INFORME DE LABORATORIO 1

Paradigmas de Programación

18 DE NOVIEMBRE DE 2020

NICOLÁS MARCELO VALDÉS HERRERA RUT: 19682704-8
Sección: D-4 Profesor: Gonzalo Martínez R

I. Contenido del informe

II. Introducción

Uno de los objetivos del ramo es el uso debido de lenguajes de programación que utilizan un paradigma funcional, un ejemplo de estos es Scheme, con el cual, utilizando Dr.Racket, se solicita para este laboratorio. Este laboratorio tiene como objetivo que el estudiante sea capaz de utilizar de forma debida Scheme para simular las funciones que se utilizan en un programa de preguntas y respuestas como lo es Stack Overflow.

III. Descripción del problema

Utilizando el lenguaje de programación Scheme, se le pide al estudiante implementar las funciones necesarias para simular Stack Overflow tomando en consideración lo explicado en el enunciado de laboratorio, para ello se pide crear TDAs necesarios para su implementación y la creación de las funciones “register”, “login”, “ask”, “reward”, ”answer”, “accept”, “stack->string”, “vote”(a elección), “ReportOffense”(a elección), “ranking”(a elección), “search”(a elección), “clean”(a elección).

Para la creación de las funciones solicitadas se piden prerrequisitos y requisitos de implementación descritos en el enunciado, una implementación específica, encabezado y ejemplos de su uso.

IV. Descripción del paradigma

Este paradigma se basa en funciones, aquellos lenguajes de programación que se rigen por este paradigma ven a todos los elementos como funciones y el código se ejecutó mediante llamadas de funciones secuenciales. Una característica es que no se asignan valores de forma directa. Las funciones se toman como subprogramas que se reutilizan y, a diferencia de un procedimiento, devuelven un resultado de forma directa.

Conceptos utilizados: programación declarativa, recursión, funciones como tipos de datos primitivos.

V. Análisis del problema

1. Funciones Globales: Antes de analizar los TDAs y funciones solicitadas en el laboratorio, se opina que es necesario un archivo en el cual contenga funciones de carácter global, esto para no repetir funciones dentro de los TDAs como en el Main.

2. TDA Respuestas: Después de leer el enunciado se llega a concluir que para guardar las respuestas se requiere de una lista que vaya guardando todas las respuestas publicadas por los usuarios registrados, en la que cada una tendrá la información necesaria para su debido uso durante todo el laboratorio. Una vez hecha la lista de respuestas, es necesario funciones de selección para extraer información de cada una de estas, así como también validadores que confirmen que la respuesta que se quiere ingresar tenga el formato correcto.

3. TDA Preguntas: Para poder tener todas las preguntas subidas por los usuarios registrados, es necesario una lista que vaya guardado cada una de estas con toda la información necesaria para su debida implementación en el laboratorio. Una vez hecha la lista de preguntas, es necesario funciones de selección para extraer información de cada una de estas, así como también validadores que confirmen que la pregunta que se quiere ingresar tenga el formato correcto. Se verifica que para poder crear TDA Preguntas es necesario implementar TDA Respuestas.

4. TDA Usuarios: Para poder tener todos los usuarios registrados, es necesario una lista que vaya guardando cada uno de estos con la información necesario para su implementación en el laboratorio. Una vez echa la lista de usuarios, es necesario funciones de selección para extraer información de cada uno de estos, así como también validadores, que confirmen que el usuario tenga el formato correcto. Se verifica que para poder implementar TDA Usuarios es necesario implementar TDA Preguntas y TDA Respuestas.

5. Register: Para poder usar esta función, se es necesario los parámetros mostrados en el enunciado estos son stack, usuario y contraseña. La idea de esta función es retornar un stack actualizado que contiene un usuario registrado. Para poder implementarlo es necesario crear una lista nueva en la que se introducen los datos de forma individual utilizando recursión natural.

6. Login: Para poder usar esta función, es necesario los parámetros mostrados en el enunciado, estos son stack, usuario, contraseña y operación. La idea de esta función es validar que el usuario este registrado y que la contraseña sea correcta. Para poder implementarlo, es necesario crear una lista nueva en la que se introducen los datos de forma individual utilizando recursión natural, a parte, el usuario del stack tiene que ser cambiado si el usuario este registrado y la contraseña sea correcta, en el caso de no serlo, se retorna la operación con el stack sin modificar.

7. Ask: Para poder usar esta función, es necesario los parámetros mostrados en el enunciado usando la función login, estos son stack, usuario, contraseña, operación ask, fecha, titulo de la pregunta, lista de etiquetas y contenido. La idea de esta función es subir una pregunta al stack, para ello es necesario crear una lista nueva en la que se incorporan los datos actualizados, estos serán subidos de forma individual usando recursión natural. La función retornara la lista stack con la pregunta agregada y actualizada si los parámetros de entrada son válidos, en el caso de no serlos o de que login no inicie sesión, retorna el stack ingresado sin la pregunta agregada.

8. Reward: Para poder usar esta función, es necesario los parámetros mostrados en el enunciado usando la función login, estos son stack, usuario, contraseña, operación reward, ID y recompensa. La idea de esta función es dar una recompensa retenida a una pregunta, para ello el usuario que quiere dar la recompensa debe tener una reputación mayor o igual a esta, en el caso de no tenerla o de que no se ejecute login, se retorna el stack ingresado no actualizado, en el caso de que las variables de entrada sean validas, se retorna un stack con las recompensas y reputaciones actualizadas. Para poder crear el stack, se incorporan los datos actualizados de forma individual usando recursión natural.

9. Answer: Para poder usar esta función, es necesario los parámetros mostrados en el enunciado usando la función login, estos son stack, usuario, contraseña, operación answer, fecha, ID y respuesta. La idea de esta función es de incorporar una respuesta a una pregunta, para que esto se cumpla se tiene que ejecutar el login y las variables de entrada tienen que ser validas, en el caso de no serlo se retorna un stack no actualizado, en el caso de que sean válidas, se retorna un stack con la respuesta incorporada a la pregunta señalada. Para la creación del stack se crea una lista nueva en la que se incorporan los datos actualizados de forma individual usando recursión natural.

10. Accept: Para poder usar esta función, es necesario los parámetros mostrados en el enunciado usando la función login, estos son stack, usuario, contraseña, operación accept, ID y correlativo. La idea de esta función es dar la recompensa de una determinada pregunta a una determinada respuesta, para ello se debe ejecutar login y las variables de entrada tienen que ser validas, en el caso de no serlo se retorna un stack no actualizado, en el caso de cumplir con los requisitos, se entrega un stack nuevo con las recompensas, preguntas y respuestas actualizados. Para la creación del stack se crea una lista nueva en la que se incorporan los datos actualizados de forma individual usando recursión natural.

VI. Diseño de solución

1. TDA Respuestas: Representación de un conjunto de respuestas las cuales serán utilizadas en el laboratorio, contiene las funciones necesarias para extraer y validar cada uno de los elementos que componen una respuesta. Cada respuesta consiste en una lista con múltiples elementos los cuales le dan formato. Los elementos que la componen son:

   (usuario fecha ID correlativo votos estado reportes respuesta recompensa)
   Ejemplo: (list "Usuario1" (list 12 5 2020) 1 1 (list 5 2) 1 0 "Respuesta 1.1" 0)

   Para extraer cada elemento de una respuesta, se crean funciones selectoras que se caracterizan por anteponer un “get”, así mismo, se crean funciones de pertenencia para validar que cada elemento de una respuesta tenga el formato correcto. Cabe destacar la existencia de dos funciones de obtención especial llamadas getRespuestas_usuario y getRespuestas_pregunta las cuales tienen un uso específico. getRespuestas_usuario obtiene todas las respuestas que un usuario específico a publicado, getRespuestas_pregunta entrega todas las respuestas publicadas en una pregunta en específico. Se crea una lista con todas las respuestas emitidas llamada listaR.

2. TDA Preguntas: Representación de un conjunto de preguntas las cuales serán utilizadas en el laboratorio, contiene las funciones necesarias para extraer y validar cada uno de los elementos que componen una pregunta. Cada pregunta consiste en una lista con múltiples elementos los cuales le dan formato. Los elementos que la componen son:

   (votos respuestas ID etiquetas título contenido fecha autor estado recompensa reportes recompensas_retenidas)
   Ejemplo: (list (list 1 0) (getRespuestas_pregunta 1 listaR (list)) 1 (list "java" "C++" "tipeo" "1") "Titulo 1" "Contenido 1" (list 1 6 2020) "Usuario1" 1 20000 0 (list))

   Para extraer cada elemento de una pregunta, se crean funciones selectoras que se caracterizan por anteponer un “get”, así mismo, se crean funciones de pertenencia para validar que cada elemento de una pregunta tenga el formato correcto. Cabe destacar la existencia de dos funciones de obtención especial llamadas getPreguntas_usuario y getPregunta_ID las cuales tienen un uso específico. getPreguntas_usuario obtiene todas las preguntas que un usuario específico a publicado, getPregunta_ID entrega una pregunta en específico señalada con su ID. Se crea una lista con todas las preguntas emitidas llamada listaP.

3. TDA Usuarios: Representación de un conjunto de usuarios los cuales serán utilizadas en el laboratorio, contiene las funciones necesarias para extraer y validar cada uno de los elementos que componen un usuario. Cada usuario consiste en una lista con múltiples elementos los cuales le dan formato. Los elementos que lo componen son:

   (credencial preguntas respuestas reputación)
   Ejemplo: (list (list "Usuario2" "contrasena2") (getPreguntas_usuario "Usuario2" listaP (list))
   (getRespuestas_usuario "Usuario2" listaR (list)) 20)

   Para extraer cada elemento de un usuario, se crean funciones selectoras que se caracterizan por anteponer un “get”, así mismo, se crean funciones de pertenencia para validar que cada elemento de un usuario tenga el formato correcto. Cabe destacar la existencia de una función de obtención especial llamada getUsuario_usuario la cual tiene un uso específico. getUsuario_usuario obtienen un usuario en específico dentro de una lista de usuarios señalado con su nombre usuario. Se crea una lista con todos los usuarios registrados llamada listaU.

4. TDA stack: Representación de todas las respuestas, preguntas, usuarios y el nombre del usuario con sesión iniciada que serán utilizadas en el laboratorio, contiene las funciones necesarias para extraer y validar cada uno de los elementos que componen un stack. El stack consiste en una lista con múltiples elementos los cuales le dan formato. Los elementos que lo componen son:

   (preguntas respuestas usuarios usuario)
   Ejemplo: (define stack(list listaP listaR listaU ""))

   Para extraer cada elemento de un stack, se crean funciones selectoras que se caracterizan por anteponer un “get”, así mismo, se crean funciones de pertenencia para validar que cada elemento de un stack tenga el formato correcto. Se crea una lista con todas las respuestas, preguntas, usuarios y usuario con sesión iniciada llamada stack.

5. Globales: archivo que contiene funciones que se utilizan en todo el laboratorio. Las funciones creadas son: vacio?(verifica que una lista está vacía), len(entrega la cantidad de elementos que tienen una lista) y reversed(da vuelta una lista).

6. Register: Función que registra a un usuario. Crea un alista nueva en la que incorpora las respuestas y preguntas de un stack y crea una lista nueva de usuarios con el usuario nuevo incorporado con cada elemento correspondiente. Se crea una función extra llamada credencialValida?, la cual verifica que el usuario que se quiere registrar no esté ya registrado.

   Dominio: stack, usuario, contraseña – Recorrido: stack actualizado

7. Login: Función que inicia sesión y retorna la operación ingresada, se le entrega un stack y crea una lista nueva con las respuestas, preguntas y usuarios del stack pero con el usuario con sesión iniciada actualizado. Utiliza la función de pertenencia específica echa en el TDA Usuarios llamada usuarioValido?, la cual verifica que el usuario que quiere iniciar sesión esté registrado y la contraseña sea correcta.

   Dominio: stack, usuario, contraseña, operación – Recorrido: operación con stack actualizado

8. Ask: Función currificada que añade una pregunta al stack. Se crea una lista en la que se añade las respuestas del stack, las preguntas del stack actualizados y los usuarios del stack actualizados para luego ser retornada, en el caso de un cumplir con el login, retorna el stack sin modificar. Para incorporar las preguntas y usuarios actualizados, hace uso de dos funciones que ,utilizando recursión natural ,los actualizan, estas son agregarPregunta_preguntas y agregarPregunta_usuarios.

   Dominio: stack, fecha, titulo, lista de etiquetas y contenido – Recorrido: stack actualizado con sesión cerrada

9. Reward: Función currificada que añade una recompensa retenida a una pregunta. Se crea una lista nueva en la que se añaden las respuestas del stack, las preguntas del stack actualizados y los usuarios del stack actualizados junto a un cierre de sesión, en el caso de no cumplir con el login, retorna el stack sin modificar. Para incorporar las preguntas y usuarios actualizados se hace uso de dos funciones que, usando recursión natural, los actualizan, estas funciones son agregarRec_preguntas y agregarRec_usuarios.

   Dominio: stack, ID y recompensa – Recorrido: stack actualizado con cierre de sesión

10. Answer: Función currificada que permite registrar una respuesta a una pregunta en específico. Se crea una lista nueva en la que se añaden las respuestas actualizadas del stack, las preguntas y usuarios actualizados. Para incorporar las respuestas, preguntas y usuarios actualizados se usan tres funciones que los actualizan, estas son agregarRespuesta_R_a, agregarRespuesta_P_a y agregarRespuesta_U.

    Dominio: stack, fecha, ID y respuesta en string – Recorrido: stack actualizado con cierre de sesión

11. Accept: Función currificada que permite otorgar una recompensa a una respuesta y marcarla como aceptada. Se crea una lista nueva en la que se añaden las respuestas, preguntas y usuarios actualizados. Para poder actualizarlos se hace uso de tres funciones, estas son getRespuestas_acc, getPreguntas_acc y getUsuarios_acc.

    Dominio: stack, ID, correlativo – Recorrido: stack actualizado

VII. Aspectos de implementación

La estructura del código se rige por múltiples archivos con formato ". rkt", estos son: "Globales.rkt", "TDA_Respuestas.rkt", "TDA_Preguntas.rkt", "TDA_Usuarios.rkt", "TDA_Stack.rkt" y "main.rkt". La razón por la que se crearon múltiples archivos fue para facilitar la creación y manejo de funciones para un área específica (y porque se planteó así en el enunciado). Para la creación de los archivos se utilizó la aplicación "Dr. Racket" que utiliza un lenguaje de programación "Scheme", en este se utilizó solo la librería "#lang racket" dado que solo estaba permitido las funciones básicas.

VIII. Instrucciones de uso

1. Register:
    a. Uso correcto:
        i. (register stackOverflow "usuario" "contraseña")
    b. Variables de entrada:
        i. stackOverflow: stack con formato descrito en TDA_Stack
        ii. Usuario: string con el nombre de usuario
        iii. Contraseña: string con la contraseña
    c. Ejemplo:
        i. (register stack "Usuario1" "Contrasena1")

2. Login:
    a. Uso correcto:
        i. (login stackOverflow "usuario" "contraseña" operacion)
    b. Variables de entrada:
        i. stackOverflow: stack con formato descrito en TDA_Stack
        ii. Usuario: string con el nombre de usuario
        iii. Contraseña: string con la contraseña
        iv. Operación: función currificada que señala la acción que se ejecutara una vez iniciado sesión.
    c. Ejemplo:
        i. (login stack "Usuario1" "Contrasena1" ask)

3. Ask:
    a. Uso correcto:
        i. (((ask stackOverflow) fecha) título etiquetas contenido)
    b. Variables de entrada:
        i. stackOverflow: stack con formato descrito en TDA_Stack
        ii. Fecha: lista que contiene la fecha en la que se sube la pregunta
        iii. Titulo: string que contiene el título de la pregunta
        iv. Etiquetas: lista de strings que señalan las etiquetas de la pregunta
        v. Contenido: contenido descriptivo de la pregunta
    c. Ejemplo usando login:
        i. (((login stack "Usuario1" "contrasena1" ask) (list 18 11 2020)) "Mi pregunta 1" (list "etiqueta 3" "etiqueta 2" "etiqueta 1") "contenido 1")

4. Reward:
    a. Uso correcto:
        i. (((reward stackOverflow) ID) recompensa)
    b. Variables de entrada:
        i. stackOverflow: stack con formato descrito en TDA_Stack
        ii. ID: entero que señala la pregunta
        iii. Recompensa: entero que señala la recompensa que se dará a la pregunta
    c. Ejemplo usando login:
        i. (((login stack "Usuario1" "contrasena1" reward) 1) 2)

5. Answer:
    a. Uso correcto:
        i. ((((answer stackOverflow) fecha) ID) respuesta)
    b. Variables de entrada:
        i. stackOverflow: stack con formato descrito en TDA_Stack
        ii. Fecha: lista que contiene la fecha en la que se hace la operación
        iii. ID: entero que señala la pregunta
        iv. Respuesta: respuesta con formato descrito en TDA_Respuestas
    c. Ejemplo usando login:
        i. ((((login stack "Usuario1" "contrasena1" answer) (list 25 10 2020)) 6) "Mi respuesta 1")

6. Accept:
    a. Uso correcto:
        i. (((accept stackOverflow) ID) correlativo)
    b. Variables de entrada:
        i. stackOverflow: stack con formato descrito en TDA_Stack
        ii. ID: entero que señala la pregunta
        iii. Correlativo: entero que señala la respuesta de una pregunta
    c. Ejemplo usando login:
        i. (((login stack "Usuario1" "contrasena1" accept) 1) 2)

IX. Resultados obtenidos

Funciones	Pruebas	Resultados
Register	1. (register stackOverflow "Soy Batman" "zorro no te lo lleves") 2. (register stackOverflow "Pedrito" " Pedrito123") 3. (register stackOverflow "Jose" "xxjosexx1234")	100% 1. Se Registra sin problemas 2. Se Registra sin problemas 3. Se Registra sin problemas
Login	Pruebas echas con las funciones siguientes	-
Ask	1. (((login stack "Usuario1" "contrasena1" ask) (list 18 11 2020)) "Mi pregunta 1" (list "etiqueta 3" "etiqueta 2" "etiqueta 1") "contenido 1") 2. (((login stack "Usuario2" "contrasena999" ask) (list 18 11 2020)) "Mi pregunta 2" (list "etiqueta 3" "etiqueta 2" "etiqueta 1") "contenido 2") 3. (((login stack "Usuario333" "contrasena333" ask) (list 18 11 2020)) "Mi pregunta 333" (list "etiqueta 3" "etiqueta 2" "etiqueta 1") "contenido 333")	100% 1. Pregunta subida sin problemas 2. Pregunta no subida, contraseña incorrecta 3. Pregunta no subida, usuario no registrado
Reward	1. (((login stack "Usuario1" "contrasena1" reward) 1) 2) 2. (((login stack "Usuario2" "contrasena2" reward) 2) 12) 3. (((login stack "Usuario generico" "contrasena generica" reward) 99) 99999)	100% 1. Recompensa subida sin problemas 2. Recompensa no subida, correlativo no existe 3. Recompensa no subida, ID no existe

Funciones	Pruebas	Resultados
Answer	1. ((((login stack "Usuario1" "contrasena1" answer) (list 25 10 2020)) 6) "Mi respuesta 1") 2. ((((login stack "Usuario2" "contrasena2" answer) (list 25 10 2020)) 1) "Mi respuesta 2") 3. ((((login stack "Usuario generica" "contrasena generica" answer) (list 25 10 2020)) 9999) "Mi respuesta generica")	100% 1. Respuesta subida sin problemas 2. Respuesta subida sin problemas 3. Respuesta no subida, pregunta 9999 no existe
Accept	1. (((login stack "Usuario1" "contrasena1" accept) 1) 2) 2. (((login stack "Usuario2" "contrasena2" accept) 5) 999) 3. (((login stack "Usuario generico" "contrasena generica" accept) 99) 2)	100% 1. Recompensa entregada con éxito 2. Recompensa no entregada, reputación insuficiente 3. Recompensa no entregada, usuario no registrado

x. Conclusión

Se puede concluir en este informe que, utilizando lenguaje funcional, y a pesar de sus limitaciones, se puede llegar a realizar algoritmos complejos como los planteados en el laboratorio, aunque cabe destacar que no son muy eficientes en cuando a manejo de memoria dado que se tienen que crear múltiples variables y listas en cada función que se ejecuta.