



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

Paradigmas de Programación

Laboratorio N°1

Alumno: Israel Arias Panez.
Sección: B-2.
Profesor: Víctor Flores Sánchez.

Santiago - Chile
2-2020

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
2.4 Referencias.	12
CAPÍTULO 3. Anexo.	13

Índice de Figuras

Figura 1: Representación del stack	7
Figura 2: Stack dentro del programa	7
Figura 3: Stack con preguntas, respuestas y usuarios	7
Figura 4: Función recursiva	8
Figura 5: Función currificada	8
Figura 6: Estructura del programa	9

CAPÍTULO 1. INTRODUCCIÓN

En el presente informe se describirá el problema a solucionar del laboratorio n°1 de la asignatura de Paradigmas de programación. Se hará una descripción del problema, un análisis de la solución propuesta para su solución, cosas consideradas para su implementación, instrucciones de uso, para finalmente hacer una retroalimentación de los resultados y determinar conclusiones respecto al cumplimiento de la solución.

1.1.1 Descripción del problema

Como alumnos de la asignatura de Paradigmas de programación, para el presente semestre 2/2020, se nos ha encomendado la tarea de hacer una simulación de un software de sistema de foros, tomando como referencia el conocido foro web “StackOverflow”, para lograrlo, se debe implementar características que usa un foro, como por ejemplo: registrar usuarios, crear preguntas, responder preguntas, además cada usuario tiene características como username, contraseña, reputación, todas las preguntas que ha hecho, etc. Las preguntas tienen cantidad de votos, respuestas a la pregunta, fecha de publicación, recompensa por la pregunta, entre otros. Cada laboratorio tiene requisitos específicos a ser implementados, los cuales serán detallados en las siguientes secciones.

1.1.2 Descripción del paradigma utilizado

Para esta primera entrega del laboratorio del ramo, se solicita hacer uso del paradigma funcional en conjunto del lenguaje de programación Scheme, la principal característica del paradigma funcional que se utilizara en el desarrollo de la solución de este laboratorio es la de la transparencia referencial, la cual asegura que bajo una misma entrada, la función siempre tendrá la misma salida, pudiendo facilitar bastante el trabajo en ese sentido, ya que al modificar una función de las que compone la solución, no tendrá efectos en los resultados de las demás funciones, otra característica principal del paradigma es que los programas solo se componen de funciones, cosa que también se ve reflejada en la solución del problema.

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:

- Implementación de distintos TDA's para cumplir con todos los siguientes requisitos funcionales, con el TDA stack como base del programa.
- Función register: permite registrar a un usuario en el stack.
- Función login: permite a un usuario ya registrado iniciar sesión, lo que le permitirá hacer ciertas funciones.
- Función ask: permite a un usuario con sesión iniciada hacer una pregunta, la cual se agrega al stack.
- Función reward: permite a un usuario con sesión iniciada ofrecer una recompensa por una pregunta.
- Función answer: permite a un usuario con sesión iniciada responder a una pregunta, la respuesta se agrega al stack.
- Función accept: permite a un usuario con sesión iniciada aceptar una respuesta a una de sus preguntas, lo que entregara la recompensa y modificara la reputación del usuario cuya respuesta fue aceptada.
- Función stack->string: función que transforma todo el stack a un string posible de visualizar de forma comprensible por el usuario, el string puede ser mostrado en pantalla con la función display o la función write.

También se solicita implementar una función complementaria dentro de una lista de cinco funciones, para este laboratorio se eligió implementar la función vote:

- Función vote: permite a un usuario con sesión iniciada, votar positiva o negativamente una pregunta o respuesta que se encuentre en el stack, al hacerlo se registrara el voto y se sumara o restara reputación a los usuarios según corresponda.

2.1.2 Diseño de la solución.

El enfoque aplicado para diseñar la solución de este primer laboratorio consistió en una división de problemas en subproblemas, el cual se logro planear de una manera bastante fácil gracias a la organización de las instrucciones del laboratorio, los subproblemas derivados de esta división fue la misma en el mismo orden que la lista de requerimientos listada en el punto 2.1.1 de este informe. Partiendo con la definición de todos los TDA, con el TDA stack como base de todo el programa y otros TDA, por lo que muchas veces dentro del TDA stack se ejecutarán funciones de otros TDA. Los 5 TDA implementados para la creación de la solución fueron: StackTDA, respuestaTDA, preguntaTDA, fechaTDA y UsuariosTDA. También fue creado un archivo main, el cual solo contiene la definición de las funciones obligatorias solicitadas. Una vez creado todos los TDA, se siguió la construcción de la solución con la implementación de las funciones register, login, ask, reward, answer, accept, stack->string y finalmente vote.

El stack TDA tiene la siguiente estructura:

```
;Representacion
;(list X list X list X list)
;(list preguntas respuestas usuarios nombreUsuarioActivo)
```

Figura 1: Representación del Stack

```
> (crearStack)
(() () () ())
~|
```

Figura 2: Stack dentro del programa.

El cual es una lista que tiene cuatro listas dentro, la primera lista contendrá las preguntas que se hagan en el foro, la segunda lista contendrá las respuestas a las preguntas que se hagan en el foro, la tercera lista tendrá a todos los usuarios con su respectiva información y la cuarta lista contendrá al usuario que se encuentre conectado a través de la función login.

```
((("como hago hola mundo" 1 "israel" (14 11 2020) ("python" "c") "abierta" 0 0) ("buenos dias" 2 "juan" (14 11 2020) ("saludos") "abierta" 0 0))
(("pone print" 1 1 "juan" (14 11 2020) ("asistencia") "pendiente" 0) ("que sucede" 2 2 "israel" (14 11 2020) ("comunidad") "pendiente" 0))
(("israel" "123" (1) 500) ("juan" "qwerty" (2) 500))
())
```

Figura 3: Stack con preguntas, respuestas y usuarios.

Para lograr la implementación de algunas funciones, antes de ejecutar la función es necesario verificar si era posible ejecutar la función, por ejemplo, en la función register no se puede permitir registrar a un usuario que ya esta registrado, entonces se debió implementar una función extra que recorriera todo el stack de usuarios, sin embargo en el paradigma funcional no existen funciones iteradoras como while o for, entonces para poder

recorrer toda la lista se hizo necesaria el uso de la técnica de recursión, para poder iterar sobre las listas

```
;descripción: Permite saber si un usuario ya se encuentra dentro del stack de usuarios.
;dom: lista X string
;rec: booleano
;tipo de recursión: natural
(define yaEstaRegistrado
  (lambda (listaUsuarios username)
    (if (null? listaUsuarios);Caso base alcanzado al llegar al final de la lista o si la lista se encuentra vacia
        #f ;No esta registrado
        (if (equal? (car(listaUsuarios)) username) ;Segundo caso base, compara el nombre del usuario de la posición del stack con username
            #t ; Ya se encuentra registrado
            (yaEstaRegistrado (cdr listaUsuarios) username) ;Descomposicion recursiva, se pasara al siguiente usuario
        )
    )
  )
)
```

Figura 4: Función recursiva.

Por ejemplo, en la figura 3 se observa la función yaEstaRegistrado, la cual permite saber si el usuario que se quiere registrar, ya se encuentra dentro del stack, para ello se usa recursión, la cual permite ir posición por posición verificando el nombre de todos los usuarios registrados del stack, para pasar al siguiente usuario, se vuelve a llamar recursivamente a la función, entregándole el cuerpo (cdr) de la lista de usuarios. La recursión fue una técnica implementada en muchas funciones en este laboratorio.

Otra técnica muy usada para este laboratorio fue la de currificación “Una función currificada o parcializada es aquella que recoge más de un conjunto de parámetros de entrada en distintos momentos. La currificación recoge un primer conjunto de parámetros y devuelve como resultado una función que espera el o los siguientes conjuntos de los mismos, precargando los ya indicados como una instanciación parcial.”^[1]

Para las funciones siguientes a login fue fundamental el uso de la currificación, debido a que se necesitaba que login retornara una función parcialmente evaluada al stack actualizado, la cual debía entrar como argumento a la función que acompañara a login, por ende fue necesario currificar las funciones que necesitaban llamarse desde login, por ejemplo la función reward que se ve en la figura 4.

```
;descripción: Permite a un usuario logueado ofrecer una recompensa por una pregunta resuelta
;dom: stack X entero X entero
;rec: stack
(define reward
  (lambda (stack)
    (lambda (idPregunta)
      (lambda (recompensa)
        ;En primer lugar se verifica que el usuario se encuentre logueado
        (if (equal? (isUserLoggedIn? (stackGetActiveUsers stack)) #f)
            stack ;si no esta logueado, se retorna el stack sin cambios
            ;En caso de estar logueado se revisara que el id y la recompensa sean enteros
            (if (not(and (integer? idPregunta) (integer? recompensa)))
                stack ;no son enteros, se retorna el mismo stack sin cambios
                (if (equal? (puedeDarReputacion? (stackGetUsers stack) (stackGetLoggedUser stack) recompensa) #t)
                    (traspasarRecompensa (stackGetUsers stack) (stackGetQuestions stack) (stackGetLoggedUser stack) idPregunta recompensa stack)
                    stack ;no puede dar la recompensa, entonces se retorna el stack sin cambios
                )
            )
        )
      )
    )
  )
)
```

Figura 5: Función currificada.

2.1.3 Aspectos de implementación.

La implementación de este programa se hizo en el lenguaje Scheme, el cual es un lenguaje interpretado a través de DrRacket.

No hace uso de librerías, solo funciones nativas del lenguaje de programación Scheme.

El programa se encuentra estructurado de la siguiente manera:

1: main_20110122_AriasPanez.rkt	2: StackTDA_20110122_AriasPanez.rkt	3: preguntaTDA_20110122_AriasPanez.rkt	4: respuestaTDA_20110122_AriasPanez.rkt	5: UsuariosTDA_20110122_AriasPanez.rkt	6: fechaTDA_20110122_AriasPanez.rkt
---------------------------------	-------------------------------------	--	---	--	-------------------------------------

Figura 6: Estructura del programa.

Como es posible observar en la figura 5, existen 6 archivos .rkt que componen el programa, en main_20110122_AriasPanez.rkt se encuentran las definiciones de todas las funciones descritas en el punto 2.1.1 de este programa.

En StackTDA_20110122_AriasPanez.rkt se encuentra la definición del TDA stack, el cual es el principal de este programa y contiene la definición, representación y funciones de stack.

UsuariosTDA_20110122_AriasPanez.rkt, contiene la representación de los usuarios en el programa, junto a sus funciones definidas dentro del TDA.

preguntaTDA_20110122_AriasPanez.rkt, contiene la representación de las preguntas dentro del programa, junto a sus funciones definidas dentro del TDA.

respuestaTDA_20110122_AriasPanez.rkt, contiene la representación de las respuestas dentro del programa, junto a sus funciones definidas dentro del TDA.

Por último, fechaTDA_20110122_AriasPanez.rkt, contiene la representación de la fecha dentro del programa, junto a sus funciones definidas dentro del TDA.

2.1.4 Instrucciones de uso.

A continuación, se explicará las instrucciones para poder usar el programa de una manera adecuada:

- En primer lugar, verificar que se tengan los 6 archivos .rkt detallados en el punto 2.1.3 del informe en la misma carpeta.
- Para usar el programa primero se debe abrir el archivo `main.rkt_20110122_AriasPanez` (ver Anexo 1)
- Luego poner a correr el programa, con el botón run (ver Anexo 2)
- Para crear un stack vacío se puede usar la función `(crearStack)` en la consola.
- Por ejemplo, para registrar al usuario “pedro” con una contraseña “123” en el stack, se debe ingresar en consola `(register (crearStack) "pedro" "123")`
- Para poder visualizar todo de una manera más comprensiva se recomienda el uso de la función `stack-> string`, por ejemplo, para el visualizar el stack luego de haber registrado a pedro, se debe ingresar por consola `(display (stack->string (register (crearStack) "pedro" "123")))` lo cual al ejecutarse mostrará comprensiblemente el stack con el usuario pedro registrado.
- Para cada función en específico se incluyen ejemplos para su ejecución, las cuales se encuentran en el mismo `main.rkt`, ejemplos de ejecución se anexarán a este informe. (ver Anexo 3 y 4)
- Para poder ejecutar las funciones del main a modo de ejemplo se ha optado por ir definiendo de a poco el stack con las funciones, desde crear un stack vacío, hasta llegar a votar sobre una pregunta o respuesta, debido a que de otra manera, al llegar a la función `vote`, se tendría una cadena de comandos con demasiados caracteres que dificultaría la comprensión de la función que se está intentando ejecutar, en el `main_20110122_AriasPanez.rtk` al final se encuentra un resumen de ejemplos de ejecución con esta modalidad. Cabe destacar que las funciones no funcionarían si se borra una definición de un prerequisite de una función, por ejemplo, borrar la definición del constructor de stack, provocaría que ninguna función se logre ejecutar, ya que todas las funciones tienen como prerequisite la existencia de un stack.

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 7.8
Estándar: Se deben usar funciones estándares del lenguaje.	Logrado: Solo se usan funciones estándares de Scheme.
No usar variables.	Logrado: No se usan variables ni nada que simule el funcionamiento del paradigma imperativo.
Documentación de funciones.	Logrado: Todas las funciones implementadas se encuentran comentadas, se indica descripción, dominio y recorrido de cada función. En los casos que se usan recursiones, se indica su tipo.
Dom -> Rec	Logrado: Se respeta las entradas y salidas de las funciones, sin efectos colaterales.
Organización	Logrado: el programa se encuentra estructurado en archivos distintos, uno para cada TDA implementado, y dentro de main.rkt se encuentran las funciones obligatorias requeridas.
Historial: al menos 10 commits en un mínimo de 2 semanas, con un mínimo de 10 commits.	Logrado: primer commit hecho el 22 de octubre de 2020, ultimo commit hecho el 18 de noviembre de 2020, con un total de 15 commits.
Ejemplos:	Logrado, hay un mínimo de 3 ejemplos por cada función en el archivo main.rkt, los cuales se encuentran justo luego de haber definido la función.
Prerrequisitos de funciones:	Logrado, cada función tiene su prerrequisito implementado como se solicitó.

Requerimientos Funcionales:

Todos los requerimientos funcionales funcionan sin problemas, cumpliendo con las restricciones solicitadas para cada una de ellas, fueron probadas ingresándoles distintas entradas a cada función y verificando que la salida fuera la esperada de acuerdo con la entrada, se adjuntan ejemplos dentro del archivo main_20110122_AriasPanez.rkt que prueba esto.

2.3 Conclusión.

Considerando los resultados listados en el punto 2.2 se puede concluir que el resultado conseguido de la implementación de este primer laboratorio cumple con todos los objetivos propuestos inicialmente.

A pesar de que la característica de la transparencia referencial facilitó bastante el debug de las funciones implementadas, debido a que al cambiar una función no cambiaba todo lo demás (efectos colaterales) como ocurre a veces en el paradigma Imperativo, si fue mucho más dificultoso el implementar el sistema de foro, debido a que por costumbre como alumno he estado programando bajo el paradigma imperativo y ahora cambiar la manera de programar y perder funciones como los iteradores while o for y en especial la definición de variables, dificultó bastante en un principio la programación. Sin embargo, también me hizo dar cuenta de todas las ventajas que ofrece el paradigma funcional, siendo el más importante sin duda alguna la transparencia referencial.








2.4 Referencias

- [1] Fernández J. (2015). *Funciones curried o parciales*.

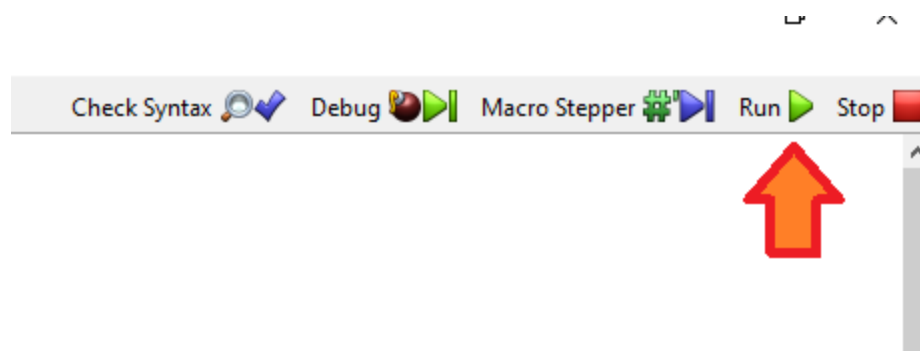
(Recuperado 16/11/2020).

<https://applecoding.com/cursos/swift-avanzado-funciones-curried-parciales>

CAPÍTULO 3. ANEXO

Nombre	Fecha de modificación	Tipo	Tamaño
 fechaTDA_20110122_AriasPanez.rkt	14-11-2020 3:33	Racket Document	2 KB
 main_20110122_AriasPanez.rkt	18-11-2020 22:25	Racket Document	13 KB
 preguntaTDA_20110122_AriasPanez.rkt	14-11-2020 22:40	Racket Document	3 KB
 README.md	15-11-2020 18:03	Archivo MD	1 KB
 respuestaTDA_20110122_AriasPanez.rkt	14-11-2020 22:44	Racket Document	3 KB
 StackTDA_20110122_AriasPanez.rkt	18-11-2020 22:25	Racket Document	27 KB
 UsuariosTDA_20110122_AriasPanez.rkt	12-11-2020 21:42	Racket Document	2 KB

Anexo 1: Abrir el archivo main_20110122_AriasPanez.rkt para poder ocupar el programa.



Anexo 2: ejecución del programa main_20110122_AriasPanez.rkt

```

Welcome to DrRacket, version 7.8 [3m].
Language: scheme, with debugging; memory limit: 128 MB.
> ;Create Stack:

(define stackOverflow(crearStack))

;Ejemplos register:

(define stackOverflow(register stackOverflow "pedro" "123"))
(define stackOverflow(register stackOverflow "israel" "qwerty"))
(define stackOverflow(register stackOverflow "maria" "456"))

;Ejemplos login:

(login stackOverflow "israel" "qwerty" stack->string)
(login stackOverflow "maria" "456" stack->string)
(login stackOverflow "pedro" "123" stack->string)

;Ejemplos ask:

(define stackOverflow(((login stackOverflow "israel" "qwerty" ask) (date 16 11 2020))"Hola ¿como puedo hacer hola mundo?" "python"))
(define stackOverflow(((login stackOverflow "pedro" "123" ask) (date 17 11 2020))"Soy nuevo, como puedo usar el foro ?" "ayuda" "usuario nuevo"))
(define stackOverflow(((login stackOverflow "maria" "456" ask) (date 17 11 2020))"Como puedo hacer iteraciones en scheme ?" "paradigma funcional" "scheme"))

;Ejemplos reward:

(define stackOverflow(((login stackOverflow "pedro" "123" reward)2)300))
(define stackOverflow(((login stackOverflow "israel" "qwerty" reward)3)50))
(define stackOverflow(((login stackOverflow "israel" "qwerty" reward)1)100))

;Ejemplos answer:

(define stackOverflow(((login stackOverflow "israel" "qwerty" answer) (date 18 11 2020))2) "Hola, te recomiendo leer el informe y los ejemplos del archivo main.rtk" "duda" "foro"))
(define stackOverflow(((login stackOverflow "pedro" "123" answer) (date 18 11 2020))3) "Hola, hay que usar recursión a fin de iterar sobre algo en scheme" "recursion" "scheme"))
(define stackOverflow(((login stackOverflow "maria" "456" answer) (date 19 11 2020))1) "Puedes hacer uso de la funcion print en python para ello" "duda" "python"))

;Ejemplos accept:

(define stackOverflow(((login stackOverflow "maria" "456" accept)3)2))
(define stackOverflow(((login stackOverflow "israel" "qwerty" accept)1)3))
(define stackOverflow(((login stackOverflow "pedro" "123" accept)2)1))

;Ejemplos stack->string:

(stack->string stackOverflow)
(display (stack->string stackOverflow))
(login stackOverflow "israel" "qwerty" stack->string)

;Ejemplos vote:

(define stackOverflow(((login stackOverflow "israel" "qwerty" vote)(getQuestion)2>false))
(define stackOverflow(((login stackOverflow "pedro" "123" vote)(getAnswer 1))3>true))
(define stackOverflow (((login stackOverflow "maria" "456" vote)(getAnswer 2))1>false))
(define stackOverflow(((login stackOverflow "pedro" "123" vote)(getQuestion)3>true))

```

Anexo 3: Ejecución de resumen de funciones (disponibles al final de archivo main_20110122_AriasPanez.rkt)

```

Pregunta: Id: 1 Fecha de publicacion: 16/11/2020 Recompensa: 0 Votos: 0 Estado: cerrada
Hola ¿como puedo hacer hola mundo?
Etiquetas: python Autor: israel

Respuesta: Id: 3 Fecha de respuesta: 19/11/2020 Votos: 1 Estado: aceptada
Puedes hacer uso de la funcion print en python para ello
Etiquetas: duda python Autor: maria

Pregunta: Id: 2 Fecha de publicacion: 17/11/2020 Recompensa: 0 Votos: -1 Estado: cerrada
Soy nuevo, como puedo usar el foro ?
Etiquetas: ayuda usuario nuevo Autor: pedro

Respuesta: Id: 1 Fecha de respuesta: 18/11/2020 Votos: -1 Estado: aceptada
Hola, te recomiendo leer el informe y los ejemplos del archivo main.rtk
Etiquetas: duda foro Autor: israel

Pregunta: Id: 3 Fecha de publicacion: 17/11/2020 Recompensa: 0 Votos: 1 Estado: cerrada
Como puedo hacer iteraciones en scheme ?
Etiquetas: paradigma funcional scheme Autor: maria

Respuesta: Id: 2 Fecha de respuesta: 18/11/2020 Votos: 0 Estado: aceptada
Hola, hay que usar recursión a fin de iterar sobre algo en scheme
Etiquetas: recursion scheme Autor: pedro

Usuarios Registrados en el Foro:
Usuario:
Username: pedro Reputación: 265 Preguntas realizadas(Id): 2
Usuario:
Username: israel Reputación: 665 Preguntas realizadas(Id): 1
Usuario:
Username: maria Reputación: 636 Preguntas realizadas(Id): 3
>

```

Anexo 4: Stack impreso con (display(stack->string stackOverflow)) con el resultado de todas las funciones de ejemplo disponibles al final del archivo main_20110122_AriasPanez.rkt