FACULTAD DE
INGENIERÍA
UNIVERSIDAD DE SANTIAGO DE CHILE

Paradigmas de Programación
Laboratorio 1: Simulación de sistema de archivos en Scheme.

Nombre: Isidora Oyanedel

Profesor: Gonzalo Martínez

22 de mayo 2023

1. INTRODUCCIÓN.....3
1.1 DESCRIPCIÓN DEL PROBLEMA..... 3
1.2 DESCRIPCIÓN DEL PARADIGMA..... 4
1.3 OBJETIVOS..... 4
2. DESARROLLO.....5
2.1 ANÁLISIS DEL PROBLEMA..... 5
2.2 DISEÑO DE SOLUCIÓN..... 6
2.3 ASPECTOS DE IMPLEMENTACIÓN..... 7
2.4 INSTRUCCIONES DE USO.....7
2.5 RESULTADOS Y AUTOEVALUACIÓN..... 7
3. CONCLUSIÓN.....8
4.BIBLIOGRAFÍA..... 8
5.ANEXO..... 9

1. INTRODUCCIÓN

El presente informe corresponde al laboratorio n°1 de Paradigmas de programación, donde nos centramos en el paradigma funcional y utilizamos el lenguaje de programación Scheme a través del compilador Dr. Racket, a través de estas herramientas debemos llegar a la resolución del problema planteado, el cual es la creación de un simulador de sistema operativo, centrado en un sistema de archivos incluyendo comandos que permitan operar en él.

1.1 DESCRIPCIÓN DEL PROBLEMA

Como se mencionó, se pide desarrollar un simulador de sistema operativo en el cual se puedan crear drives, usuarios y poder crear carpetas y archivos dentro de los drives, los cuales tendrán sus respectivos permisos, especificaciones, entre otras funciones. Para lo cual se necesita crear ciertos elementos que sean capaces de hacerlo, como, por ejemplo:

Sistema: El elemento principal para iniciar la solución del problema, al tenerlo creado se puede empezar a crear los drives, usuarios, carpetas, rutas, etc. Pues en él serán almacenadas estas unidades.

Drive: Creaciones del usuario, en donde el mismo puede seleccionar alguno para poder trabajar en él y aplicarle cambios. Además se pueden realizar acciones entre drives a pesar de que no sea el seleccionado para trabajar.

Usuario: Son los usuarios registrados en el sistema, los cuales se identifican por un nombre de usuario. Se le atribuyen permisos, con los cuales, mientras esté logueado, será capaz de utilizar las funciones del sistema.

Archivos/carpetas: Creaciones del usuario, en cada creación se registra un autor, la fecha de creación, su nombre, su contenido, entre otros aspectos principales en el que cada cambio quedará registrado.

Otras operaciones: Funciones encargadas de loguear, desloguear, crear, modificar, en el sistema. En resumen, hacen el trabajo necesario.

Cada uno de los elementos especificados debe ser implementado en el lenguaje Scheme y el compilador Dr. Racket, centrando su implementación en el paradigma funcional.

1.2 DESCRIPCIÓN DEL PARADIGMA

En este laboratorio nos centramos en el uso del paradigma funcional, siendo paradigma una especie de patrón y paradigma funcional un estilo de programación que se enfoca en el uso de funciones y trata de que no hayan cambios de estados, y debido a esto no permite las variables actualizables.

En el paradigma funcional existen ciertos conceptos que son claves al momento de trabajar con él:

Funciones anónimas: Proviene del cálculo lambda y corresponde a una función sin nombre asociado a argumentos de entrada y un cuerpo, la cual se declara para que entregue un resultado

Currificación: Transforma la evaluación de una función de múltiples argumentos en una secuencia de funciones de un argumento, usado para simplificar la composición de funciones.

Función de orden superior: Función que puede operar sobre funciones e incluso puede devolver funciones como resultado

Recursión: En esta forma de programación resulta ser una herramienta bastante útil para la resolución de problemas mediante soluciones más pequeñas del mismo problema. Para este paradigma existen 3 tipos de recursiones: recursión natural, recursión arbórea y recursión de cola, siendo la última la más eficiente.

Composición de funciones: Operación en donde dos funciones unidas producen una tercera, usualmente usado para crear funciones más complejas.

1.3 OBJETIVOS

En este proyecto se aprenderá el cómo funciona el paradigma funcional y la programación funcional, también algunos aspectos nuevos de la programación en el lenguaje Scheme y sobre él como la entendíamos antes del paradigma. Aprender a programar en lenguaje Scheme y entender las herramientas de Dr.Racket a través del desarrollo del laboratorio.

2. DESARROLLO

2.1 ANÁLISIS DEL PROBLEMA

Se desarrolla la simulación de un sistema de archivos a través de la consola de Dr. Racket, dando a entender que ninguno de los datos que se trabajan son reales y es una simulación simple que no busca ser 100% fiel a uno real y funcional. Partiendo por el elemento principal, el sistema que es a partir del cual se pueden desarrollar el resto de funciones que hacen funcionar la simulación, y guarda todos los datos ingresados, como drives, usuarios, archivos, fechas, etc. Se necesita representar el sistema en scheme:

Para poder acceder a las carpetas ya sea para agregar contenido o verlo, se necesita que haya un usuario con sesión iniciada.
Las funciones que se pueden aplicar en el sistema son:

- add-drive: verifica si hay un drive con la misma letra del que se quiere crear, si no es así, se crea un nuevo drive.
- register: verifica si hay un usuario con el mismo “username”, si no es así se crea un usuario nuevo.
- login: loguea un usuario si es que está registrado su username.
- logout: desloguea al usuario del sistema.
- switch-drive: fija la unidad de drive en la que el usuario trabajará.
- md (make directory): crea un directorio dentro de una unidad.
- cd (change directory): cambia la ruta donde se realizarán operaciones.
- add-file: agregar un archivo en la carpeta en la ruta.
- del: elimina un elemento o varios, puede ser un archivo o una carpeta.
- rd (remove directory): elimina una carpeta si es que esta está vacía.
- copy: copia una carpeta o archivo a una ruta de destino desde una ruta de origen.
- move: mueve un archivo o una carpeta.
- ren (rename): renombra una carpeta o un archivo.
- dir (directorio): muestra el contenido que hay en una registro o una ruta entera.

- format: formatea el contenido de un drive.
- encrypt: encripta un archivo o una carpeta.
- decrypt: desencripta un archivo o una carpeta.
- plus-one: función que a un string lo transforma sumando un 1 a su código ASCII a cada carácter.
- minus-one: función que a string lo transforma restando un 1 a su código ASCII a cada carácter.
- view-trash: obtiene el contenido que hay en la papelera de reciclaje.
- restore: recupera el contenido que se encuentra en la papelera.

Desde la función md se necesita una sesión iniciada para funcionar correctamente. Se han separado estas funciones de acuerdo a su TDA (tipo de dato abstracto) correspondiente, pues cada TDA se encarga de un aspecto en específico del sistema.

2.2 DISEÑO DE SOLUCIÓN

Se ha necesitado la creación de TDA para poder realizar las solicitudes del laboratorio pues las herramientas de Scheme por sí solas no eran suficientes para cumplir las exigencias.

- TDA SYSTEM: lista de listas que contiene el nombre del archivo, fecha, usuarios, drives, etc.
- TDA DRIVE: Encargado de crear y cambiar de drive según lo solicite el sistema.
- TDA USER : Lista que contiene a los usuarios correctamente creados.
- TDA FOLDER lista de listas que contiene los archivos, la fecha de modificación y la ruta actual.
- TDA FILE: lista que contiene el archivo, su extensión y su contenido.

La solución optada se centró en la manipulación del sistema, el cual incluye todos los tipos de datos abstractos (TDA) creados. Esta estrategia permitió un acceso y modificación más eficiente de los TDA implicados en el proceso.

Las funciones obligatorias del sistema se encuentran en la tabla 1 del anexo, donde se especifica el método de solución en el código.

Al momentos de ir realizando el código hay detalles que deben ser tomados en cuenta:

- **Función cons:** Al momento de comenzar a realizar este proyecto, se empezó a emplear la función cons, la cual, como su nombre indica es un constructor, proviene de la palabra en inglés “Construct”, esta función tiene como principal uso, construir listas y gracias a esto, se pudieron realizar varias operaciones sobre el sistema, esta técnica se utilizó en varias funciones más ya que los datos se fueron apilando a través de la función cons, con esta técnica se pueden ver los todos los cambios que han ocurrido a lo largo del código y sirve para ir comprobando si está correcto o no el resultado.
- **Funciones map y filter:** Al momento de ir avanzando en el código, se llega a un punto en el que agregar elemento no basta para solucionar los problemas que van naciendo, así que desde este punto, nace la idea de usar las funciones filter y map, estas funciones fueron de gran utilidad ya que permitieron obtener y aplicar operaciones sobre el sistema, es decir, al

usar filter, permite buscar o aplicar cambios sobre una carpeta o archivo específico, en cambio map, permite aplicar funciones sobre un elemento, esto igual fue de gran utilidad al momento de querer hacer operaciones sobre una lista con varios elementos.

2.3 ASPECTOS DE IMPLEMENTACIÓN

En este laboratorio se usó el compilador de DrRacket versión 8.8, por otro lado, no se usan bibliotecas externas ni la función set!, ya que se respeta que la simulación se tiene que realizar a través del Paradigma Funcional.

La estructura del proyecto para que tenga coherencia va con respecto a los TDA creados, con esto, quiero decir, que la mejor forma de visualizar la forma en cómo se fue construyendo el código es la siguiente: Primero se partió con el TDA Sistema, para luego crear el TDA Drive, luego el TDA Usuario, el siguiente fue TDA carpeta y por último se creó el TDA archivo. Todas las funciones de estos se pueden ver en una tabla en el anexo en orden.

2.4 INSTRUCCIONES DE USO

Se necesita que se tengan descargados y juntos todos los archivos del proyecto, caso contrario el programa probablemente tirará error por falta de un TDA.
Si se quiere probar usando los ejemplos de prueba, solo se necesita apretar run, pues tendrá un run sin comentar, y para probar las demás funciones solo se necesita descomentarlas.
En caso contrario si se desea crear pruebas personalizadas se necesita seguir la estructura de cada script.

1. Primero que nada, se necesita crear un system para que las funciones hagan algo, con el script (define S0 (system "nombre")) donde “nombre” puede ser cualquier palabra siempre y cuando se mantenga entre comillas.
2. Con eso hecho se puede usar cualquier función del programa hasta la función md pues este necesita de un usuario conectado. Para conectar un usuario se debe hacer a través de la función Login junto con un previo register del usuario que se desee conectar.
3. Con un usuario registrado ya se puede usar el resto de funciones del código y ya según la operación escogida hay que prestar atención a los parámetros que se entregan pues no en todas es igual. Haciendo eso debería poder usar todas las funciones del main.

Revisar figuras 2, 3 y 4 para una demostración gráfica del funcionamiento.

2.5 RESULTADOS Y AUTOEVALUACIÓN

RESULTADOS ESPERADOS: Se espera que funcionen la mayoría de las funciones de acuerdo a los scripts entregados por los profesores, pudiendo hacer la simulación correctamente hasta la función que se pudo llegar, dando resultados correctos, excepto en dos casos puntuales que son Move (en caso de mover archivos) y dir (problema con la entrada vacía). Funciona correctamente con los scripts de prueba entregados por los profesores.

AUTOEVALUACIÓN:

0: No realizado – 0.25: Funciona 25% de las veces – 0.5: Funciona 50% de las veces 0.75: Funciona 75% de las veces – 1: Funciona 100% de las veces.

Ver tabla n° 7 y 8 de autoevaluación en el anexo.

La función dir tiene 0.75 porque no realiza el caso de entrada vacía pero en el caso de carpetas ocultas y archivos lo hace bien y el mismo puntaje para move, que como fue mencionado, no mueve correctamente los archivos en ciertas condiciones. El resto de las funciones hechas no dan error al correrlas.

3. CONCLUSIÓN

Al haber realizado la mayoría de la actividad, se tuvo como resultado el haber podido completar los objetivos planteados al inicio, pues se pudo crear la mayoría de las funciones obligatorias solicitadas en el proyecto y con el tiempo se pudo lograr manejar y entender Scheme entre otras herramientas mostradas en clase como TDA´s, recursividad, funciones de orden superior, semi-currificación , etc.

Lo más complicado del laboratorio fueron los momentos en los que uno tenía una idea y al momento de implementarla puede que funcionara pero cuando avanzabas en el resto de funciones, uno se daba cuenta que eso que funcionaba en la función ‘x’, hacia que la función’ y’ no funcionara correctamente, por lo cual había que devolverse a la función’ x’ a pensar una nueva forma de solución al problema. Otro problema fue que al pedir ayuda a los profesores se demoraban mucho en responder o su respuesta era muy compleja de entender en escrito, lo cual al final no ayudaba mucho. Para concluir debo decir que a mi se me complicó demasiado entender algunas funciones (como md o cd), las cuales en mi proyecto hacen algo pero no sé si sean tan eficaces al momento de ejecutarse.

Fuera de eso, se pudo aprender a usar git y variedad de funciones en scheme. Además espero que este laboratorio haya servido como experiencia para poder llevar de mejor manera el siguiente laboratorio.

4.BIBLIOGRAFÍA

Gonzales. R (2023) “proyecto semestral de laboratorio” Obtenido de Universidad Santiago de Chile. Paradigma de Programación. Enunciado de proyecto online. https://docs.google.com/document/d/1x_w9ydqj9nAnn0v0LeXXBMoWgMRE6HEA5qLfS717f9c/edit

Gonzales. R (2023) “3 - P. Funcional” Obtenido de Universidad Santiago de Chile. Paradigma de Programación. Material de clases online ”https://uvirtual.usach.cl/moodle/course/view.php?id=10036&section=11

5.ANEXO

Tabla 1. descripción de solución de las funciones.

Tipo de función	Nombre	Descripción
Modificador	add-drive	Ingresa la letra entregada a la función y si la letra no está en uso la agrega a la lista de unidades del sistema, en caso contrario el sistema se mantiene sin cambios.
Modificador	register	primero verifica si el nombre que recibe de entrada corresponde a un string, en caso de ser así, verifica si ese nombre de usuario existe en los usuarios del sistema, si no existe la función agrega un nuevo usuario a la lista de usuarios del sistema y devuelve el sistema actualizado con el nuevo usuario. En caso contrario, el sistema se mantiene sin cambios.
Modificador	Login	Verifica si hay un usuario conectado al sistema, si lo hay el sistema se mantiene sin cambios. En caso contrario, verifica si el usuario existe en la lista de usuarios del sistema. Si existe se agrega el usuario a la lista de usuarios conectados y retorna el sistema actualizado con el usuario conectado.
Modificador	Logout	Si hay un usuario en la lista de usuarios conectados al sistema, se borra el usuario de la lista, en caso contrario, mantiene el sistema sin cambios.
Modificador	switch-drive	Se verifica que la letra entregada sea un carácter válido, en caso de ser así se verifica si la letra ingresada existe en las unidades del sistema y se fija la unidad solo si no había ninguna letra fijada antes.
Modificador	md (make directory)	devuelve una nueva versión del sistema actualizado con la nueva carpeta creada si se cumple la condición de que el nombre sea un string y no esté ya en la lista de carpetas existentes en la ruta actual. Si la condición no se cumple, devuelve la misma versión del sistema sin cambios.
Modificador	cd (change directory)	verifica si la ruta es un string. Si es un string, se comprueba si el directorio especificado en la ruta existe en la ruta actual y si no se encuentra en la ruta actual. Si se encuentra el directorio, la función cambia al directorio especificado y devuelve el nuevo sistema con la ruta actualizada y los archivos correspondientes. Si la ruta es `..`, la función cambia al directorio anterior, si la ruta es `/`, la función cambia al directorio raíz. Si la ruta es una ruta absoluta que no está en la ruta actual, la función cambia al directorio especificado en la ruta y devuelve el nuevo sistema con la ruta actualizada y los archivos correspondientes. Caso contrario no realiza cambios.
Modificador	add-file	Si file es un string, agrega un archivo a la lista de archivos de la carpeta actual y devuelve el sistema actualizado. Caso contrario devuelve el sistema sin cambios. El nuevo archivo se agrega con el dueño de la carpeta actual y se guarda en la lista de archivos de la carpeta actual.
Modificador	del	Primero verifica si el nombre del archivo se corresponde con los tipos de archivos ".txt" o ".docx". Si es así, elimina todos los archivos que tienen el mismo nombre. Si el nombre de archivo no se corresponde con ninguno de los tipos de archivo anteriores, la función verifica si hay algún archivo en la carpeta actual que tenga la misma letra inicial que el nombre del archivo. Si es así, elimina todos los archivos que tienen esa letra inicial.Si el nombre de archivo no coincide con ninguna de las condiciones anteriores, la función verifica si hay un archivo en la carpeta actual con el mismo nombre. Si es así, lo elimina. Si el nombre de archivo no coincide con ninguna de las condiciones anteriores, la función verifica si existe una carpeta en la ruta actual con el mismo nombre. Si es así, elimina la carpeta y devuelve una nueva ruta sin esa carpeta.Si no coincide con ninguna condición entrega el sistema sin cambios.

Modificador	rd (remove directory)	Se verifica si foldername es un string y existe en la ruta actual del sistema. Si se cumplen ambos, se verifica si la última actualización archivos de la carpeta está vacía, si es así la carpeta se elimina y se entrega una nueva versión del sistema. Si los archivos no están vacíos se entrega el sistema original.
Modificador	copy	La función comprueba si la fuente y el destino son strings, y luego comprueba si la última posición del destino es una carpeta existente y si el archivo de origen existe en la lista de archivos del sistema. Si ambos son ciertos, se crea una nueva carpeta en el destino y se agrega el archivo de origen a la lista de archivos de esa carpeta en el sistema. Si solo la carpeta es miembro de la ruta y el drive existe en el sistema, entonces se agrega la carpeta de origen y sus archivos a la lista de rutas en el sistema. Si ninguna se cumple, se devuelve el sistema sin cambios. La función asegura que los archivos originales no sean eliminados.
Modificador	move	La función verifica si tanto source como target son cadenas de texto, y si la carpeta de origen source es un miembro de la lista de rutas del sistema de archivos actual. Si ambos son verdaderos, la función intenta mover la carpeta source al directorio de destino target. Sí target es una letra de unidad, se mueve la carpeta a la raíz de esa unidad. Sí target es una ruta de acceso, se mueve la carpeta a esa ubicación en el sistema de archivos. Si source es una carpeta y target es una letra de unidad, la función verifica si existe la carpeta y la letra de unidad en el sistema de archivos. Si es así, la función crea una nueva estructura de sistema de archivos con la carpeta source movida a la raíz de la letra de unidad de destino target. Sí target es una ruta de acceso, la función verifica si el último elemento en la ruta de destino es una carpeta existente en el sistema de archivos, y si source es una carpeta existente en el sistema de archivos. Si ambos son verdaderos, la función crea una nueva estructura de sistema de archivos con la carpeta source movida a la ubicación target en el sistema de archivos. En caso contrario no realiza cambios.
Modificador	ren	Primero, verifica si "name" y "new-name" son cadenas de texto. Luego, verifica si el archivo o carpeta existe en el sistema de archivos y si el nuevo nombre no está siendo utilizado por otro archivo o carpeta. Si se está renombrando un archivo de texto y se cumplen las condiciones anteriores, la función crea un nuevo sistema de archivos que contiene una copia de la carpeta que contiene el archivo de texto renombrado con su nuevo nombre. Si se está renombrando una carpeta y se cumplen las condiciones anteriores, la función crea un nuevo sistema de archivos que contiene una copia de la carpeta renombrada con su nuevo nombre. En caso contrario no ocurren cambios.
Constructor	dir (directorio)	La función imprime un string formateado el cual mostrará un listado del contenido actual de un directorio dependiendo de la entrada que se le entregue
modificador	format	A través de la función set-capacidad, en el cual a través de filtros se recupera el drive y capacidad previo a eliminar, se le cambia el nombre al drive, el filtro permite que no se pierda la capacidad que tenía el drive. formatea un drive borrando todo su contenido y cambiando su nombre, conservando su capacidad

Modificador	encrypt	Se verifica si password es un string, en caso de que path sea un carpeta se verifica si esa carpeta existe en el sistema y al momento de agregar sus archivos, se encriptan con la función encriptar, encriptando su contenido y título de cada archivo que tenga. En caso de que path sea un archivo, se busca si el archivo existe en el sistema, de ser así se aplicará la función encriptar-t a los archivos pero la función solo encripta el archivo con el mismo nombre de path. En caso contrario, se entrega el sistema sin cambios.
Modificador	decrypt	Se verifica que la entrada sea un string, si es asi, pasa al siguiente condicional (caso carpeta) donde pregunta si password coincide con el password del sistema y si es miembro el folder de entrada (en forma de plus-one porque fue el último cambio que tuvo la carpeta) si es así, se le aplica minus-one a los archivos y la carpeta del sistema con la password correcta. En el caso de archivo, se pregunta si existe un archivo como path aplicado plus-one y que tenga la misma contraseña de encriptación, en caso de ser asi los archivos pasan por la función desencriptar-t que busca el archivo encriptado y lo regresa a su estado original. En ambos casos, luego de desencriptar, borra la contraseña de la carpeta. En caso contrario se regresa el sistema sin cambios.
Modificador	plus-one	Recibe de entrada un string que suele ser o el nombre de una carpeta o el nombre de un archivo, al meterlo en una función recursiva, se encarga de aumentar en 1 el número ASCII del string (archivo o carpeta y archivos depende del caso)
Modificador	minus-one	Recibe de entrada un string que puede ser el nombre de un archivo o el de una carpeta, al combinarlo con una función recursiva, se encarga de disminuir en 1 el número ASCII de cada string (archivo o carpeta y archivos según el caso)
[Las funciones que no han sido mencionadas en esta tabla es porque no se han realizado.]		

Figura 1.

DIAGRAMA DEPENDENCIA DE TDA.

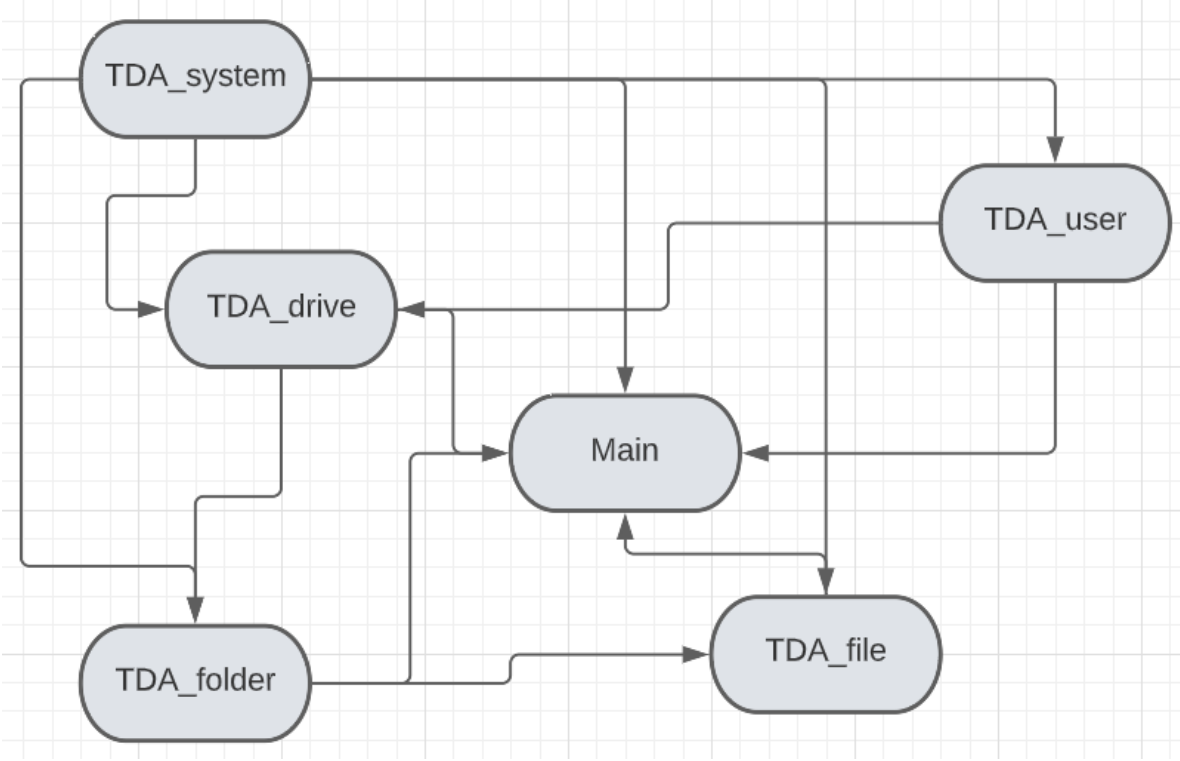

Tabla 2: funciones de los TDA´s de sistema.

TDA sistema:

Tipo	Nombre	Descripción
Constructora	make-system	Crea la estructura sistema
Constructora	make-carpeta	Crea la estructura carpeta
Pertenencia	is-string	Comprueba si la entrada es un string
Selectora	get-system-name	Obtiene el nombre del sistema
Selectora	get-system-ruta	obtiene la/s ruta/s del sistema
Selectora	get-posicion	obtiene la ruta actual del sistema

Tabla 3: funciones de los TDA´s de drive

TDA drive:

Tipo	Nombre	Descripción
Constructora	make-drive	Crea la estructura drive
Pertenencia	is-char	Comprueba si la entrada es un char
Modificadora	set-capacidad	Crea un nuevo drive con la

		capacidad del drive anterior que tenia el mismo char
Selectora	get-system-drive	Obtiene los drives del sistema
Selectora	get-system-drive-seleccionado	Obtiene el drive
Selectora	remove1	Obtiene las rutas que no calzan con el drive de la letra ingresada, esto se realiza en la posición en que se encuentrala ruta del sistema.
Selectora	no-remove	Obtiene el drive que tiene la misma letra que la ingresada esto se realiza en la posición en que se encuentran todos los drives del sistema. Con el objetivo de obtener la capacidad del drive que se va a cambiar.
Selectora	remove2	Obtiene los drives que no tienen la misma letra que la ingresada esto se realiza en la posición en que se encuentran todos los drives del sistema.

Tabla 4: funciones de los TDA´s de user.

TDA user:

Tipo	Nombre	Descripción
Constructor	make-user	Crea la estructura usuario.
Constructor	is-status-user	Crea un usuario en la lista de usuario conectado si es que el usuario existe en la lista de usuarios registrados
Pertenencia	is-member	Comprueba si el usuario es miembro de la lista de usuarios registrados.
Selectora	get-system-usuarios	Obtiene los usuarios del sistema.
Selectora	get-system-usuario-conectado	Obtiene el usuario que está conectado en el sistema.

Tabla 5: funciones de los TDA´s de carpeta.

TDA carpeta:

Tipo	Nombre	Descripción
Otras funciones	recursion-system	Crea un string de las carpetas del sistema en el drive

		seleccionado.
Otras funciones	recursion-system-archivos2	Crea un string de los archivos de su carpeta correspondiente en el sistema y drive seleccionado.
Modificadora	set-encriptar	Encripta todo los archivos contenidos en una carpeta.
Modificadora	set-desencriptar	Encripta el archivo solicitado en la entrada de la función y en la última carpeta seleccionada.
Selectora	get-carpetas	Obtiene las carpetas del sistema.
Selectora	get-password	Obtiene la contraseña de la carpeta encriptada.
Selectora	get-recuperar-ruta	Obtiene la última ruta del drive seleccionado.
Selectora	get-remove-posicion	Remueve una careta de la ruta del sistema.
Selectora	get-rec-archivos	Recupera la última actualización en los archivos de alguna carpeta.
Selectora	get-remove-carpeta-drive	Remueve una carpeta de un drive.
Selectora	get-remove-nulos	Remueve los nulos de la lista de carpetas del drive actual.
Selectora	get-no-recuperar-listas	Recupera los cambios que se produjeron en los drive que no son el actual.
Selectora	get-recuperar-listas	Recupera todos los cambios del drive actual.

Tabla 6: Funciones de los TDA´s de archivo.

TDA archivo:

Tipo	Nombre	Descripción
Constructor	make-file	Crea el archivo file
Constructor	file	Crea la entrada de la función add-file en conjunto con make-file.
Modificadora	set-cambia-nombre-archivo	Hace todo el proceso de cambiar el nombre a un archivo, le agrega el nombre nuevo y rearma la lista de archivos con el archivo que fue modificado.
Modificadora	set-eliminar_archivo	Elimina el archivo de un drive que ha sido movido a otro drive.
Modificadora	set-encriptar-t	Encripta un archivo específico

		de una carpeta con contraseña correcta.
Modificadora	set-desencriptar-t	Desencriptar un archivo específico de una carpeta con contraseña correcta.
Selectora	get-files	Obtiene los archivos del sistema.
Selectora	get-remove-titulo2	Obtiene todos los archivos que no tengan el mismo título que el ingresado.
Selectora	get-no-remove-archivo2	Obtiene el archivo que tenga el mismo nombre que el título ingresado.
Selectora	get-remove-extension	Filtra los elementos que tengan la misma extensión que la extensión ingresada.
Selectora	get-remove-titulo	Remueve los archivos que tengan el mismo título que sea ingresado.
Selectora	get-letter-titles	Remueve los elementos que tengan como letra inicial la letra que se recibe de entrada.
Selectora	get-no-remove-archivo	Obtiene las propiedades del archivo de entrada en su estado más actual para así poder moverlo a otra ubicación.

Tabla 7: Auto Evaluación de requerimientos funcionales.

Nombre	Puntaje
LENGUAJE	1
VERSIÓN	1
STANDART	1
NO VARIABLES	1
DOCUMENTACIÓN	1
DOM -> REC	1
ORGANIZACIÓN	1
HISTORIAL	1
SCRIPT DE PRUEBAS	1

PRE REQUISITOS	1

Tabla 8: Auto Evaluación de requerimientos NO funcionales.

Funcion	Puntaje
SYSTEM	1
RUN	1
ADD-DRIVE	1
REGISTER	1
LOGIN	1
LOGOUT	1
SWITCH-DRIVE	1
MD	1
CD	1
ADD-FILE	1
DEL	1
RD	1
COPY	1
MOVE	0.75
REN	1
DIR	0.75
FORMAT	1
ENCRYPT	1
DECRYPT	1
MINUS-ONE	1
PLUS-ONE	1
GREP	0

VIEW-TRASH	0
RESTORE	0

Figura 2. Creación del sistema.

```
> S0
'("newSystem" () () () () () 1684713993)
```

Figura 3. Sistema con Drives, Usuarios y Usuario conectado.

```
> S10
'("newSystem" ((#\D "Util" 2000) (#\C "SO" 1000)) (("user2") ("user1")) ("user2") () () 1684713993)
```

Figura 4. Creación de ruta y carpetas en el sistema.

```
> S16
'("newSystem"
  ((#\D "Util" 2000) (#\C "SO" 1000))
  (("user2") ("user1"))
  ("user2")
  (#\C)
  (("c:/" "folder3" () ("user2") () 1684713993)
   ("c:/" "folder2" () ("user2") () 1684713993)
   ("c:/" "folder1" () ("user2") () 1684713993)
   ("c:/" () () ("user2") () 1684713993))
  1684713993)
```