

Trabajo Práctico

Programación Funcional

Redes Sociales

Introducción a la Programación - Primer cuatrimestre 2023

Fecha límite de entrega:
Viernes 19 de mayo

Cambios Versión 2.1 (3 de mayo 2023)

- Se agrego en la definición del predicado `MismosElementos` que $|l1| = |l2|$.
- Se agrego a los problemas `amigosDe`, `publicacionesDe` y `publicacionesQueLeGustanA` el siguiente asegura: `asegura: {SinRepetidos(res)}`

Introducción

El objetivo de este Trabajo Práctico es aplicar los conceptos de programación funcional vistos en la materia para programar un ejemplo de red social utilizando Haskell.

A continuación se explica el problema a resolver y como anexo, al final de este enunciado, se presenta una especificación formal de los ejercicios que se deben programar.

En nuestro ejemplo, los usuarios solamente pueden realizar las siguientes **acciones**:

- Relacionarse entre sí, es decir, *ser amigos*.
- Postear publicaciones de texto.
- Dar *me gusta* a las publicaciones de sus amigos.

La **red social** se define a partir de los **usuarios**, las **publicaciones** de cada uno y las **relaciones de amistad** entre ellos, donde:

- Cada **usuario** está representado con una tupla de 2 elementos, donde el primero corresponde al número de identificación (id) y el segundo a su nombre de usuario. Todos los usuarios de la red se encuentran en una lista de **usuarios**.
- Una **publicación** es una tupla de 3 elementos compuesta por: el autor de dicha publicación, el texto publicado y el conjunto de los usuarios que le dieron *me gusta*. Todas las publicaciones de la red están en la lista **publicaciones**.
- La **amistad** entre dos miembros de la red está representada con una tupla de dos usuarios. Las relaciones de amistad se encuentran en la lista **relaciones**.

Luego, podemos definir nuestra **red social** como una tupla de 3 elementos que contenga los tres conceptos antes mencionados, es decir: `(usuarios, relaciones, publicaciones)`

Definiciones y funciones

Para representar y manipular la red social se cuenta con el archivo `iap1-tp.hs` que tiene las siguientes definiciones de tipos:

```
type Usuario = (Integer, String1)
```

¹Cabe aclarar que el tipo `String` es un sinónimo de `[Char]`, es decir una lista de caracteres.

- `type Relacion = (Usuario, Usuario)`
- `type Publicacion = (Usuario, String, [Usuario])`
- `type RedSocial = ([Usuario], [Relacion], [Publicacion])`

Además, en el archivo `iap1-tp.hs` se definen las siguientes funciones básicas:

- `usuarios :: RedSocial -> [Usuario]`
Devuelve el conjunto de usuarios.
- `relaciones :: RedSocial -> [Relacion]`
Devuelve el conjunto de relaciones.
- `publicaciones :: RedSocial -> [Publicacion]`
Devuelve el conjunto de publicaciones.
- `idDeUsuario :: Usuario -> Integer`
Devuelve el número de identificación de un usuario.
- `nombreDeUsuario :: Usuario -> String`
Devuelve el nombre de un usuario.
- `usuarioDePublicacion :: Publicacion -> Usuario`
Devuelve el usuario de una publicación.
- `likesDePublicacion :: Publicacion -> [Usuario]`
Devuelve el conjunto de usuarios que le dieron *me gusta* a una publicación.

Ejercicios obligatorios

Se pide implementar las siguientes funciones:

Ejercicio 1. `nombresDeUsuarios :: RedSocial -> [String]`

Ejercicio 2. `amigosDe :: RedSocial -> Usuario -> [Usuario]`

Ejercicio 3. `cantidadDeAmigos :: RedSocial -> Usuario -> Int`

Ejercicio 4. `usuarioConMasAmigos :: RedSocial -> Usuario`

Ejercicio 5. `estaRobertoCarlos :: RedSocial -> Bool`

Ver video: <https://www.youtube.com/watch?v=PZXaQijiiAA>, o googlear: roberto carlos, yo solo quiero”.

Ejercicio 6. `publicacionesDe :: RedSocial -> Usuario -> [Publicacion]`

Ejercicio 7. `publicacionesQueLeGustanA :: RedSocial -> Usuario -> [Publicacion]`

Ejercicio 8. `lesGustanLasMismasPublicaciones :: RedSocial -> Usuario -> Usuario -> Bool`

Ejercicio 9. `tieneUnSeguidorFiel :: RedSocial -> Usuario -> Bool`

Ejercicio 10. `existeSecuenciaDeAmigos :: RedSocial -> Usuario -> Usuario -> Bool`

Pautas de Entrega

Para la entrega del trabajo práctico se deben tener en cuenta las siguientes consideraciones:

- El trabajo se debe realizar en grupos de cuatro alumnos.
- Se debe entregar un conjunto de casos de test que el grupo utilizó para testear los programas.
- Los programas deben pasar con éxito los casos de test entregados por la cátedra (en el archivo `test-catedra.hs`).
- El archivo con el código fuente debe tener nombre `iap1-tp.hs`. Además, en el archivo entregado debe indicarse, en un comentario arriba de todo: nombre de grupo, nombre, email y LU (o DNI) de cada integrante.
- El código debe poder ser ejecutado en el GHCi instalado en los laboratorios del DC, sin ningún paquete especial.
- No está permitido alterar los tipos de datos presentados en el enunciado, ni utilizar técnicas no vistas en clase para resolver los ejercicios (como por ejemplo, alto orden).
- Pueden definirse todas las funciones auxiliares que se requieran. Cada una debe tener un comentario indicando claramente qué hace.
- No es necesario entregar un informe sobre el trabajo.
- La fecha límite de entrega es el Viernes 19/5.

Los objetivos a evaluar para aprobar este trabajo práctico son:

- **Correctitud:** todos los ejercicios obligatorios deben estar bien resueltos, es decir, deben respetar su especificación.
- **Declaratividad:** el código debe estar comentado y los nombres de las funciones que se definan deben ser apropiados.
- **Consistencia:** el código debe atenerse al uso correcto de las técnicas vistas en clase como recursión o *pattern matching*.
- **Prolijidad:** evitar repetir código innecesariamente y usar adecuadamente las funciones previamente definidas (por el enunciado o por ustedes mismos).
- **Testeo:** Todos los ejercicios deben tener sus propios casos de test que pasen correctamente. Además, sus implementaciones deben pasar satisfactoriamente los casos de test "secretos" de la cátedra.

Importante: se admitirá un único envío, sin excepción, por grupo. Planifiquen el trabajo para llegar a tiempo con la entrega.

Referencias del lenguaje Haskell

- **The Haskell 2010 Language Report:** la última versión oficial del lenguaje Haskell a la fecha, disponible online en <http://www.haskell.org/onlinereport/haskell2010>.
- **Learn You a Haskell for Great Good!:** libro accesible, para todas las edades, cubriendo todos los aspectos del lenguaje, notoriamente ilustrado, disponible online en <http://learnyouahaskell.com> y en <http://aprendehaskell.es/> (en español).
- **Real World Haskell:** libro apuntado a zanjar la brecha de aplicación de Haskell, enfocándose principalmente en la utilización de estructuras de datos funcionales en la "vida real", disponible online en <http://book.realworldhaskell.org/read>.
- **Hoogle:** buscador que acepta tanto nombres de funciones y módulos, como signatures y tipos *parciales*, online en <http://www.haskell.org/hoogle/>

Anexo - Especificación

Funciones básicas

- problema usuarios (red: RedSocial) : $seq\langle Usuario \rangle$ {
 requiere: $\{true\}$
 asegura: $\{res = red_0\}$
}
- problema relaciones (red: RedSocial) : $seq\langle Relacion \rangle$ {
 requiere: $\{true\}$
 asegura: $\{res = red_1\}$
}
- problema publicaciones (red: RedSocial) : $seq\langle Publicacion \rangle$ {
 requiere: $\{true\}$
 asegura: $\{res = red_2\}$
}
- problema idDeUsuario (u: Usuario) : \mathbb{Z} {
 requiere: $\{true\}$
 asegura: $\{res = u_0\}$
}
- problema nombreDeUsuario (u: Usuario) : $seq\langle Char \rangle$ {
 requiere: $\{true\}$
 asegura: $\{res = u_1\}$
}
- problema usuarioDePublicacion (p: Publicacion) : Usuario {
 requiere: $\{true\}$
 asegura: $\{res = p_0\}$
}
- problema likesDePublicacion (p: Publicacion) : $seq\langle Usuario \rangle$ {
 requiere: $\{true\}$
 asegura: $\{res = p_2\}$
}

Predicados Auxiliares

- pred Pertenece (e:T, l: $seq\langle T \rangle$) {
 $(\exists i : \mathbb{Z})(0 \leq i < |l| \wedge l[i] = e)$
}
- pred MismosElementos (l1: $seq\langle T \rangle$, l2: $seq\langle T \rangle$) {
 $|l1| = |l2| \wedge (\forall x : \mathbb{Z})(0 \leq x < |l1| \rightarrow Pertenece(l1[x], l2)) \wedge (\forall x : \mathbb{Z})(0 \leq x < |l2| \rightarrow Pertenece(l2[x], l1))$
}
- pred RedSocialValida (red:RedSocial) {
 $UsuariosValidos(usuarios(red)) \wedge RelacionesValidas(relaciones(red))$
 $\wedge PublicacionesValidas(publicaciones(red))$
}

```

}

▪ pred UsuariosValidos (us:seq<Usuario>) {
   $(\forall x : \mathbb{Z})(0 \leq x < |us| \rightarrow UsuarioValido(us[x])) \wedge noHayIdsRepetidos(us)$ 
}

▪ pred UsuarioValido (u:Usuario) {
   $idDeUsuario(u) > 0 \wedge |nombreDeUsuario(u)| > 0$ 
}

▪ pred noHayIdsRepetidos (us:seq<Usuario>) {
   $(\forall x, y : \mathbb{Z})(0 \leq x < |us| \wedge 0 \leq y < |us| \wedge x \neq y \rightarrow idDeUsuario(us[x]) \neq idDeUsuario(us[y]))$ 
}

▪ pred RelacionesValidas (us:seq<Usuario>, rels:seq<Relacion>) {
   $UsuariosDeRelacionValidos(us, rels) \wedge RelacionesAsimetricas(rels) \wedge NoHayRelacionesRepetidas(rels)$ 
}

▪ pred UsuariosDeRelacionValidos (us:seq<Usuario>, rels:seq<Relacion>) {
   $(\forall x : \mathbb{Z})(0 \leq x < |rels| \rightarrow (Pertenece(rels[x]_0, us) \wedge Pertenece(rels[x]_1, us) \wedge rels[x]_0 \neq rels[x]_1))$ 
}

▪ pred RelacionesAsimetricas (rels:seq<Relacion>) {
   $(\forall x : \mathbb{Z})(0 \leq x < |rels| \rightarrow \neg Pertenece((rels[x]_1, rels[x]_0), rels))$ 
}

▪ pred NoHayRelacionesRepetidas (rels:seq<Relacion>) {
   $(\forall x, y : \mathbb{Z})(0 \leq x < |rels| \wedge 0 \leq y < |rels| \wedge x \neq y \rightarrow (idDeUsuario(rels[x]_0) \neq idDeUsuario(rels[y]_0) \vee idDeUsuario(rels[x]_1) \neq idDeUsuario(rels[y]_1)))$ 
}

▪ pred PublicacionesValidas (us:seq<Usuario>, pubs:seq<Publicacion>) {
   $UsuariosDePublicacionSonUsuariosDeRed(us, pubs) \wedge UsuariosDeLikeDePublicacionSonUsuariosDeRed(us, pubs) \wedge NoHayPublicacionesRepetidas(pubs)$ 
}

▪ pred UsuariosDePublicacionSonUsuariosDeRed (us:seq<Usuario>, pubs:seq<Publicacion>) {
   $(\forall x : \mathbb{Z})(0 \leq x < |pubs| \rightarrow Pertenece(usuarioDePublicacion(pubs[x]), us))$ 
}

▪ pred UsuariosDeLikeDePublicacionSonUsuariosDeRed (us:seq<Usuario>, pubs:seq<Publicacion>) {
   $(\forall x : \mathbb{Z})(0 \leq x < |pubs| \rightarrow UsuariosLikeValidos(us, likesDePublicacion(pubs[x])))$ 
}

▪ pred UsuariosLikeValidos (us:seq<Usuario>, usl:seq<Usuario>) {
   $(\forall x : \mathbb{Z})(0 \leq x < |usl| \rightarrow Pertenece(usl[x], us))$ 
}

▪ pred NoHayPublicacionesRepetidas (pubs:seq<Publicacion>) {
   $(\forall x, y : \mathbb{Z})(0 \leq x < |pubs| \wedge 0 \leq y < |pubs| \wedge x \neq y \rightarrow (idDeUsuario(usuarioDePublicacion(pubs[x])) \neq idDeUsuario(usuarioDePublicacion(pubs[y]))) \vee (pubs[x]_1 \neq pubs[y]_1))$ 
}

▪ pred CadenaDeAmigos (us:seq<Usuario>, red:RedSocial) {
   $(\forall x : \mathbb{Z})(0 \leq x < |us| - 1 \rightarrow RelacionadosDirecto(us[x], us[x+1], red))$ 
}

▪ pred RelacionadosDirecto (u1:Usuario, u2:Usuario, red:RedSocial) {
   $Pertenece((u1, u2), Relaciones(red)) \vee Pertenece((u2, u1), Relaciones(red))$ 
}

```

```

}
▪ pred SonDeLaRed (red:RedSocial, us:seq<Usuario>) {
  (∀x : Z)(0 ≤ x < |us| → Pertenece(us[x], usuarios(red)))
}
▪ pred EmpiezaCon (e:T, l:seq<T>) {
  head(l) = e
}
▪ pred TerminaCon (e:T, l:seq<T>) {
  subseq(l, |l| - 1, |l|) = e
}
▪ pred SinRepetidos (l:seq<T>) {
  (∀x, y : Z)((0 ≤ x < |l| ∧ 0 ≤ y < |l| ∧ x ≠ y) → l[x] ≠ l[y])
}

```

Especificaciones de Ejercicios

Ejercicio 1

```

problema nombresDeUsuarios (red: RedSocial) : seq<seq<Char>> {
  requiere: {RedSocialValida(red)}
  asegura: {MismosElementos(res, proyectarNombres(usuarios(red)))}
}

problema proyectarNombres (us: seq<Usuario>) : seq<seq<Char>> {
  requiere: {True}
  asegura: {(∀x : Z)(0 ≤ x < |us| → Pertenece(nombreUsuario(us[x]), res))}
  asegura: {((∀x : Z)(0 ≤ x < |res| → (∃u : Usuario)(Pertenece(u, us) ∧ nombreUsuario(u) = res[x]))}
  asegura: {SinRepetidos(res)}
}

```

Ejercicio 2

```

problema amigosDe (red:RedSocial, u:Usuario) : seq<Usuario> {
  requiere: {RedSocialValida(red) ∧ UsuarioValido(u) ∧ Pertenece(u, usuarios(red))}
  asegura: {(∀r : Relacion)(Pertenece(r, relaciones(red)) ∧ r1 = u → Pertenece(r2, res))}
  asegura: {(∀r : Relacion)(Pertenece(r, relaciones(red)) ∧ r2 = u → Pertenece(r1, res))}
  asegura: {(∀u2 : Usuario)(Pertenece(u2, res) → (∃r : Relacion)(Pertenece(r, relaciones(red)) ∧ ((r1 = u ∧ r2 = u2) ∨ (r2 = u ∧ r1 = u2))))}
  asegura: {SinRepetidos(res)}
}

```

Ejercicio 3

```

problema cantidadDeAmigos (red:RedSocial, u:Usuario) : Z {
  requiere: {RedSocialValida(red) ∧ UsuarioValido(u) ∧ Pertenece(u, usuarios(red))}
  asegura: {res = |amigosDe(red, u)|}
}

```

Ejercicio 4

```

problema usuarioConMasAmigos (red:RedSocial) : Usuario {
  requiere: {RedSocialValida(red) ∧ |usuarios(red)| > 0}
  asegura: {Pertenece(res, usuarios(red)) ∧ (∀u : Usuario)(Pertenece(u, usuarios(red)) → (cantidadDeAmigos(red, u) ≤ cantidadDeAmigos(red, res)))}
}

```

Ejercicio 5

```

problema estaRobertoCarlos (red:RedSocial) : Bool {
  requiere: {RedSocialValida(red)}
  asegura: {res = true ⇔ (∃u : Usuario)(Pertenece(u, usuarios(red)) ∧ cantidadDeAmigos(red, u) > 1000000)}
}

```

Ejercicio 6

```

problema publicacionesDe (red:RedSocial, u:Usuario) : seq<Publicacion> {
  requiere: {RedSocialValida(red) ∧ UsuarioValido(u) ∧ Pertenece(u, usuarios(red))}
  asegura: {(∀pub : Publicacion)((Pertenece(pub, publicaciones(red)) ∧ (usuarioDePublicacion(pub) = u)) →
    Pertenece(pub, res))}
  asegura: {((∀pub : Publicacion)(Pertenece(pub, res) → ((usuarioDePublicacion(pub) = u) ∧
    Pertenece(pub, publicaciones(red)))))}
  asegura: {SinRepetidos(res)}
}

```

Ejercicio 7

```

problema publicacionesQueLeGustanA (red:RedSocial, u:Usuario) : seq<Publicacion> {
  requiere: {RedSocialValida(red) ∧ UsuarioValido(u) ∧ Pertenece(u, usuarios(red))}
  asegura: {(∀pub : Publicacion)((Pertenece(pub, publicaciones(red)) ∧ Pertenece(u, likesDePublicacion(pub))) →
    Pertenece(pub, res))}
  asegura: {(∀pub : Publicacion)(Pertenece(pub, res) → (Pertenece(u, likesDePublicacion(pub)) ∧
    Pertenece(pub, publicaciones(red)))))}
  asegura: {SinRepetidos(res)}
}

```

Ejercicio 8

```

problema lesGustanLasMismasPublicaciones (red:RedSocial, u1:Usuario, u2:Usuario) : Bool {
  requiere: {RedSocialValida(red) ∧ UsuarioValido(u1) ∧ UsuarioValido(u2) ∧ Pertenece(u1, usuarios(red)) ∧
    Pertenece(u2, usuarios(red))}
  asegura: {res = true ⇔
    MismosElementos(publicacionesQueLeGustanA(red, u1), publicacionesQueLeGustanA(red, u2))}
}

```

Ejercicio 9

```

problema tieneUnSeguidorFiel (red:RedSocial, u:Usuario) : Bool {
  requiere: {RedSocialValida(red) ∧ UsuarioValido(u) ∧ Pertenece(u, usuarios(red))}
  asegura: {res = true ⇔ (∃u2 : Usuario)(Pertenece(u2, usuarios(red)) ∧ u ≠ u2 ∧
    (∀pub : Publicacion)(Pertenece(pub, publicaciones(red)) ∧ usuarioDePublicacion(pub) = u →
    Pertenece(u2, likesDePublicacion(pub)))) ∧ |publicacionesDe(red, u)| > 0}
}

```

Ejercicio 10

```

problema existeSecuenciaDeAmigos (red:RedSocial, u1:Usuario, u2:Usuario) : Bool {
  requiere: {RedSocialValida(red) ∧ UsuarioValido(u1) ∧ UsuarioValido(u2) ∧ Pertenece(u1, usuarios(red)) ∧
    Pertenece(u2, usuarios(red))}
  asegura: {(res = true ⇔ (∃us : seq<Usuario>)(|us| ≥ 2 ∧ EmpiezaCon(u1, us) ∧ TerminaCon(u2, us)
    ∧ SonDeLaRed(red, us) ∧ CadenaDeAmigos(us, red)))}
}

```