



**UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA**

Paradigmas de Programación

Laboratorio N°1

Alumno: Cristhofer Parada Salazar
Sección: B-2.
Profesor: Víctor Flores Sánchez.

TABLA DE CONTENIDOS

Índice de Figuras	4
CAPÍTULO 1. Introducción	5
1.1.1 Descripción del problema.	5
1.1.2 Descripción del paradigma utilizado.	5
CAPÍTULO 2. Solución del problema.	6
2.1.1 Análisis del problema respecto a sus requisitos específicos.	6
2.1.2 Diseño de la solución.	7
2.1.3 Aspectos de implementación.	9
2.1.4 Instrucciones de uso.	10
2.2 Resultados y autoevaluación.	11
2.3 Conclusión.	12
CAPÍTULO 3. Anexo.	13

CAPÍTULO 1. INTRODUCCIÓN

El siguiente informe tiene como objetivo mostrar al lector todos los detalles relacionados al Laboratorio 1 de Paradigmas de Programación, los cuales a grandes rasgos pueden ser resumidos como descripción del problema, descripción del paradigma utilizado, el cómo se resolvió el problema y las diversas estructuras o TDAs que se utilizaron para la implementación de la solución, además de la forma de utilizar este programa a través de Dr Racket y la explicación de las diversas funciones implementadas.

1.1.1 Descripción del problema

En el semestre 1/2021, a los alumnos de la asignatura de Paradigmas de Programación, se les solicitó la implementación de una Red Social, la cual tenía como funciones principales acciones como registrarse en la red social, crear publicaciones, comentar publicaciones, seguir a otros usuarios, etc. Para ello, se les solicitó a los alumnos que el programa debiese estar escrito en el lenguaje de Scheme y separar el programa en diversos archivos, específicamente un archivo para cada TDA, siendo un TDA la abstracción de datos que pueden representar un cierto

1.1.2 Descripción del paradigma utilizado

En esta ocasión, se solicitó que se utilizara el paradigma funcional para crear la solución al problema. El paradigma funcional tiene características bastante representativas, como el hecho que se evitan los datos mutables, y las funciones siguen la notación lambda, así como podemos ver que las funciones pueden ser parte del dominio o recorrido de otras funciones. Este lenguaje tiene como enfoque el “qué” estamos haciendo y no el “como” se esta haciendo.

CAPÍTULO 2. SOLUCIÓN DEL PROBLEMA

2.1.1 Análisis del problema respecto a sus requisitos específicos.

Para el desarrollo de la solución se solicitan los siguientes requisitos funcionales específicos:

- Implementar abstracciones apropiadas para el problema mediante el uso de TDAs siendo un TDA llamado SocialNetwork el que estará principalmente enfocado el programa
- Función register: Función que permite que una persona se registre en una red social existente, para ello se le solicita una red social, un nombre de usuario, una contraseña e ingresar la fecha en la que se está registrando
- Función login: Permite que un usuario que tenga cuenta en la red social inicie sesión y realice una acción.
- Función post: función que permite a un usuario con sesión iniciada en la red social, realice una publicación.
- Función follow: permite a un usuario con sesión iniciada en la red social siga a otro usuario, siempre y cuando este ultimo exista en la red social.
- Función share: permite a un usuario con sesión iniciada compartir una publicación, siempre y cuando el autor de dicha publicación siga al usuario con la sesión iniciada.
- Función socialnetwork->string: función que tiene como propósito generar un string para poder ser mostrada con claridad. Si se inicia como parámetro para la función login, retorna un string con la información del usuario loggeado. Si se ocupa fuera de los parámetros de login, permite entregar toda la información que se encuentre en la red social.

Además de las funciones antes mencionadas, se da la libertad de escoger otras funciones que se pueden implementar en el programa, específicamente en este caso, se implementaron las funciones de comment, like y viral

- Función comment: permite a un usuario comentar una publicación siempre y cuando esta última exista.
- Función like: permite a un usuario con sesión iniciada darle like a una publicación, siempre y cuando esta última exista.
- Función viral: permite consultar cuales publicaciones dentro de la red social han llegado a cierto numero de compartidos.

2.1.2 Diseño de la solución.

La implementación para la solución de este problema en un principio puede llegar a ser tedioso, porque antes que todo hay que analizar el problema y pensar en los diversos posibles TDAs que pueden servir para resolver el caso. En nuestro caso, se crearon 4 TDAs que le darían solución al problema, los cuales se presentaran a continuación:

TDA Fecha:

Dada la gran importancia que toma las fechas en este programa (puesto que se utilizan en casi todas las funciones) se decidió asignarle su respectivo TDA, el cual será útil para verificar si las fechas ingresadas corresponden a lo que sería una fecha en la realidad. La representación de este TDA vendría dada por 3 números enteros, el primero correspondería al día, el segundo al mes y el tercero al año.

TDA User:

Corresponde a la presentación de un usuario en una red social. El usuario estaría representado por un ID único, que se genera al momento de registrarse en la red social, un nombre de usuario, una contraseña, una fecha que correspondería a la fecha de creación de la cuenta en la red social, una lista con los usuarios que siguen al usuario y una lista con las publicaciones hechas o compartidas por el usuario.

TDA publicación:

TDA correspondiente a lo que sería una publicación dentro de la red social, la cual estaría representada por una ID única dentro de la red social, el texto correspondiente a lo que queremos publicar, una lista con IDs de usuarios que le han dado “like” a la publicación, una lista con IDs de otras publicaciones que corresponderían a comentarios de la publicación, una fecha de creación de la publicación, una lista de IDs de usuarios etiquetados y una lista de IDs de usuarios que han compartido la publicación.

TDA SocialNetwork:

Principal TDA del programa, el cual correspondería a lo que sería una red social dentro del programa, este TDA está compuesto por un nombre de la red social, una fecha de creación, una lista con todos los usuarios que se han registrado, una lista con las publicaciones que se han creado en la red social, un número que corresponde al usuario con sesión activa, una función para encriptar y una función para desencriptar. A continuación, se mostrará un ejemplo de una red social vacía:

```
> (crearRedSocial "twitter1" '(25 5 2021) encrypt encrypt)
'("twitter1" (25 5 2021) () () 0 #<procedure:encrypt> #<procedure:encrypt>)
> |
```

Figura 1: Creación de una red social con 0 usuarios registrados, 0 publicaciones y ningún usuario en línea (vacía)

Con una red social creada podemos registrar un nuevo usuario en ella, pero para ello, debemos comprobar primero que el usuario ingresado no exista ya en la red social, por lo que viene la implementación de una técnica en programación que se conoce como una función recursiva natural (puesto que en este paradigma no existen las funciones que hacen ciclos), la cual permite que dada una red social, un nombre de usuario y una contraseña (datos correspondiente al nuevo usuario que se quiere registrar) verifique que el usuario ya no exista, dicha función se muestra a continuación:

```
137 ; Descripción: función que permite agregar un usuario a la lista de usuarios
138 ; Dominio: TDA User x lista
139 ; Recorrido: TDA SocialNetwork
140 (define agregarUsuarioRS
141   (lambda (usuario listaUsuarios)
142     (if (equal? listaUsuarios null)
143         (list usuario)
144         (if (equal? (getUsername usuario) (getUsername (car listaUsuarios)))
145             #f
146             (cons (car listaUsuarios) (agregarUsuarioRS usuario (cdr listaUsuarios)))
147         )
148     )
149 )
150 )
```

Figura 2: Función de tipo recursiva natural

La función anterior en la línea 144 comprueba si el username del usuario que se quiere registrar ya existe en la red social, de ser así retorna el bool false, caso contrario crea una lista con el primer usuario con el que se comparo el nombre y hace el llamado recursivo con el resto de la lista de usuarios que se encuentra en la red social, si la lista de usuarios de la red social es nula (caso base) se agrega el usuario que se quiere registrar a la lista de usuarios de la red social, retornando dicha lista.

Comprobado que el usuario que se quiere registrar no existe en la red social, la función register procede a crear una nueva red social modificando solamente el dato de la lista de usuarios, la cual ahora contendría al usuario recién registrado.

Dado un usuario registrado, podemos comprobar la siguiente función la cual vendría a ser login, la cual recibe una red social, un nombre de usuario, una contraseña y el ultimo parámetro corresponde a una función, que puede ser cualquiera que se le pueda aplicar a la red social. La función login es una función currificada, lo que significa que recibe los parámetros de entrada en distintos momentos, es decir, evaluar un primer conjunto de variables y retorna otra función que espera el siguiente conjunto de variables.

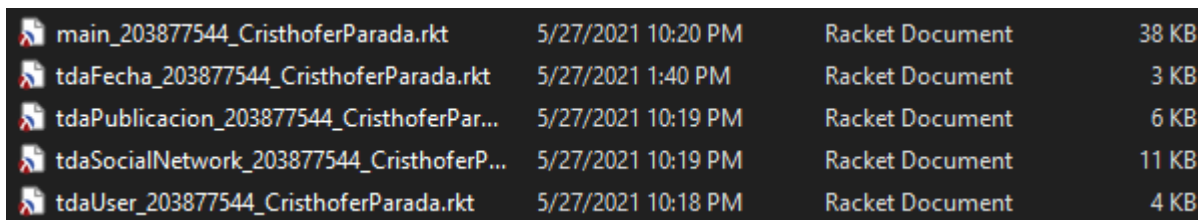
Para el resto de las funciones el login es esencial, puesto que como posee en su cuarto parámetro una función, es a ella la que se le aplicarían el resto de funciones, como por ejemplo la función post, solo toma sentido cuando tiene un usuario con sesión iniciada, y esto se logra a través de la función login.

```
(define (follow redSocial)
  (lambda (fecha)
    (lambda (user)
      (if (equal? (getUserID (IDtoUser redSocial (getUsuarioOnline redSocial))) (getUserID (UsernametoUser redSocial user))) ;Verificar si es el mismo usuario
          (display "El usuario no se puede seguir a si mismo")
          (crearRedSocialAux (getNombreRedSocial redSocial)
                             (getFechaRedSocial redSocial)
                             (actualizarUsuario (getUsuariosRedSocial redSocial)
                                                  (crearUsuario (getUserID (UsernametoUser redSocial user))
                                                                (getUsername (UsernametoUser redSocial user))
                                                                (getPassword (UsernametoUser redSocial user))
                                                                (getUserFecha (UsernametoUser redSocial user))
                                                                (addNumber (getUserFollowers (UsernametoUser redSocial user)) ; agrego el número de seguidores
                                                                (getUserID (IDtoUser redSocial (getUsuarioOnline redSocial))))
                                                  (getUserPosts (UsernametoUser redSocial user))
                                                  )
                             )
          (getPublicacionesRedSocial redSocial)
          0
          (getEncryptFn redSocial)
          (getDecryptFn redSocial)
          )
      )
    )
  )
)
```

Figura 3: Ejemplo de función currificada

2.1.3 Aspectos de implementación.

Este programa fue escrito en el lenguaje de Scheme, el cual es interpretado por DrRacket. No hay uso de librerías, puesto que solamente se usaron las funciones nativas del lenguaje de Scheme. Los archivos del programa se pueden ver de la siguiente manera:

A screenshot of a file explorer window with a dark background. It displays five files, each with a small icon of a document with a red 'X' in the top left corner. The files are listed in a table-like format with four columns: filename, date and time, file type, and size.

main_203877544_CristhoferParada.rkt	5/27/2021 10:20 PM	Racket Document	38 KB
tdaFecha_203877544_CristhoferParada.rkt	5/27/2021 1:40 PM	Racket Document	3 KB
tdaPublicacion_203877544_CristhoferPar...	5/27/2021 10:19 PM	Racket Document	6 KB
tdaSocialNetwork_203877544_CristhoferP...	5/27/2021 10:19 PM	Racket Document	11 KB
tdaUser_203877544_CristhoferParada.rkt	5/27/2021 10:18 PM	Racket Document	4 KB

Figura 4: Organización del programa

El programa esta compuesto por 5 archivos .rkt, en el archivo

“main_203877544_CristhoferParada.rkt” se encuentras las funciones descritas en la sección 2.1.1 además de otras funciones auxiliares que aportan al funcionamiento del programa.

En el archivo “tdaSocialNetwork_203877544_CristhoferParada.rkt” se encuentran todas las funciones correspondientes al TDA SocialNetwork (constructores, selectores, verificadores, modificadores y otras funciones auxiliares que aportan al TDA).

El archivo “tdaUser_203877544_CristhoferParada.rkt” contiene todas las funciones correspondientes al TDA User, tales como constructores, modificadores, selectores y verificadores, además de otras funciones que aportan al TDA.

El archivo “tdaPublicacion_203877544_CristhoferParada.rkt” corresponde a la definición del TDA publicación y contiene todas las funciones correspondientes al TDA y otras funciones auxiliares que aportan al funcionamiento de este.

El archivo “tdaFecha_203877544_CristhoferParada.rkt” contiene al TDA fecha y todas las funciones correspondientes a este TDA.

2.1.4 Instrucciones de uso.

En primer lugar, se debe verificar que todos los archivos anteriores se encuentren en la misma carpeta de raíz, de otro modo DrRacket arrojará un error que no encuentra archivos necesarios para la ejecución del main.

Luego, se debe abrir el archivo “main_203877544_CristhoferParada.rkt” y hacerlo correr buscando el botón “Run” en la esquina superior derecha. Con esto el programa ya estaría en ejecución a la espera de que se ingresen los comandos.

Lo primero que podemos hacer para seguir el curso normal del programa, sería crear una red social vacía, lo cual puede ser logrado por el comando “crearRedSocial <nombre> <fecha> <encrypt> <encrypt>” lo cual devuelve una red social vacía. Con esto hecho, podemos registrar un usuario de nombre “Harunomi”, con una contraseña “123” de la siguiente manera: “(register <nombreRedSocial> <fecha de registro> <”Harunomi”> <”123”>)”. Lo cual nos retornaría una red social nueva con el usuario “Harunomi” registrado en la lista de usuarios.

Dentro de la función main.rkt en el encabezado de cada función, se encontrarán comentarios con diversas maneras en las que se puede probar cada función, es recomendable que se ejecuten en el orden que se muestran.

Una forma en la que puedes ver de mejor manera el contenido de una red social es mediante la función “socialnetwork->string” la cual, dada una red social existente, permite mostrar información referente a ella de una manera mucho más comprensible que la dada por DrRacket. Una forma en la que se puede usar es la siguiente (asumiendo al existencia de una red social) “(display (socialnetwork->string <red social>))”

Dada la naturaleza de este paradigma, se recomienda que cada comando que se haga sea re-definido con nombres que sean comprensivos, esto es para de alguna forma “ir guardando” las redes sociales que son resultantes de los comandos utilizados.

2.2 Resultados y autoevaluación.

En esta sección se hará una revisión de los resultados, para ello se verificarán los requisitos no funcionales y funcionales solicitados

Requerimientos no funcionales:

Lenguaje Scheme obligatorio	Logrado: el lenguaje usado en el programa es Scheme.
Versión: Usar DrRacket versión 6.11 o superior.	Logrado: la versión de DrRacket usada es la 8.0
Estándar: Se deben usar funciones estándares del lenguaje.	Logrado: Solo se usan funciones estándares de Scheme.
No usar variables.	Logrado: No hay ningún tipo de variable en este programa o funciones que simulen variables.
Documentación de funciones.	Logrado: Todas las funciones cuentan con su encabezado que corresponde a la descripción de la función, su dominio y su recorrido, además de ejemplos.
Dom -> Rec	Logrado: Se respeta las entradas y salidas de las funciones, sin efectos colaterales.
Organización	Logrado: cada TDA esta representado en su respectivo .rkt permitiendo una buena organización del programa
Historial: al menos 10 commits en un mínimo de 2 semanas, con un mínimo de 10 commits.	Logrado: primer commit hecho el 2 de mayo de 2021, último commit hecho el 27 de mayo de 2021, con un total de 15 commits.
Ejemplos:	Logrado: Cada función principal en el main.rkt cuenta con un mínimo de 3 ejemplos
Prerrequisitos de funciones:	Logrado, cada función tiene su prerrequisito implementado como se solicitó.

Requerimientos Funcionales:

Todas las funciones cumplen con los requerimientos funcionales exigidos por la coordinación, por lo que este punto también fue respetado en su gran totalidad.

2.3 Conclusión.

Tomando en consideración los datos entregados en el punto 2.2, puedo decir que estoy bastante conforme con la realización de este programa. En un comienzo costó bastante trabajar con variables inmutables, pero dado el hecho que se tuvo en cuenta bien el concepto de TDA y todas las funciones que ayudan al TDA (constructores, modificadores, selectores, verificadores, etc) permitió que la implementación de las funciones en el main se simplificaran considerablemente, puesto que para cada acción que se requería implementar, estaban involucradas dichas funciones pertenecientes al TDA.

Aspectos a mejorar para futuros laboratorios pueden ser pensar de mejor manera como puede ser un TDA, puesto que ha medida que se iba avanzando en las funciones, iban surgiendo pequeñas necesidades de modificar el TDA o algún selector de algún TDA, esto se pudo haber evitado si se hubiese realizado una mejor lectura del problema en la que se proyecte con una gran visión todas las posibles cosas necesarias para llevar a cabo la solución del problema.

CAPÍTULO 3. ANEXO

```
; (define emptyTwitter (crearRedSocial "Twitter" '(20 5 2021) encrypt encrypt))
; (define emptyFacebook (crearRedSocial "Facebook" '(20 5 2021) encrypt encrypt))
; (define emptyRs (crearRedSocial "RedSocial" '(20 5 2021) encrypt encrypt))

; (define twitter1 (register emptyTwitter '(20 5 2021) "Haru" "123"))
; (define twitter2 (register twitter1 '(20 5 2021) "Haru" "123")) ejemplo de usuario ya existente
; (define twitter3 (register(register(register emptyTwitter '(25 5 2021) "Harunomi" "321") '(25 5 2021) "Cybele" "uwu") '(25 5 2021) "Hylia" "ily"));
```

Anexo 1: creación de redes sociales y register

```
; (define emptyTwitter (crearRedSocial "Twitter" '(20 5 2021) encrypt encrypt))
; (define emptyFacebook (crearRedSocial "Facebook" '(20 5 2021) encrypt encrypt))
; (define emptyRs (crearRedSocial "RedSocial" '(20 5 2021) encrypt encrypt))
; (define twitter1 (register emptyTwitter '(20 5 2021) "Haru" "123"))
; (define twitter2 (register twitter1 '(20 5 2021) "Haru" "123"))
; (define twitter3 (register(register(register emptyTwitter '(25 5 2021) "Harunomi" "321") '(25 5 2021) "Cybele" "uwu") '(25 5 2021) "Hylia" "ily"))
; (define twitter4 (login twitter3 "Harunomi" "321" display))
; (define twitter5 (login twitter3 "Cybele" "uwu" car))
; (define twitter6 (login twitter3 "Hylia" "ily" cdr))
; (define twitter7 ((login twitter2 "Harunomi" "321" post) '(20 5 2021)) "Mi primer HolaMundo")
; (define twitter8 ((login twitter7 "Hylia" "ily" post) '(20 5 2021)) "Contenta en la nueva red Social")
; (define twitter9 ((login twitter8 "Cybele" "uwu" post) '(20 5 2021)) "gracias por invitarme a la red social" "Harunomi")
; (define twitter10 ((login twitter8 "Cybele" "uwu" follow) '(22 5 2021)) "Harunomi")
; (define twitter11 ((login twitter10 "Hylia" "ily" follow) '(23 5 2021)) "Harunomi")
; (define twitter12 ((login twitter11 "Harunomi" "321" follow) '(23 5 2021)) "Hylia")
; (define twitter13 ((login twitter12 "Harunomi" "321" share) '(23 5 2021)) 2)
; (define twitter14 ((login twitter13 "Harunomi" "321" share) '(23 5 2021)) 3)
; (define twitter15 ((login twitter14 "Cybele" "uwu" share) '(23 5 2021)) 1) ;
; (display (socialnetwork->string twitter14))
; (display (socialnetwork->string twitter13))
; (login twitter14 "Harunomi" "321" socialnetwork->string)
; (define twitter17 ((login twitter14 "Cybele" "uwu" comment) '(28 5 2021)) 1) "Dame follow")
; (define twitter18 ((login twitter17 "Harunomi" "321" comment) '(28 5 2021)) 4) "De inmediato"))
; (define twitter19 ((login twitter18 "Hylia" "ily" comment) '(28 5 2021)) 5) "No lo sigas, es una cuenta maliciosa"))
; (define twitter20 ((login twitter19 "Harunomi" "321" like) '(27 5 2021)) 2))
; (define twitter21 ((login twitter20 "Hylia" "Ily" like) '(27 5 2021)) 2))
; (define twitter22 ((login twitter21 "Harunomi" "321" like) '(27 5 2021)) 4))
; (viral twitter22 2)
; (viral twitter22 1)
; (viral twitter22 4)
```

Anexo 2: totalidad de scripts de pruebas