Ejercicio 1

```
// programa A
int main() {
    Foo* foo = new Foo(2.0);            // f = 2.0
    foo->sil();                         // f = 1.0
    cout<<foo->tolo()<<endl;            // 1.0
    Bar* bar = new Bar(4.0);            // f = 5.0
    bar->sil();                         // f = 10.0
    cout<<bar->tolo()<<endl;            // 9.0
    Foo* foobar = new Bar(8.0);         // f = 9.0
    foobar->sil();                      // f = 4.5
    cout<<foobar->tolo()<<endl;         // 3.5
}

// programa B
int main() {
    Foo<int> ifoo(155);
    cout<<ifoo.meta()<<endl;
    Foo<double> dfoo(42);
    cout<<dfoo.meta()<<endl;
    Bar bar;                // error de compilación, conflicto de
    cout<<bar.meta()<<endl; // tipos en la clase Bar
}
```

Ejercicio 2

Apartado a)
Una clase abstracta es aquella que tiene una parte que sólo especifica un cierto comportamiento a ofrecer pero no lo implementa.
- Define e implementa una parte del comportamiento de la clase.
- Define pero no implementa otra parte.
- No se puede instanciar.
- Sirve como prototipo para definir otras clases mediante herencia.

Apartado b)
Una clase abstracta sirve como prototipo para definir otras clases mediante herencia.

Apartado c)
La limitación que tiene una clase abstracta viene dada de que sus métodos no se pueden implementar en dicha clase, sino que deben ser implementados en las respectivas clases hijas.

Presenta esta limitación puesto que como dicta su definición esta clase únicamente sirve como prototipo para definir otras clases mediante herencia.

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

Apartado d)
```
class Vehiculo {
protected:
    string matricula;
    int ruedas;
public:
    Vehiculo(const string& mat, int rue) : matricula(mat), ruedas(rue)
{}

    virtual void info() const = 0;
};
```

```
class Coche : public Vehiculo {
private:
    int asientos;
public:
    Coche(const string& mat, int asi) : Vehiculo(mat, 4), asientos(asi)
{}
    void info() const override {
        cout << "Vehículo: Coche" << endl;
        cout << "Matrícula: " << matricula << endl;
        cout << "Ruedas: " << ruedas << endl;
        cout << "Asientos: " << asientos << endl;
    }
};

class Moto : public Vehiculo {
public:
    Moto(const string& mat) : Vehiculo(mat, 2) {}

    void info() const override {
        cout << "Vehículo: Moto" << endl;
        cout << "Matrícula: " << matricula << endl;
        cout << "Ruedas: " << ruedas << endl;
    }
};

void mostrar(const Vehiculo& v) {
    v.info();
}

int main() {
    Coche a("1234 ABC", 5);
    mostrar(a);
    cout << endl;
    Moto b("6789 XYZ");
    mostrar(b);
    // Vehiculo v("5392 HTM", 2); -> Apartado e
}
```

Apartado e)
En el ejemplo se puede observar una jerarquía de clases donde hay una clase padre y dos hijas.

En la clase padre se encuentran los atributos matricula y ruedas (los cuales heredaran sus hijas) y el método info, el cual como podemos observar, gracias a las palabras clave "virtual" y "= 0" se define como abstracto. Este método es implementado en cada una de las clases hijas.

Ejercicio 3

```
type Tupla = (Double, Double, Double)
type Recorrido = [Tupla]

distanciaTotal :: Recorrido -> Double
distanciaTotal [] = 0.0
distanciaTotal [(x,y,t)] =  sqrt (x**2 + y**2)
distanciaTotal (x:xs) = if length xs == 2--(x2,y2,t2) /= last xs
                        then sqrt ((x2 - x1)**2 + (y1 - y2)**2) +
distanciaTotal xs
                        else
                            distanciaTotal xs
    where
        [(x1,y1,t1),(x2,y2,t2)] = take 2 (x:xs)

esSubversivo :: Recorrido -> Bool
esSubversivo [] = False
esSubversivo [(x,y,t)] = t > 600
esSubversivo (x:xs)
    | t1 > 600 = True
    | t2-t1 > 600 = True
    | otherwise = esSubversivo xs
    where
        [(x1,y1,t1),(x2,y2,t2)] = take 2 (x:xs)

tiempoTotal :: Recorrido -> Double
tiempoTotal [] = 0.0
tiempoTotal [(x,y,t)] = t
tiempoTotal (x:xs) = tiempoTotal xs


velocidadMedia :: [Recorrido] -> Double
velocidadMedia [] = 0.0
velocidadMedia (x:xs) = distanciaTotal (x) / tiempoTotal(x) +
velocidadMedia xs
```

Ejercicio 4

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

class Chat {
    list<Mensaje*> chat;
public:
    Chat() {}
    void agnadir(Mensaje& m) {
        // Mensaje* aux = &m;
        // chat.push_back(aux);
        chat.push_back(&m);
    }
    void borrar(int instante) {
        for (auto i : chat) {
            if (i->su_instante() < instante) {
                chat.remove(i);
            }
        }
    }
    void generar() {
        list<string> miniaturas;
        for (auto i : chat) {
            // miniaturas.push_back(i->suMiniatura());
        }
    }
};

class Mediateca {
    list<Chat> lista_chats;
public:
    Mediateca() {}

    void crear() {
        Chat c;
        lista_chats.push_back(c);
    }

    void agnadir(Mensaje& m, Chat& c) {
        c.agnadir(m);
    }

    void borrar(int instante) {
        for (auto i : lista_chats) {
            i.borrar(instante);
        }
    }
```

```
    void generar(Chat& c) {}
};

class Mensaje {
protected:
    int identificador;
    string emisor;
    int fechaHora;
    int tamagno;
public:
    Mensaje(int id, const string& em, int fh, int tam) :
identificador(id), emisor(em), fechaHora(fh), tamagno(tam) {}
    int su_instante() {
        return fechaHora;
    }
};

class Medio_Audiovisual : public virtual Mensaje {
protected:
    int duracion;
public:
    Medio_Audiovisual(int id, const string& em, int fh, int tam, int
dur) : Mensaje(id, em, fh, tam), duracion(dur) {}
};

class Medio_Imagen : public virtual Mensaje {
protected:
    string lugar;
    int resolucion;
public:
    Medio_Imagen(int id, const string& em, int fh, int tam, string l,
int res) : Mensaje(id, em, fh, tam), lugar(l), resolucion(res) {}

};

class Video : public Medio_Imagen, public Medio_Audiovisual {
    // Miniatura miniatura
public:
    Video(int id, const string& em, int fh, int tam, int dur, string
lug, int res) : Mensaje(id, em, fh, tam), Medio_Audiovisual(id, em, fh,
tam, dur), Medio_Imagen(id, em, fh, tam, lug, res) {}

};

class Audio : public Medio_Audiovisual {
    // Miniatura miniatura
    Audio(int id, const string& em, int fh, int tam, int dur) :
Mensaje(id, em, fh, tam), Medio_Audiovisual(id, em, fh, tam, duracion)
{}

};
```

```
class Foto : public Medio_Imagen {
    // Miniatura miniatura
    Foto(int id, const string& em, int fh, int tam, string lug, int
res) : Mensaje(id, em, fh, tam), Medio_Imagen(id, em, fh, tam, lug,
res) {}
};

class Texto : public Mensaje {
    // Miniatura miniatura
    Texto(int id, const string& em, int fh, int tam) : Mensaje(id, em,
fh, tam) {}
};
```