ladoDer2 = a12*a23
            a01 = sqrt ((x1-x0)**2 + (y1-y0)**2)
            a12 = sqrt ((x2-x1)**2 + (y2-y1)**2)
            a23 = sqrt ((x3-x2)**2 + (y3-y2)**2)
            x01x12 = (x1-x0)*(x2-x1)
            y01y12 = (y1-y0)*(y2-y1)
            x12x23 = (x2-x1)*(x3-x2)
            y12y23 = (y2-y1)*(y3-y2)
            [(x0,y0),(x1,y1),(x2,y2),(x3,y3)] = take 4 (x:xs)

esPoligonoRegular :: Polilinea -> Bool
esPoligonoRegular xs = esPoligono xs && esEquilatero xs && esEquiangulo
xs
```

Ejercicio 6

```
#include <iostream>
#include <list>
using namespace std;

class Comic {
    protected:
    string saga;
    int agnoProduccion;
    public:
    Comic(const string& s, int a) : saga(s), agnoProduccion(a){}
    virtual ~Comic(){}
    string suSaga () const {
        return saga;
    }

    int suAgno () const{
        return agnoProduccion;
    }
};

class Volumen : public Comic{
    public:
    Volumen(const string& s, int a) : Comic(s,a){}
};

class Episodio : public Comic{
    private:
    string serie;
    public:
    Episodio(const string& s, int a, const string& se) : Comic(s,a),
serie(se){}

    string suSerie () const{
        return serie;
    }
};
```

```
class Serie : public Comic{
    private:
    list<Episodio> ep;
    string nombre;
    public:
    Serie (const string& s, int a, list<Episodio> e, const string& n) :
Comic(s,a), ep(e), nombre(n){}

    void agnadirEpisodio(Episodio e){
        for(auto i : ep){
            if(e.suAgno() < i.suAgno()){
                ep.insert(e);
            }
        }
    }

    string suNombre() const {
        return nombre;
    }
};

class Saga{
    private:
    list<Comic*> co;
    int agnoProduccion;
    string nombre;
    public:
    Saga (list<Comic*> c, int a, const string& n) : co(c),
agnoProduccion(a), nombre(n){}

    string suNombre() const {
        return nombre;
    }

    list<Comic*> suLista () const{
        return co;
    }

    void agnadirComic(Comic* c){
        for(auto i : co){
            if(c->suAgno() < i->suAgno()){
                co.insert(c);
            }
        }
    }
```

```
 list<Comic*> buscaAnteriores(int agno) const{
        list<Comic*> c;
        for(auto i : co){
            if(i->suAgno() <= agno){
                c.push_back(i);
            }
        }
        return c;
    }
};

class Biblioteca {
    private:
    list<Saga> s;
    public:
    Biblioteca (list<Comic*> co){
        bool sagaEncontrada = false, serieEncontrada = false;
        for(auto i : co){
            for(auto j : s){
                if (j.suNombre() == i->suSaga()){
                    sagaEncontrada = true;
                }
            }
            if (!sagaEncontrada){
                if (dynamic_cast<Episodio*>(i) != nullptr){
                    Episodio* episodio = dynamic_cast<Episodio*> (i);
                    for(auto x : co){   //Buscamos si está la serie a
la q pertenece el episodio i
                        if(dynamic_cast<Serie*> (x) != nullptr){
                            Serie* serie = dynamic_cast<Serie*> (i);
                            if(serie->suNombre() ==
episodio->suSerie()){
                                serieEncontrada = true;
                            }
                        }
                    }
                    if(!serieEncontrada){
                        list<Episodio> nueva;
                        Serie
se(episodio->suSaga(),episodio->suAgno(),nueva,episodio->suSerie());
                        se.agnadirEpisodio(*episodio);
                        list<Comic*> n;
                        Saga sa(n,i->suAgno(),i->suSaga());
                        sa.agnadirComic(i);
                        s.push_back(sa);
                    }
                }
```

```
                else {
                        list<Comic*> n;
                        Saga sa(n,i->suAgno(),i->suSaga());
                        sa.agnadirComic(i);
                        s.push_back(sa);
                }
            }
        }
    }

    list<Comic*> buscaAnteriores(int agno) const{
        list<list<Comic*>> c;
        list<Comic*> final;
        for(auto i : s){
            c.push_back(i.buscaAnteriores(agno));
        }
        for(auto i : c){
            for(auto j : i){
                final.push_back(j);
            }
        }
        return final;
    }
};
```