Ejercicio 1

```
int main(int argc, char** argv) {
    Foo* foo = new Foo(1);         // t = 1
    foo->sil();                    // t = 2
    foo->sil();                    // t = 3
    foo->output();                 // 3

    Bar* bar = new Bar(2);         // t = 4
    bar->sil();                    // t = 8
    bar->sil();                    // t = 16
    bar->output();                 // | 16 |

    Foo* foobar = new Bar(3);      // t = 6
    foobar->sil();                 // t = 7
    foobar->sil();                 // t = 8
    foobar->output();              // | 8 |
}
```

```
int main(int argc, char** argv) {
  Bar<int> ibar(new Foo<int>(42));
  std::cout<<ibar.tolo()<<std::endl;

  Bar<float> fbar(new Foo<float>(4.8f));
  std::cout<<fbar.tolo()<<std::endl;
  Bar<double> dbar(new Foo<int>(1024)); // error de compilación por
  std::cout<<dbar.tolo()<<std::endl;    // diferencia de tipos de datos
}
```

Ejercicio 2

Apartado a)

La programación genérica es un estilo de programación que permite escribir código que opera de la misma forma con cualquier tipo de datos, sin que estos tipos tengan que estar relacionados por la herencia. Asimismo permite escribir código reutilizable independientemente de los datos sobre los que se va a aplicar.

Apartado b)

Una clase genérica es aquella que se especifica mediante un tipo desconocido (paramétro) en la definición de la clase. Este tipo se concretará después en su instanciación.

Un método genérico es aquel que utiliza tipos de datos (parámetros) desconocidos a priori, de manera que podrá utilizar cualquier tipo de dato compatible con las operaciones realizadas en dicho método.

Apartado c)

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

template <typename T>
class A {
    T atr1;
public:
    A (T atr1) : atr1(atr1){}

    T metodo1(T dato) {
        return atr1 + dato;
    }
};
```

Apartado d)

```
int main() {
    A<int> a1(5);
    A<double> a2(2.78);
    cout << a1.metodo1(3) << endl;
    cout << a2.metodo1(1.85) << endl;
}
```

Apartado e)

En el programa realizado se crea una clase genérica A, como se puede observar en el tipo T utilizado en su especificación. Este tipo T es utilizado en el atributo atr1. Por otra parte, el método definido en la clase A también es genérico, ya que sus parámetros también son el tipo T empleado en la especificación de la clase. En el programa principal se reemplaza el tipo T por el tipo de datos "int" en la instanciación de la clase A, de manera que tanto el atributo atr1 como el tipo de dato utilizado en el método serán del tipo especificado en la instanciación.

Ejercicio 3

```
curveZeroes' :: [Double] -> Integer
curveZeroes' [] = 0
curveZeroes' [_] = 0
curveZeroes' (x:xs)
    | (x1 == 0 && x2 /= 0) || (x1 < 0 && x2 > 0) || (x1 > 0 && x2 < 0)
= 1 + curveZeroes' xs
    | otherwise = curveZeroes' xs
    where
        [x1,x2] = take 2 (x:xs)
```

Ejercicio 4

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

class Elemento {
protected:
    int precio;
public:
    Elemento(int pr) : precio(pr) {}
    virtual ~Elemento() {}

    int suPrecio() const {
        return precio;
    }
};

class Persona : public Elemento {
public:
    Persona() : Elemento(4) {}
};

class Vehiculo : virtual public Elemento {
public:
    Vehiculo(int pr) : Elemento(pr) {}
};

class Alojamiento : virtual public Elemento {
protected:
    bool avisado;
public:
    Alojamiento() : Elemento(6), avisado(false) {}

    void avisar() {
        cout << "Alarma alojamiento" << endl;
        avisado = true;
    }
};

class Adulto : public Persona {
    bool avisado;
public:
    Adulto() : Persona(), avisado(false) {}

    void avisar() {
        cout << "Alarma adulto" << endl;
        avisado = true;
    }
};
```

```
class Ninyo : public Persona {
public:
    Ninyo() : Persona() {}
};

class Coche : public Vehiculo {
public:
    Coche() : Vehiculo(5), Elemento(5) {}
};

class Microbus : public Vehiculo {
public:
    Microbus() : Vehiculo(10), Elemento(10) {}
};

class Tienda : public Alojamiento {
public:
    Tienda() : Alojamiento(), Elemento(6) {}
};

class Caravana : public Alojamiento {
public:
    Caravana() : Alojamiento(), Elemento(6) {}
};

class Autocaravana : public Vehiculo, public Alojamiento {
public:
    Autocaravana() : Vehiculo(5), Alojamiento(), Elemento(5) {}
};

class Parcela {
    list<Elemento*> parcela;
public:
    Parcela(list<Elemento*> p) : parcela(p) {}

    int precio() const {
        int total = 0;
        for (auto i : parcela) {
            total += i->suPrecio();
        }
        return total;
    }
```

```
void emergencia() {
        for (auto i : parcela) {
            if (dynamic_cast<Adulto*>(i) != nullptr) {
                Adulto* a = dynamic_cast<Adulto*>(i);
                a->avisar();
            }
            else if (dynamic_cast<Alojamiento*>(i) != nullptr) {
                Alojamiento* a = dynamic_cast<Alojamiento*>(i);
                a->avisar();
            }
        }
    }
};

class Camping {
    list<Parcela> camping;
public:
    Camping(list<Parcela> c) : camping(c) {}

    int ingresos() const {
        int total = 0;
        for (auto i : camping) {
            total += i.precio();
        }
        return total;
    }

    void emergencia() {
        for (auto i : camping) {
            i.emergencia();
        }
    }
};

int main() {
    Parcela p1( {new Adulto(), new Adulto(), new Ninyo(), new
Caravana()} );
    Parcela p2( {new Adulto(), new Coche(), new Coche(), new Tienda()}
);
    Parcela p3( {new Adulto(), new Autocaravana()} );
    Parcela p4( {new Adulto(), new Microbus(), new Tienda()} );
    Camping c( {p1, p2, p3, p4} );
    cout << c.ingresos() << endl;
    c.emergencia();
}
```