Punto 1

Se está codificando un juego de aventuras en un lenguaje procedural.
El juego incluye un tablero de control en el que se indica el puntaje de cada equipo.
Estas son las estructuras y procedimientos que intervienen en el cálculo de dicho valor:

```
typedef struct {
  int poblacion;
  Poblador pobladores[100];
} Pueblo;

// un poblador puede ser: un guerrero o un sabio
typedef struct {
  int tipo;
  Arma arma_guerr;
  int fuerza_guerr;
  int edad_sabio;
  int gradoExperiencia_sabio;
} Poblador;

int valorPueblo(Pueblo pueblo) {
  int resultado = 0;
  for (i = 0; i < pueblo.size; i++) {
    Poblador pobl = pueblo.elementos[i];

    // calculo el valor del elemento
    int valorElemento;
    switch (pobl.tipo) {
      // guerrero: valor del arma + 5 * la fuerza
      case 0:    // 0 quiere decir guerrero
        valorElemento =
            valorArma(pobl.arma_guerr) + (5* pobl.fuerza_guerr);
        break;
      // sabio: edad / 2
      case 1:    // 1 quiere decir sabio
        valorElemento = pobl.gradoExperiencia_sabio * 10 + 7;
        break;
    }

    resultado += valorElemento;
  }
  return resultado;
}
```

Se pide:

1. Modelar la misma situacion en objetos, con el mismo nivel de detalle, y respetando las ideas del paradigma de objetos vistas en la materia.
2. En el codigo se utiliza una funcion llamada valorArma, pero no se muestra el desarrollo de la misma. ¿Cómo se traduce eso a objetos? Relacionar con los conceptos de mensaje, metodo e interfaz.
3. Indicar en el código (o pseudocódigo) que escribió, dónde se usa delegación y dónde polimorfismo, justificando sus respuestas.
4. Indicar ventajas de la solución de objetos vs. la planteada ante estos escenarios de extensión:
    a. se agrega un nuevo tipo de poblador, p.ej. obrero.
    b. se quiere saber cuántos pobladores de un pueblo tienen valor menor a n, donde n es un valor que se pasa como parámetro.
    c. se quiere saber si un pueblo tiene pobladores grosos. Un guerrero es groso si su valor es 100 o más, un sabio es groso si su valor es mayor a su edad.

Punto 2

Tenemos estos dos juegos de expresiones en Smalltalk y en Haskell, que hacen lo mismo:

ident.	Smalltalk	Haskell
e1	`lista select: cond`	`filter cond lista`
e2	`(lista select: cond) size`	`length (filter cond lista)`
e3	`(lista select: cond)` `  copyFrom: 1 to: 5`	`take 5 (filter cond lista)`

Tener en cuenta una lista muuuy larga, de la cual aprox. la mitad de los elementos cumplen la condición, y están distribuidos razonablemente (o sea, no es que están todos adelante los que sí y atrás los que no, ni viceversa). En Smalltalk `lista` es una referencia a una colección ordenada.

Se pide

1. Ordenar las expresiones e1, e2 y e3 de Smalltalk de la que tarda menos a la que tarda más. Tener en cuenta que copyFrom:to:, como devuelve una nueva colección, tiene que crearla y agregarle 5 elementos; suponer que pedirle el tamaño a una colección ordenada es una operación rápida.
2. Hacer lo mismo para las expresiones e1, e2 y e3 en Haskell.
3. Los órdenes no coinciden: explicar con qué concepto está relacionado esto, y para cada caso entre e1, e2 y e3, si influye o no.
   Ayuda: pensarlo desde funcional.
4. Si cambio `lista` por `[1..]` en las expresiones en Haskell, indicar para cada una entre e1, e2 y e3, que pasaría al hacer la consulta, y explicar.

Punto 3

Se tienen estos programas en Prolog y en Haskell que definen a quiénes se les puede dar crédito en una institución financiera.

```
seLeDaCredito(X):- rico(X).
seLeDaCredito(X):-
      amigo(X,A), poderoso(A).
seLeDaCredito(X):- elegante(X).
```

```
seLeDaCredito x = rico x
seLeDaCredito x =
      any poderoso (amigos x)
seLeDaCredito x = elegante x
```

Se entiende que el argumento de todos estos predicados y funciones representa una persona; las funciones rico, poderoso y elegante devuelven un booleano, y lo mismo se espera de seLeDaCredito. Se pide:

1. Una de las dos soluciones no es correcta, es decir, el programa compila pero no ayuda a resolver el problema planteado en la situacion inicial. ¿Cuál es, y por qué? Relacionar su respuesta con las caracteristicas de los paradigmas.
2. Suponiendo que las dos soluciones fueran correctas, hay un tipo de consulta respecto de la posibilidad de dar crédito que en una de ellas puede hacerse mientras que en la otra no.
   Indicar cuál es la consulta, en cuál de los dos paradigmas se puede hacer, y qué concepto de ese paradigma es el que habilita esta posibilidad.
3. Si en el de Prolog (suponiendo que anduviera o haciéndolo andar) preguntamos si le puedo dar crédito a `juan`, y `juan` no aparece en el programa, ¿cuál sería la respuesta obtenida salvo un programa muy raro? Relacionar con algún concepto visto en la materia.