Ejercicio 1

```
#include <vector>
#include <iomanip>
#include <iostream>

class Abs {
public:
    float operator() (float a) const{
        if (a >= 0) return a;
        else return -a;
    }
};

class Rango {
public:
    bool operator() (float a) const {
        return (a >= 0.0 && a <= 1.0);
    }
};

template <typename F, typename A, typename B>
std::vector<B> map(const F& f, const std::vector<A>& v) {
    std::vector<B> res;
    for (auto x : v) {
        res.push_back(f(x));
    }
    return res;
}

template < typename T>
bool all(bool f, const std::vector<T>& v) {
    for (auto x : v) {
        if (!f(x)) {
            return false;
        }
    }
    return true;
}

bool unit(const std::vector<float>& v) {
    return all(Rango(), map(Abs(), v));
}

int main() {
    std::vector<float> v1 = {1.3, 0.2, -0.5, 2.1};
    std::vector<float> v2 = {1.0, 0.15, -0.7, -0.8};
    std::cout << "v1 = " << std::boolalpha << unit(v1) << std::endl;
    std::cout << "v2 = " << std::boolalpha << unit(v2) << std::endl;
}
```

Ejercicio 2

Apartado C++)

```
#include <iostream>

template <typename T>
T max(T a, T b) {
    if (a>b) return a;
    else return b;
}

int main() {
    std::cout << max(3,5) << std::endl;
    // std::cout << max("aa",5) << std::endl;
}
```

Apartado Java)

```
class Max {
    public static <T> T max(T a, T b) {
        if (a>b) return a;
        else return b;
    }

    public static void main(String[] args[]) {
        System.out.println(max(3,5));
    }
}
```

Apartado Haskell)

```
max :: (Ord t) => t -> t -> t
-- max :: t -> t -> t
max a b
    | a > b = a
    | otherwise = b

main = print (Main.max 3 5)
```

Apartado d)

En algunos lenguajes de programación como es el caso de Haskell y Java, es necesario acotar o restringir el tipo de dato génerico que se va a usar. Como podemos observar en la solución del apartado b se ha establecido que los tipos utilizados como parámetro deben ser ordenables (Ord t)

Ejercicio 3

```
gray :: Int -> [String]

gray 0 = [""]
gray n = map ('0':) xs ++ map ('1':) (reverse xs)
    where xs = gray (n-1)
```

Ejercicio 4

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

class Cuenta {
    int ident;
    double saldo;
public:
    Cuenta(int id, double sal) : ident(id), saldo(sal) {}

    int suId() const {
        return ident;
    }

    double suSaldo() const {
        return saldo;
    }

    void ingreso(double cant) {
        saldo += cant;
    }

    void retirada(double cant) {
        saldo -= cant;
    }
};

class Operacion {
protected:
    double cantidad;
public:
    Operacion(double cant) : cantidad(cant) {}
    virtual ~Operacion() {}
};

class Ingreso : public Operacion {
    int destino;
public:
    Ingreso(double cant, int dest) : Operacion(cant), destino(dest) {}

    int suDestino() const {
        return destino;
    }
};
```

```
class Extraccion : public Operacion {
    int origen;
public:
    Extraccion(double cant, int orig) : Operacion(cant), origen(orig)
{}

    int suOrigen() const {
        return origen;
    }
};

class Transferencia : public Operacion {
    int origen, destino;
public:
    Transferencia(double cant, int orig, int dest) : Operacion(cant),
origen(orig), destino(dest) {}

    int suOrigen() const {
        return origen;
    }

    int suDestino() const {
        return destino;
    }
};
class Cajero {
    list<Cuenta> cuentas;
    list<int> usadas;
    list<Operacion*> registro;
public:
    Cajero(list<Cuenta> c) : cuentas(c) {}

    bool ingreso(double cant, int dest) {
        for (auto i : cuentas) {
            if (i.suId() == dest) {
                i.ingreso(cant);
                registro.push_back(new Ingreso(cant, dest));
                usadas.push_back(i.suId());
                return true;
            }
        }
        return false;
    }
```

```
bool retirada(double cant, int orig) {
        for (auto i : cuentas) {
            if (i.suId() == orig && i.suSaldo() >= cant) {
                i.retirada(cant);
                Extraccion* ex = new Extraccion(cant, orig);
                registro.push_back(ex);
                usadas.push_back(i.suId());
                return true;
            }
        }
        return false;
    }

bool transferencia(double cant, int orig, int dest) {
        for (auto i : cuentas) {
            if (i.suId() == dest) {
                for (auto j : cuentas) {
                    if (j.suId() == orig) {
                        if (j.suSaldo() >= cant) {
                            j.retirada(cant);
                            usadas.push_back(j.suId());
                            i.ingreso(cant);
                            usadas.push_back(i.suId());
                            registro.push_back(new Transferencia(cant,
orig, dest));
                            return true;
                        }
                        else {
                            return false;
                        }
                    }
                }
                return false;
            }
        }
        return false;
    }

    list<int> cuentasUsadas() const {
        return usadas;
    }

    void reset() {
        usadas.clear();
    }
};
```