Punto 1

A partir de las siguientes definiciones:

```
map f [] = []
map f (h:t) = (f h) : map f t

take 0 _  = []
take _ [] = []
take n (x:xs)  = x : take (n-1) xs

foo f y = take y ( map  (f y) [y..])
```

Indicar:

a) Para entender mínimamente qué hace la función: Indicar qué sucede en Haskell ante las siguientes invocaciones:

```
foo (*) 3
foo (-) 0
foo (+) (length [1..])
foo (++) ' '
```

b) ¿Cuáles de los siguientes conceptos se utilizan en la función foo y cuáles no? Justificar brevemente las respuestas, indicando donde se aplica cada concepto y qué aportan a la solución

- Orden Superior
- Aplicación Parcial
- Evaluación Diferida
- Expresiones Lambda
- Composición
- Inversibilidad

c) ¿Cuál son las principales dificultades que aparecen para programar esto en Lógico y Objetos?

Punto 2

Se tiene una aplicación en Smalltalk que calcula el valor de una entrada al teatro. Por el momento hay sólo entradas para ballet y obras infantiles, pero en un futuro pueden agregarse otros estilos de obras.

La política del teatro para todos les estilos que se agreguen en un futuro es respetar que a partir de cierta edad se cobra siempre $10 y que a los menores de cierta edad es gratis, pero dichas edades pueden ser diferentes según el estilo. El precio de las entradas para el resto de las personas es un valor que depende del estilo, es decir que por ejemplo una entrada para cualquier obra infantil sale lo mismo, pero puede costar un precio distinto al de una obra de ballet, musical o cualquier nuevo estilo.

```
ObraTeatro (v.i. nombre, actores, estilo. v.c. PrecioBallet, PrecioInfantil)

precioPara: alguien
(estilo nombre = 'ballet' & alguien edad  < 18) ifTrue:[ ^0].
(estilo nombre = 'ballet' & alquien edad  > 70) ifTrue:[ ^10].
(estilo nombre = 'ballet' & alguien edad between: 18 and: 70) ifTrue:[
     ^PrecioBallet].
(estilo nombre = 'infantil' & alguien edad  < 3) ifTrue:[ ^0].
```

```
(estilo nombre = 'infantil' & alguien edad  > 50) ifTrue:[ ^10].
(estilo nombre = 'infantil' & alguien edad between: 3 and: 50) ifTrue:[
     ^PrecioInfantil].

Persona (m.i. edad …)

Estilo (m.i. nombre …)
```

a) ¿Qué objeciones se pueden hacer a la solución propuesta? Indicar qué conceptos no se están usando.

b) Plantear una solución mejor, incluyendo un diagrama de clases con codificación de todos los métodos necesarios. Explicar los conceptos que ayudaron a plantear la solución.

c) Hay un nuevo requisito de incorporar nuevas obras de cualquier estilo en las que no sólo cambian las edades límites y los precios sino que además se aplican recargos en función de la cantidad de actores que participa en la obra (más de 10 actores, un plus de $50) ¿Qué nuevos conceptos aparecen? Codificar y justificar.

Punto 3

Se cuenta con las siguientes definiciones en los lenguajes Haskell y Prolog, que funcionan correctamente.

```
agregarAlFinal [] c = [c]
agregarAlFinal (x:xs) c = x: agregarAlFinal xs c

alFinal([], C, [C]).
alFinal([X|Xs], C, [X|Zs]):- alFinal(Xs, C, Zs)
```

a) Explicar en ambas soluciones la aplicación del concepto de pattern matching y unificación.

b) ¿De qué tipos de datos pueden ser los elementos de las listas en cada solución? Explicar similitudes y diferencias entre ambas soluciones. En la solución de funcional, explicitar el domino e imagen de la función de la forma más genérica posible.

c) ¿Qué consultas/invocaciones se pueden hacer con cada solución? Mostrar un ejemplo donde se vea un uso similar y uno donde haya diferencias. Justificar conceptualmente.