

**UNIVERSIDAD DE SANTIAGO DE CHILE**

**FACULTAD DE INGENIERÍA DEPARTAMENTO DE INGENIERÍA INFORMÁTICA**

**paradigmas de programación**

**Paradigma lógico en Prolog**

Alumno: Felipe Fernandez Hernandez

Profesor: Daniel Gacitúa Vásquez

Santiago – Chile

01-2021

**Tabla de contenidos**

1.-introduccion

2.- Descripción del problema

3.- Descripción del paradigma

4.- Análisis del problema

5.- Diseño de la solución

6.- Aspectos de la implementación

7.- Instrucciones de uso

8.- Resultados

9.- Conclusiones

10.-Referencia

**1.-INTRODUCCIÓN**

Con Documentos de Google, puedes escribir documentos, editarlos o colaborar en ellos dondequiera que estés y de forma gratuita

Con una edición inteligente y unas herramientas de estilo permiten que sea más fácil dar formato al texto y a los párrafos, entre cientos de fuentes, y agrega vínculos, imágenes y dibujos.

Siendo de fácil acceso, creando , editando y compartiéndolo para trabajar en equipo

En el desarrollo de la asignatura de paradigmas de programación, se inicia con la unidad del paradigma funcional, en donde se solicita un programa en el lenguaje Scheme que se apegue a este paradigma creando una simulación de un Documento de Google (GoogleDoc).

**2.-Descripsion del problema**

En este laboratorio se intentará abordar la implementación de un GoogleDoc con sus funciones más populares (registrarse, ingresar, crear un doc, compartirlo , editarlo) en un lenguaje que cumple con el paradigma funcional el cual es Sheme

**3.- Descripción del paradigma y el leguaje**

**el lenguaje funcional:** Cualquier elemento, situación o proceso podemos representarlo a través de funciones, esto ofrece versatilidad y simpleza, a partir del principio de abstracción en lo que se basa la concepción del mundo a partir de funciones.

La programación funcional es un paradigma de programación, en donde trabajamos a base de funciones

Las funciones que pueden tomar funciones como parámetros y devolver otras funciones como resultado se les denomina funciones de orden superior.

El paradigma se enfoca más en los resultados y de menor manera en la forma en la cual fue realizado el programa, vale decir que expresamos la lógica sin describir controles de flujo, no se utilizan ciclos.

Tiene su enfoque en el cálculo lambda, que se conoce como el primer lenguaje funcional de la historia, este está basado en el concepto matemático de función, en donde el programa consiste en una serie de definiciones, como funciones o tipos de datos, las estructuras de control base son la composición y recursión, no existe la operación de asignación y las variables almacenan definiciones, no cambian pues se debe cumplir lo que se conoce como transparencia referencial.

**Scheme:** es un lenguaje de programación que surgió en los laboratorios del MIT en 1975, cuando Guy L. Steele y Gerarld J. Sussman buscaban un lenguaje con una semántica muy clara y sencilla. Pensaban que los lenguajes no se deberían desarrollar añadiendo muchas características, sino quitando las debilidades y las limitaciones que hacen que las características adicionales parezcan necesarias. Scheme es un dialecto de Lisp, es un lenguaje interpretado, muy expresivo y soporta varios paradigmas. Estuvo influenciado por el cálculo lambda.

**4.- Análisis del problema**

Como se mencionó anteriormente, se usará el lenguaje Sheme, Este paradigma tiene una gran cantidad de aplicaciones y todo es a base funciones. En este laboratorio se seguirá la forma de documentación de la implementar tipos de datos abstractos (TDA), y representaciones para cada implementación de modo que se puedan aplicar a las funciones pedidas, con base en esto se debe crear un TDA de usuario (usuario y contraseña) un TDA de Doc donde contendrá la información de cada Doc creado (fecha de publicación, ID, las modificaciones, quienes tienen acceso y su contenido) ,un TDA principal que contiene las funciones pedidas por enunciado

**5.- Diseño de la solución**

Para la solución de este problema, se implementaron los siguientes TDAs:

* TDA Usuario: es representado por una lista que contiene el nombre de usuario y contraseña
* TDA Doc: que contiene todas los GoogleDoc hechos. Se representa por una lista con el autor (lista nombre de usuario y contraseña), la fecha, las publicaciones, la lista de personas que pueden tener acceso, y su identificador (ID)
* TDA búsqueda: permite a un usuario buscar un doc en especifico en el cual el tenga acceso
* TDA Tiempo: enfocado especialmente en la creación y lectura de fechas
* TDA general: es el que contiene las funciones principales y solicitadas por el enunciado las cuales son:
* register: permite registrar a un nuevo usuario en el GoogleDoc.
* login: permite autenticar a un usuario registrado iniciar sesión y junto con ello permite la ejecución de comandos concretos dentro del doc.
* Create: permite a un usuario con sesión iniciada en la plataforma crear un doc (certificación).
* Share: permite a un usuario a través del ID de un post de alguien replicar el mismo contenido en su cuenta.
* Añadir: permite añadir mas información a un doc creado
* Restorar: permite restaurar la información de un doc que ha sido editado
* Remover acceso: permite eliminar los permisos dejando solamente al creador del doc con la función de poder editar el doc
* Buscar: Función que permite al usuario buscar documentos (propios o que le hayan sido compartidos
* stack->string: recibe un doc y entrega una representación de este como un string posible de visualizar de forma comprensible al usuario.

**6.-Aspectos de la implementación**

el sistema sería una simulación de GoogleDoc representado por otros TDA, los cuales contienen funciones. Los constructores permiten crear estos funciones y definiciones, las funciones de pertenencia sirven para verificar que las definiciones sean verdaderas, los selectores sirven para obtener partes internas de cada función creada, los modificadores permiten agregar, editar o quitar elementos a alguna de las zonas internas, otras operaciones son comandos obligatorios o extras pedidos en el enunciado del proyecto, o bien sirven de apoyo a otras funciones.

• Import: permite obtener otros archivos que se crearon que sean compatibles con el programa y así poder usar en la definición de otro archivo.

• Error: el intérprete detecta un error durante la ejecución del programa y ocurre una excepción

Archivo de texto plano

Se crea un archivo de texto plano llamado “README” que se caracteriza por entregar ejemplos de uso para cada función requerida y observaciones para no cometer errores en la evaluación de las funciones del programa, También un archivo “autoevaluación” donde se señalan los requerimientos funcionales y no funcionales, con una evaluación para cada uno, Además un archivo “repositorio” donde se incluye url de repositorio.

Archivo principal

El TDA es la unidad básica donde están contenidas las funciones a trabajar, se escribe un mensaje para indicar que se hace en cada caso, para señalar representaciones, constructores, funciones de pertenencia o selectores. Cada TDA representa un aspecto del problema por separado y luego se llaman en un TDA principal el cuál trabaja en conjunto con las implementaciones realizadas.

*Librerías Racket*: De esta librería se utilizan todas las funciones que son útiles para la implementación del problema.

*Otras*: Para el caso de funciones no pertenecientes a la librería señala, lo que se hace es definir la función en el archivo correspondiente a su utilización por lo que no se requieren otras librerías.

**7.- Instrucciones de uso**

Primero se debe compilar el código, para ello se debe abrir la carpeta contenedora llamada “**lab1\_20575068\_fernandez\_carpetaDeArchivos**” en el cual se encuentra contenido en el archivo del código fuente llamado **“lab\_git\_20575068\_FernandezHernandez.rkt”.** al accionarlo Se debe abrir el intérprete Dr. Racket y en la esquina superior derecha se ubica un icono tipo flecha que permitirá abrir la consola.

Ahora es posible compilar y evaluar las funciones requeridas por enunciado a través de la consola de Dr. Racket. Si no se abren los archivos correctamente y no son compilados no se podrá interactuar con la consola. Primero se deben abrir y compilar, para luego trabajar en la consola con las funciones creadas y mencionadas anteriormente.

**8.- Resultados**

En la siguiente tabla se mostrará la completitud de cada predicado:

|  |  |
| --- | --- |
| Predicado | Completo |
| Registre | 100% |
| Login | 100% |
| create | 100% |
| share | 100% |
| add | 100% |
| restoreVersion | 100% |
| revokeAllAccesses: | 100% |
| search | 100% |
| paradigmadocs->string | 100% |

Todas las funcionalidades implementadas pasaron distintas pruebas:

• Crear un usuario no registrado

• Crear un usuario registrado

• Aplicar múltiples veces con el fin de ver fallas en la implementación

• Intentar ingresar parámetros no validos

• Analizando que todo aquello que mostrara fuera valido

**9.- Conclusiones**

Se puedo observar que en el avance del proyecto se logró con éxito implementar que cada una de la función requerida por el enunciado del laboratorio cumpliera su objetivo

El desarrollo de este problema es un excelente ejercicio para finalizar el entendimiento de la programación funcional, si bien tiene su complejidad sobre todo cuando uno viene de un paradigma imperativo , mediante la práctica y las múltiples funciones que vas creando se vuelve más sencilla la creación de futuras funciones y el saber cómo aplicarla

**10.-referencias**

Lara, K. O. (2014). Lenguajes de programación. Portafolio Digital.

García, P. E. I. (2019). Qué es la Programación Funcional. Artículo blog Códigofacilito.

Martines.M(2020). ¿que son los paradigma de programación?. profile.es

Google (s.f). Crea documentos impactantes. docs.google.com