Ejercicio 1

```
// Programa A
int main(int argc, char** argv) {
    Bar bar;                              // t = 0
    bar.mar(3);                           // t = 3
    bar.mar(2);                           // t = 5
    std::cout<<bar.sil()<<std::endl;      // 5
    Foo<double> ffoo(0.3);                // t = 0.3
    ffoo.mar(3.1);                        // t = 3.4
    ffoo.mar(1.6);                        // t = 5.0
    std::cout<<ffoo.sil()<<std::endl;     // 5.0
    Foo<const char*> cfoo("ba");          // error de compilación
    cfoo.mar("tm"); cfoo.mar("an");       // const char* no tiene
    std::cout<<cfoo.sil()<<std::endl;     // comportamiento para el op“+”
}

// Programa B
int main(int argc, char** argv) {

    Foo* f = new Bar(nullptr);
    std::cout<<f->value()<<std::endl;
    f = new Bar(new Bar(nullptr));
    std::cout<<f->value()<<std::endl;
    f = new Bar(new Foo());               // Se intenta referenciar a un
    std::cout<<f->value()<<std::endl;     // elemento de la clase padre
mediante un elemento de la clase hija (error de compilación)
}
```

Ejercicio 2

C++

Apartado a)
El polimorfismo es una característica de un lenguaje de programación que permite que un mismo elemento del código pueda representar distintos elementos del lenguaje

Apartado b)
El mecanismo que permite el polimorfismo por inclusión en C++ es la herencia mediante clases. Mientras que el mecanismo que permite el polimorfismo paramétrico es la programación genérica mediante la sentencia template

Apartado c)
#include <iostream>
Ejemplo de polimorfismo por inclusión + polimorfismo paramétrico

```
template <typename T>
class A {
protected:
    T a;
public:
    A(const T& a) : a(a) {}
    virtual T metodo() const {
        return a*a;
    }
};

template <typename T>
class B : public A<T> {
    using A<T>::a;
    T b, c;
public:
    B(const T& a, const T& b, const T& c) : A<T>(a), b(b), c(c) {}
    T metodo() const override {
        return a*b*c;
    }
};

int main() {
    A<int> a(3);
    B<double> b(5.0, 2.4, 3.1);
    std::cout << a.metodo() << " " << b.metodo() << std::endl;
}
```

Apartado d)
El polimorfismo por inclusión se puede apreciar en la herencia formada por la clase A y la clase B. Asimismo en este ejemplo también se puede apreciar el polimorfismo paramétrico mediante la sentencia template <typename T>

Apartado e)
En Haskell únicamente existe el polimorfismo paramétrico

Haskell

Apartado f)

```
data Pair a = P a a

f :: [a] -> Bool
f [] = True
f _ = False

main = do
    print $ f [P 1.5 2.4, P 2.6 3.8, P 2.1 3.1]
    print $ f ""
    print $ f [P True False, P False False]
```

Apartado g)
En el ejemplo anterior se puede apreciar el polimorfismo en el tipo de dato Pair creado, que representa un par de elementos de tipo a. Asimismo, la función f se puede utilizar con listas de cualquier elemento a, como se puede observar en el main, donde se emplea con una lista de datos de tipo Pair (uno con Double y otro con Bool) y una lista de caracteres (String)

Ejercicio 3

```
{-# OPTIONS_GHC -Wno-incomplete-patterns #-}
type Nombre = String
type Edad = Int
type Epoca = String

data AgnoNacimiento = Nom Edad Epoca

instance Eq AgnoNacimiento where
    (==) (Nom e1 ep1) (Nom e2 ep2) = (ep1 == ep2) && (e1 == e2)

instance Ord AgnoNacimiento where
        (>) (Nom e1 ep1) (Nom e2 ep2) = ((ep1 == ep2) && (ep1 == "BBY")
&& (e1 > e2)) || ((ep1 == ep2) && (ep1 == "ABY") && (e1 < e2)) || (ep1
> ep2)
        (<) (Nom e1 ep1) (Nom e2 ep2) = ((ep1 == ep2) && (ep1 == "BBY")
&& (e1 < e2)) || ((ep1 == ep2) && (ep1 == "ABY") && (e1 > e2)) || (ep1
< ep2)
        (>=) (Nom e1 ep1) (Nom e2 ep2) = ((ep1 == ep2) && (e1 >= e2))
|| (ep1 >= ep2)
        (<=) (Nom e1 ep1) (Nom e2 ep2) = ((ep1 == ep2) && (e1 <= e2))
|| (ep1 <= ep2)

instance Show AgnoNacimiento where
    show (Nom e ep) = show e ++" "++ show ep

data ArbolSkywalker = Nulo | Nodo Nombre AgnoNacimiento
[ArbolSkywalker]

minimo :: ArbolSkywalker -> AgnoNacimiento
minimo Nulo = Nom 0 "ABY"
minimo (Nodo nom agno []) = agno
minimo (Nodo nom agno xs)
    | agno < minimo (head xs) = agno
    | otherwise = minimo (head  xs)

miembroMasJoven :: ArbolSkywalker -> Nombre
miembroMasJoven (Nodo nom agno []) =  nom
miembroMasJoven (Nodo nom agno xs) = if minimo (head xs) == agno then
nom
                                    else miembroMasJoven (head xs)

buscaNombre :: ArbolSkywalker -> Nombre -> Bool
buscaNombre Nulo a = False
buscaNombre (Nodo nom agno []) a = nom == a
buscaNombre (Nodo nom agno xs) a = (a == nom) || buscaNombre (head xs) a
```

```
antecesores :: ArbolSkywalker -> Nombre -> [Nombre]
antecesores Nulo n = []
antecesores (Nodo nom agno []) n
    | nom == n =[nom]
    | otherwise = []
antecesores (Nodo nom agno xs) n
    | nom == n = [nom]
    | buscaNombre (head xs) n = nom : antecesores (head xs) n
    | not (buscaNombre (head xs) n) = nom : antecesores (head (tail
xs)) n
    | otherwise = []

padre :: ArbolSkywalker -> Nombre -> [Nombre]
padre a n = drop (length l - 1) l
    where
        l = take (length lista - 1) lista
        lista = antecesores a n

arbol :: ArbolSkywalker
arbol = Nodo "Shmi" (Nom 72 "BBY")
    [Nodo "Anakin" (Nom 42 "BBY")
        [Nodo "Luke" (Nom 19 "BBY")
            [Nodo "Ben" (Nom 26 "ABY") []],
        Nodo "Leia" (Nom 19 "BBY")
            [Nodo "Jaina" (Nom 9 "ABY") [],Nodo "Jacen" (Nom 9 "ABY")
[],Nodo "Ben" (Nom 1 "ABY") [],Nodo "Anakin" (Nom 10 "ABY") []]]]
```

Ejercicio 4

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

template <typename T>
class Coleccion {
    list<T> col;
public:
    Coleccion() {}
    Coleccion(list<T> c) : col(c) {}

    void anyadir(const T& e) const {
        if (!pertenece(e)) {
            col.push_back(e);
        }
    }

    bool esCorrecta() const {
        // Si se usa la clase std::vector en vez de std::list
        // for (int i = 0; i < col.size(); i++) {
        //     for (int j = i+1; j < col.size(); j++) {
        //         if (col[i] == col[j]) {
        //             return false;
        //         }
        //     }
        // }
        // return true;
        for (auto i1 = col.begin(); i1 != col.end(); i1++) {
            for (auto i2 = next(i1, 1); i2 != col.end(); i2++){
                if (*i1 == *i2) {
                    return false;
                }
            }
        }
        return true;
    }

    bool pertenece(const T& e) const {
        for (auto i : col) {
            if (i == e) {
                return true;
            }
        }
        return false;
    }
```

```
    void mostrar() const {
        for (auto i : col) {
            cout << i << endl;
        }
    }
};

int main() {
    list<int> la = {1, 3, 2, 6, 3, 2};
    Coleccion<int> a(la);
    Coleccion<double> b;
    cout << boolalpha << a.esCorrecta() << endl;
    b.anyadir(3.4);
    b.anyadir(8.2);
    b.anyadir(1.9);
    b.anyadir(8.2);
    b.mostrar();
    cout << boolalpha << b.esCorrecta() << endl;
}
```