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

PARADIGMAS DE PROGRAMACIÓN

PROYECTO 2 – PROGRAMACIÓN FUNCIONAL.

Diseño del sistema computacional para un centro clínico.

Autor:	**FELIPE JARA R.**
Profesor:	Víctor Flores.
Ayudante:	Rodrigo Orellana.
Fecha de Entrega:	17-10-2014

Santiago de Chile

2 – 2014

TABLA DE CONTENIDOS

TABLA DE FIGURAS

CAPÍTULO 1: INTRODUCCIÓN.

1.1 ANTECEDENTES Y MOTIVACIÓN.

Un nuevo centro clínico, de nombre FastClinic, ha sido abierto en el centro de Santiago y requiere de un sistema computacional para poder administrar las fichas de los pacientes, las cuales actualmente se encuentran registradas en una base datos heredada de un antiguo sistema informático para el cual ya no se cuenta con soporte. Es por esta razón que el Departamento de Ingeniería en Informática ha solicitado a sus estudiantes que, como parte de un proyecto, puedan desarrollar una solución para la problemática presentada.

1.2 DESCRIPCIÓN DEL PROBLEMA.

A raíz de lo mencionado con anterioridad, se desprende que el problema a resolver esta vinculado con la ausencia de un sistema informático que permita a los operarios del centro clínico FastClinic poder realizar tareas de administración relacionadas con la información registrada de los pacientes que han sido atendidos en dicho centro de salud. En otras palabras, un sistema que permita a sus usuarios realizar consultas a la base de datos y modificar la información contenida en ella.

1.3 SOLUCIÓN PROPUESTA.

Se propone entonces, desarrollar un nuevo sistema informático implementando las funcionalidades que la clínica necesita y haciendo uso de los datos con los que ellos cuentan y proporcionan. Un software computacional escrito en un lenguaje de programación interpretado, Scheme, que permita al usuario realizar operaciones de manipulación y consulta a través de la ejecución de comandos desde la ventana de interacción proporcionada por el entorno de programación DrRacket.

1.4 OBJETIVOS Y ALCANCES DEL PROYECTO.

1.4.1 Objetivo general.

El objetivo general del proyecto es permitirle a un usuario poder llevar a cabo tareas de administración al facilitarle funcionalidades especificas que permitan obtener, modificar o eliminar cualquier tipo de información que se encuentra registrada en las diferentes bases de datos proporcionadas por el centro clínico FastClinic.

1.4.2 Objetivos específicos.

A continuación, se presentan los objetivos específicos del proyecto:

- Proporcionar un sistema informático con soporte para las bases de datos heredadas.
- Consultar y modifica registros contenidos en las bases de datos heredadas.
- Facilitar formas de verificar si un dato corresponde a cierto tipo de dato o no.

1.4.3 Alcances.

El resultado final que se pretende entregar a los operarios del centro clínico consiste en una aplicación de consola, caracterizada por no contar con una interfaz grafica, cuyo funcionamiento esta garantizado para las maquinas que utilizan el sistema operativo GNU/Linux. Además de su respectiva documentación y manual de usuario.

Cabe mencionar que el programa debe ser capaz de realizar las siguientes funcionalidades:

a) Obtener el nombre de un paciente dado su Rut.
b) Obtener la especialidad de un médico a partir de su Rut.
c) Listar el (o los) tratamiento(s) de acuerdo a un nivel de riesgo dado.
d) Determinar el (o los) tratamiento(s) más usado(s) para un diagnostico particular, indicando la cantidad de veces que se ha empleado.
e) Identificar el (o los) médico(s) que más altas otorga, indicando su rut, nombre y apellido y la cantidad de altas.
f) Identificar el (o los) tratamiento(s) más usado(s) en todo el sistema indicando la cantidad de veces que se ha(n) usado.
g) Dado el identificador de un paciente, indicar el rut, nombre y apellido de todos los médicos que lo han tratado (sin incluir a los que lo dieron de alta).
h) Dado el nombre y apellido de una persona, indicar cuál es el nivel de riesgo del último tratamiento recibido.
i) Conocer todos los pacientes diagnosticados con el diagnóstico X (nombre del diagnóstico) a los cuales el doctor Y (rut) les dio el Alta. Mostrando el rut, nombre y apellido de esos pacientes.
j) Dado el rut de un paciente, indicar el rut, nombre y apellido de todos los médicos que lo han tratado (excluyendo a los que lo han dado de alta).
k) Modificar el correo electrónico de un paciente.

l) Dado el nombre y apellido de un paciente, además del id de un tratamiento, modificar el resultado de su tratamiento más reciente.
m) Dado el correo del paciente y del médico, modificar el médico que da el alta en su última ocasión.

Además, el sistema debe ser capaz de verificar si un valor pertenece a un tipo de dato en particular, es decir, si un registro corresponde a un paciente, doctor, diagnostico, tratamiento, vinculación diagnostico paciente, vinculación tratamiento diagnostico o a una vinculación tratamiento diagnostico paciente.

1.5 METODOLOGÍAS Y HERRAMIENTAS UTILIZADAS.

Debido a que realizar consultas a una base de datos y modificar los registros contenidos en ella son tareas que pueden involucrar el seguimiento de una variedad de instrucciones cuya complejidad varia dependiendo de lo que se esta solicitando, se estima conveniente realizar el diseño de la aplicación teniendo como base lo propuesto por el paradigma de programación funcional, cuyos lenguajes proporcionan conceptos muy entendibles y relativamente fáciles de manejar. Es por esta razón que se pretende utilizar el lenguaje de programación ***Racket***, de la familia de Lisp y Scheme, para el diseño del sistema solicitado, un lenguaje compacto con alto grado de abstracción, que facilita el razonamiento de los subprogramas que necesitan ser desarrollados para implementar cada una de las funcionalidades requeridas por el sistema, las cuales son pensadas de acuerdo a la metodología de solución conocida como ***división en subproblemas.***

En cuanto a las herramientas utilizadas, solo se requiere de un archivo de texto y el entorno de desarrollo integrado ***DrRacket,*** además del paquete ***Planet csv.plt*** proporcionado por ***neil,*** el cual debe ser descargado e instalado desde internet (DrRacket se encarga de hacerlo automáticamente).

1.6 ORGANIZACIÓN DEL DOCUMENTO.

Posteriormente, el presente informe se encuentra estructurado en una variedad de capítulos con el propósito de desarrollar diferentes temas relacionados con la manera en que fue desarrollado el proyecto y cuales fueron los resultados obtenidos.

Entre los capítulos incluidos, es posible identificar una sección referente a la **Descripción de los métodos utilizados**, en donde se explica que técnicas y conceptos fueron necesarios para desarrollar el

programa solicitado, seguido de una **Descripción de la solución**, en donde se explica como fue afrontada cada funcionalidad para poder dar con su solución, además de una sección relativa a la **Exposición de resultados obtenidos,** en donde se exhibe el funcionamiento de las funcionalidades implementadas en el sistema computacional desarrollado. El informe termina con los capítulos referentes a las **Conclusiones**, **Instrucciones de uso** del programa y **Referencias** utilizadas.

CAPÍTULO 2: DESCRIPCIÓN DE LOS METODOS UTILIZADOS.

Como ya fue mencionado durante la introducción, para estos tipos de problemas resulta adecuado emplear la metodología de la ***división en subproblemas***, ya que esta consiste en una técnica de resolución de problemas complejos que pretende alcanzar la solución tras descomponer el problema principal en una serie de subproblemas relativamente más fáciles de manejar. Entonces, se pretende afrontar el diseño de las diferentes funcionalidades a partir del desarrollo de las diferentes sub-funcionalidades que los componen, las cuales deberían ser identificadas después de aplicar esta forma de abstracción.

Otra razón por la cual se recomienda esta metodología es porque permite una mayor legibilidad y reutilización del código, lo cual es de gran utilidad para programas que implican realizar instrucciones similares para tipos de datos diferentes, como lo es en el caso de este proyecto. En otras palabras, facilita la solución de problemas mediante el uso de soluciones ya conocidas a problemas de igual estructura. Esto último corresponde al concepto de ***Similitud de problemas,*** siendo más especifico, al concepto de ***similitud fuerte***.

Por otro lado, se emplea un mecanismo a través del cual se modela/representa un proceso, objeto, o cualquier cosa real o imaginaria para que sea operado en una máquina el cual es conocido como ***Tipo de dato abstracto (TDA).*** Un TDA se define a partir de una representación, *constructores* (funciones que dan vida a la representación), *funciones de pertenencia* (que verifican si un valor pertenece a un tipo de dato en particular), *selectores* (que permiten acceder y obtener datos contenidos en la representación), *modificadores* (que alternan los valores de la representación interna) y finalmente *funciones que operan sobre la representación.*

Cabe mencionar que una forma recurrente de manejar y solventar problemas en este tipo de lenguajes funcionales es el uso de la recursión, las cuales son usadas en problemas cuya solución depende de las soluciones de pequeñas instancias del mismo problema. De aquí surgen conceptos como recursión lineal por cola, cuando la llamada recursiva es la última instrucción en la función y recursión lineal no por la cola, cuando después de ejecutar todas las llamadas recursivas el método debe realizar una operación pendiente para completar le proceso.

CAPÍTULO 3: DESARROLLO DE LA SOLUCIÓN.

Antes de intentar desarrollar solución de cada una de las funcionalidades que debiesen ser implementadas en el sistema computacional solicitado, es preciso tener en cuenta los tipos de datos que se pretende representar y para que se necesita la representación de estos. A partir de la lectura del enunciado del proyecto, y teniendo en cuenta como se encuentran estructuradas la base de datos proporcionada por el centro clínico, se logran identificar los siguientes tipos de datos abstractos:

a) Paciente
b) Médico
c) Diagnóstico
d) Tratamiento
e) Vinculación Tratamiento Diagnostico
f) Vinculación Tratamientos Diagnostico Paciente
g) Vinculación Diagnostico Paciente

En el fondo, son tipos de datos abstractos que debiesen ser definidos con el propósito de representar cada uno de los registros contenidos en las diferentes bases de datos.

Entonces, como lo que se pretende es definir una serie de TDAs, es necesario generar para cada uno de ellos, los diferentes constructores, funciones de pertenencia, selectores y modificadores, además de las funciones que operan sobre ellos.

```
;CONSTRUCTOR:
(define (createPaciente IDPaciente rut email nombre a
        (if (and
            (number? IDPaciente)
            (> IDPaciente -1)
            (string? rut)
            (string? email)
            (string? nombre)
            (string? apellido)
            (string? fecha_Nacimiento)
            )

            (list IDPaciente rut email nombre apellic

            null

        )

  );FIN CONSTRUCTOR.

  ;FUNCIÓN DE PERTENENCIA:
  (define (isPaciente? paciente)
    (if (list? paciente)
        (if (= (length paciente) 6)
            ;Si cumple
            (if (and
                 (number? (car paciente))
                 (> (car paciente) -1)
                 (string? (cadr paciente))
                 (string? (caddr paciente))
                 (string? (cadddr paciente))
                 (string? (caddddr paciente))
                 (string? (cadddddr paciente))
                )

                #t

                #f

                ); fin de la sentecia if n°3.

            #f

            ); fin de la sentencia if n°2.

        #f

        ); fin de la sentencia if n°1.
  )
```

```
;SELECTORES:
(define (getIDPaciente paciente)
  (if (isPaciente? paciente)
      (car paciente)
      -1
      )
  )

(define (getRUTPaciente paciente)
  (if (isPaciente? paciente)
      (cadr paciente)
      -1
      )
  )

(define (getEmailPaciente paciente)
  (if (isPaciente? paciente)
      (caddr paciente)
      -1
      )
  )

(define (getNombrePaciente paciente)
  (if (isPaciente? paciente)
      (cadddr paciente)
      -1
      )
  )

;MODIFICADORES
(define (setEmailPaciente paciente nuevoEmail)
  (if (isPaciente? paciente)

      (createPaciente
                          (getIDPaciente paciente)
                          (getRUTPaciente paciente)
                          nuevoEmail
                          (getNombrePaciente paciente)
                          (getApellidoPaciente paciente)
                          (getFecha_NacimientoPaciente paciente)
      )

      null

     )
  )
```

Figura 3-1: Funciones que definen el tipo de dato abstracto Pacientes.

Por otro lado, para poder extraer la información contenida en las diferentes bases de datos facilitadas, es decir, para extraer cada uno de los registros relacionados con los tipo de datos abstractos definidos anteriormente, se recurre a una función de la librería Planet neil/csv:2:0 (csv->list (open-input-file)) que permite abrir un archivo de texto y construir una lista con todos los registros contenidos en dicho archivo, siendo estos a su vez una lista de diferentes tipos de datos. Sin embargo, estos registros no están convertidos al tipo de dato abstracto que representan, por lo que es necesario definir para cada uno de ellos una función que permita realizar la conversión, o mejor dicho, para realizar la construcción de cada uno de los tipos de datos abstractos definidos anteriormente.

Aplicando el método de la *división en subproblemas* y teniendo en cuenta el concepto de *similitud fuerte*, es posible distinguir las siguientes instrucciones básicas que en conjunto son capaces de construir una de las funcionalidades requeridas:

a) Construir una lista de un determinado tipo de dato abstracto a partir de los registros obtenidos usando la función de la librería Planet neil/csv:2.
b) Verificar la existencia de cierto dato en un determinado registro.
c) Obtener el (los) registros relacionados con un determinado elemento de los registros.
d) Extraer y enlistar un determinado elemento desde una lista de registros.
e) Re-escribir uno de los archivos de texto de la base de datos con una determinada secuencia de caracteres (texto).

Generalizando, aquellas funciones cuyo propósito es obtener un único dato requieren de la aplicación de métodos recursivos lineales por cola. Mientras que aquellas funciones que pretenden obtener más de un mismo tipo de dato, los cuales son devueltos como elementos de una lista, recurren a recursiones del tipo lineal, no por cola.

Por otro lado, es necesario declarar otro tipo de funciones complementarias, las cuales permiten comparar fechas, enlistar elementos sin repetirlos, contar la cantidad de veces que se repite un elemento en una lista, entre otras.

CAPÍTULO 4: EXPOSICIÓN DE LOS RESULTADOS OBTENIDOS.

Debido a la basta cantidad de funcionalidades implementadas en el sistema computacional desarrollado, solamente los resultados de unos cuantos de ellos serán expuestos a continuación: Esto para las funcionalidades:

a) Obtener el nombre de un paciente dado su Rut.
b) Obtener la especialidad de un médico a partir de su Rut.
c) Listar el (o los) tratamiento(s) de acuerdo a un nivel de riesgo dado.
d) Determinar el (o los) tratamiento(s) más usado(s) para un diagnostico particular, indicando la cantidad de veces que se ha empleado.
g) Dado el identificador de un paciente, indicar el rut, nombre y apellido de todos los médicos que lo han tratado (sin incluir a los que lo dieron de alta).
j) Dado el rut de un paciente, indicar el rut, nombre y apellido de todos los médicos que lo han tratado (excluyendo a los que lo han dado de alta).

```
Bienvenido a DrRacket, versión 6.1 [3m].
Lenguaje: racket [custom]; memory limit: 128 MB.


> (display (obtenerNombrePaciente "rutPaciente23"))
nombrePaciente23

> (display (tratamientoRiesgoso "alto"))
nombretratamiento4
nombretratamiento10
nombretratamiento14
nombretratamiento20
nombretratamiento24
nombretratamiento30
nombretratamiento34
nombretratamiento40
nombretratamiento44
nombretratamiento50
nombretratamiento54
nombretratamiento60
nombretratamiento64
nombretratamiento70
nombretratamiento74
nombretratamiento80
nombretratamiento84
nombretratamiento90
nombretratamiento94

> (display (especialidad "rutdoctor99"))
Dermatologia

> (display (tratamientoMasUsadoPorDiagnostico 66))
nombretratamiento5 2

> (display medicoMasAltas)
rutdoctor96,nombredoctor96
apellidodoctor96,19
```

```
> (display tratamientoMasUsado)
nombretratamiento28,17
nombretratamiento72,17

> (display (medicosTratantes 66))
rutdoctor5 nombredoctor5 apellidodoctor5
rutdoctor16 nombredoctor16 apellidodoctor16
rutdoctor25 nombredoctor25 apellidodoctor25
rutdoctor26 nombredoctor26 apellidodoctor26
rutdoctor27 nombredoctor27 apellidodoctor27
rutdoctor51 nombredoctor51 apellidodoctor51
rutdoctor53 nombredoctor53 apellidodoctor53
rutdoctor54 nombredoctor54 apellidodoctor54
rutdoctor69 nombredoctor69 apellidodoctor69
rutdoctor80 nombredoctor80 apellidodoctor80
rutdoctor86 nombredoctor86 apellidodoctor86
rutdoctor88 nombredoctor88 apellidodoctor88
rutdoctor93 nombredoctor93 apellidodoctor93
rutdoctor95 nombredoctor95 apellidodoctor95

> (display (listarMedicosTratantesPaciente "rutPaciente66"))
rutdoctor5 nombredoctor5 apellidodoctor5
rutdoctor16 nombredoctor16 apellidodoctor16
rutdoctor25 nombredoctor25 apellidodoctor25
rutdoctor26 nombredoctor26 apellidodoctor26
rutdoctor27 nombredoctor27 apellidodoctor27
rutdoctor51 nombredoctor51 apellidodoctor51
rutdoctor53 nombredoctor53 apellidodoctor53
rutdoctor54 nombredoctor54 apellidodoctor54
rutdoctor69 nombredoctor69 apellidodoctor69
rutdoctor80 nombredoctor80 apellidodoctor80
rutdoctor86 nombredoctor86 apellidodoctor86
rutdoctor88 nombredoctor88 apellidodoctor88
rutdoctor93 nombredoctor93 apellidodoctor93
rutdoctor95 nombredoctor95 apellidodoctor95

>
```

Figura 4-2: Exposición de los resultados.

CAPÍTULO 5: CONCLUSIÓN.

Teniendo en cuenta las dificultades que se presentaron durante el desarrollo del anterior proyecto de la asignatura, proyecto que consistía en desarrollar el mismo sistema computacional aplicando lo expuesto por el enfoque de programación imperativo, se ha comprendido, de una manera practica, como el enfoque de programación funcional resulta ser más efectivo al momento de resolver problemas de gran complejidad. Dicha efectividad se debe no solamente a la proporción de conceptos entendibles y relativamente fáciles de manejar, tal y como se menciono durante la introducción, sino que también a la posibilidad de describir completamente los subprogramas a partir de los datos que entran y de los datos que salen, además de la facilidad con la que se puede reutilizar el código y realizar pruebas independientes. Las facilidades que entregan estos tipos de lenguajes permiten de gran manera mejorar la productividad a nivel de programación.

Por otro lado, existe la probabilidad de que las técnicas de programación utilizadas para la construcción del sistema computacional solicitado no hayan sido del todo funcionales, es decir que, debido a la inexperiencia en este tipo de enfoque, es posible que conceptos de los lenguajes procedimentales hayan sido u otras metodologías hayan sido aplicadas en este proyecto. Un ejemplo de esto es el uso de la función *map*, que corresponde a un proceso iterativo, por lo cual no debiese ser aplicada en un programa de enfoque puramente funcional.

CAPÍTULO 6: INSTRUCCIONES DE USO:

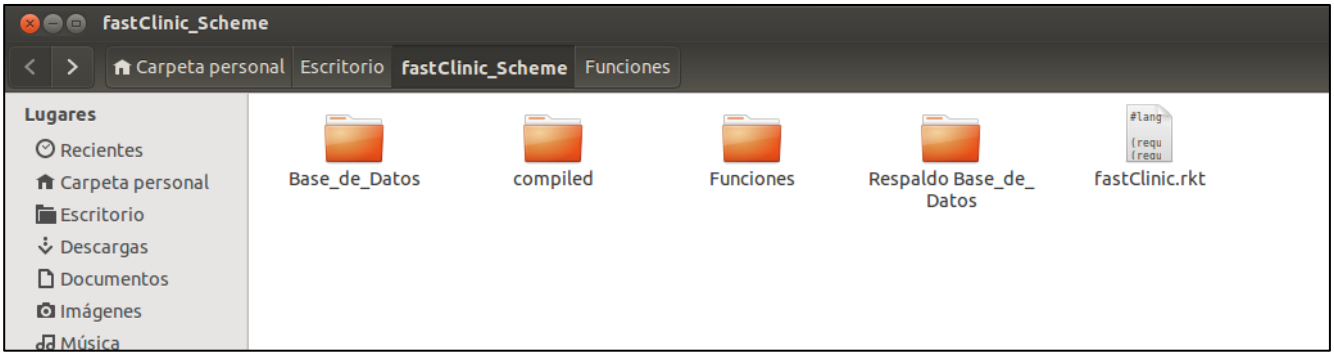
fastClinic es el nombre de un sistema computacional desarrollado para ser utilizado especificamente por los operarios del centro clínico de igual nombre con el propósito de facilitar las tareas de administración. Dicho sistema hace uso de un conjunto de archivos de texto que representan una base de datos de simple estructura.

Las funcionalidades facilitadas por el software son las siguientes:

a) Obtener el nombre de un paciente dado su Rut.
b) Obtener la especialidad de un médico a partir de su Rut.
c) Listar el (o los) tratamiento(s) de acuerdo a un nivel de riesgo dado.
d) Determinar el (o los) tratamiento(s) más usado(s) para un diagnostico particular, indicando la cantidad de veces que se ha empleado.
e) Identificar el (o los) médico(s) que más altas otorga, indicando su rut, nombre y apellido y la cantidad de altas.
f) Identificar el (o los) tratamiento(s) más usado(s) en todo el sistema indicando la cantidad de veces que se ha(n) usado.
g) Dado el identificador de un paciente, indicar el rut, nombre y apellido de todos los médicos que lo han tratado (sin incluir a los que lo dieron de alta).
h) Dado el nombre y apellido de una persona, indicar cuál es el nivel de riesgo del último tratamiento recibido.
i) Conocer todos los pacientes diagnosticados con el diagnóstico X (nombre del diagnóstico) a los cuales el doctor Y (rut) les dio el Alta. Mostrando el rut, nombre y apellido de esos pacientes.
j) Dado el rut de un paciente, indicar el rut, nombre y apellido de todos los médicos que lo han tratado (excluyendo a los que lo han dado de alta).
k) Modificar el correo electrónico de un paciente.
l) Dado el nombre y apellido de un paciente, además del id de un tratamiento, modificar el resultado de su tratamiento más reciente.
m) Dado el correo del paciente y del médico, modificar el médico que da el alta en su última ocasión.

Para poder ejecutar el programa, es necesario tener instalado el entorno de desarrollo integrado DrRacket, además de estar conectado una primera vez a internet para poder llevar a cabo la descarga e instalación automática de una de las librerías necesarias para que el sistema pueda funcionar como debe hacerlo.

El sistema se compone de dos carpetas principales, una para contener los archivos relacionado con la base de datos y otra para contener los archivos relacionados con los tipos de datos abstractos definidos, junto con las funciones que operan sobre ellas, y un archivo principal fastClinic.rkt.



.

Figura 6-3: Carpeta principal del sistema.

Para poder ejecutar la aplicación, es necesario abrir el archivo principal (fastClinic.rkt) a través del entorno de desarrollo integrado DrRacket (Es posible ejecutarlo desde la consola de GNU/LINUX, pero resulta más simple hacerlo de esta manera).

DrRacket provee de una ventana de interacción, en la cual es posible ejecutar cada una de las funcionalidades implementadas mediante la escritura de comandos, tal y como se puede apreciar en la siguiente imagen:

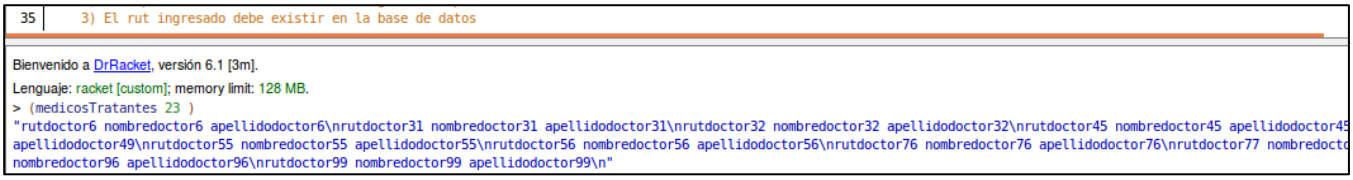


Figura *6-4: Ejecución* de la funcionalidad medicosTratantes.

A continuación se enlistan cada una de las funcionalidades que pueden ser ejecutadas desde la ventana de interacción:

a) Función (display): Comando utilizado para imprimir por pantalla la salida de una función. Resulta útil ya que puede reconocer los caracteres de salto de línea "\n" lo que facilita el entendimiento de múltiples resultados. Su uso se ejemplifica a continuación:

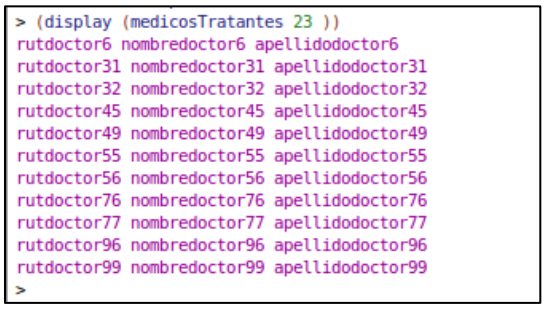


Figura 6-5: Display de la ejecución de la funcionalidad medicosTratantes.

b) Función (obtenerNombrePaciente <rut>): Utilizada para el nombre de un paciente dado su rut, parámetro que debe ser ingresado como un texto delimitado por comillas (" "). Ejemplo de uso:

(display (obtenerNombrePaciente "rutPaciente69")).

c) Función (especialidad <rut>): Utilizada para obtener la especialidad de un médico a partir de su rut, el cual debe ser ingresado como un texto delimitado por comillas (" "). Ejemplo de uso:

(display (especialidad "rutdoctor69")).

d) Función (tratamientoRiesgoso <nivel>): Utilizada para listar el (o los) tratamiento(s) de acuerdo a un nivel de riesgo dado (ingresado como un texto delimitado por comillas (" ")). Ejemplo de uso:

(display (tratamientoRiesgoso "alto")).

e) Función (tratamientoMasUsadoPorDiagnostico <identificador Diagnostico>): Utilizada para determinar el (o los) tratamiento(s) más usado(s) para un diagnóstico particular indicando la cantidad de veces que se ha empleado. Ejemplo de uso:

(display (tratamientoMasUsadoPorDiagnostico 66)).

f) Función medicoMasAltas: Utilizada para identificar el (o los) médico(s) que más altas otorga indicando su rut, nombre y apellido, y la cantidad de altas. Ejemplo de uso:

(display medicoMasAltas).

g) Función tratamientoMasUsado: Utilizada para identificar el (o los) tratamiento(s) más usado(s) en todo el sistema indicando la cantidad de veces que se ha(n) usado. Ejemplo de uso:

(display tratamientoMasUsado).

h) Función (medicosTratantes <identificador paciente>): Utilizada para, tras haber dado el identificador de un paciente, indicar el rut, nombre y apellido de todos los médicos que lo han tratad, sin incluir a los que lo dieron de alta. Ejemplo de uso:

(display (medicosTratantes 23)).

i) Función (riesgoUltimoTratamiento <nombre> <apellido>): Utilizada para, tras haber dado el nombre y apellido de una persona (ambas ingresadas como dos textos diferentes delimitado por comillas y separadas por un espacio en blanco), indicar cuál es el nivel de riesgo del último tratamiento recibido. Ejemplo de uso:

(display (riesgoUltimoTratamiento “nombrePaciente23” “apellidoPaciente23”)).

j) Función (diagnosticoPacienteMedico <nombre diagnostico> <rut doctor>): Utilizada para conocer todos los pacientes diagnosticados con el diagnóstico X (nombre del diagnóstico) a los cuales el doctor Y (rut) les dio el Alta. (Ambos ingresados como textos delimitados por comillas). Ejemplo de uso:

(display (diagnosticoPacienteMedico “diagnostico3” “rutdoctor77”)).

k) Función (listarMedicosTratantesPaciente <rut paciente>): Utilizada para, dado el rut de un paciente ingresado como un texto delimitado por comillas, indicar el rut, nombre y apellido de todos los médicos que lo han tratado. Ejemplo de uso:

(display (listarMedicosTratantesPaciente “rutPaciente23”)).

l) Función (modificarCorreoPaciente <correo antiguo> <correo nuevo>): Utilizada para modificar, en los registros contenidos por la base de datos, el correo electrónico de un paciente, cuyo correo es conocido. (Ambas entregadas como textos delimitados por comillas). Ejemplo de uso:

(modificarCorreoPaciente “correoAntiguo@yahoo.com” “correoNuevo@gmail.com”).

m) Función (modificarResultadoTratamiento <nombre> <apellido> < idTratamiento> <nuevo resultado>): Utilizada para modificar, en los registros contenidos por la base de datos, el resultado del tratamiento mas reciente de un paciente, cuyo nombre y apellido es entregado como dos textos delimitados por comillas. Ejemplo de uso:

(modificarResultadoTratamiento “nombrePaciente2” “apellidoPaciente2” 23 “Muerte cerebral”).

n) Función (modificarMedicoAlta <emailPaciente> <emailMedico>): Utilizada para modificar, en los registros contenidos por la base de datos, el medico que le dio la ultima alta registrada de un paciente. Ambos conocidos por los emails ingresados como textos delimitados por comillas. Ejemplo de uso:

(modificarMedicoAlta “mailPaciente@gmail.com” “mailMedico@gmail.com”).

En caso de que se presente cualquier problema relacionado con el funcionamiento del sistema, haga el favor de contactarse con el desarrollador: felipe.jara.r@usach.cl

CAPÍTULO 7: REFERENCIAS:

1. The Scheme Programming LanguageThird Edition R. Kent Dybvig
2. http://www.monografias.com/trabajos30/paradigma-funcional/paradigma-funcional.shtml
3. http://racket-lang.org/

CAPÍTULO 8: ANEXOS.

8.1 DOCUMENTACIÓN:

En esta sección del informe se presenta la documentación de aquellas funciones que operan sobre los tipos de datos abstractos definidos para un correcto funcionamiento del sistema computacional. Esto quiere decir que no se incluyen los constructores, modificadores, selectores ni funciones de pertenencia.

8.1.1 Diagnostico.rkt

Función extraer_Registros_Diagnosticos

Descripción: Función utilizada para retornar una lista de diagnosticos construidos a partir de los registros contenidos en la base de datos Diagnostico.txt. Se basa en el uso de una aplicación de recursión lineal, no por cola, para poder extraer cada uno de los registros y construir con ellos los diferentes diagnósticos contenidos en la base de datos. Estos son agregados a una lista, a la cual se retorna tras finalizar la ejecución de la función en el caso de que todo salga bien. Por otro lado, si algún error ocurre durante el proceso, se devuelve una lista vacía (null).

Dominio: Registros extraídos a partir de la base de datos Diagnostico.txt.

INPUT:

Lista de los registros contenidos en la base de datos Diagnostico.txt.

OUTPUT:

Lista de diagnósticos.

Función obtener_Registros_Diagnosticos_Segun_Nivel

Descripción: Función utilizada para retornar una lista de registros diagnósticos que contiene un nivel de gravedad especifico. Recurre a una recursión del tipo lineal no por cola para poder unir cada uno de los registros que cumplen con las condiciones mencionadas. En el caso de que no existan dichos registros, se retorna una lista vacía (null).

Dominio: Lista de registros diagnósticos entregado como parámetro.

INPUT:

Lista de registros de diagnósticos.

Nivel de riesgo.

OUTPUT:

Lista de diagnósticos que contienen un nivel de gravedad en especifico.

8.1.2 Paciente.rkt

Función obtener_Registros_Diagnosticos_Segun_Nombre

Descripción: Función utilizada para retornar el registro de un Diagnostico cuyo nombre se conoce, asumiendo que el nombre de un diagnostico es ubico, por lo que se recurre a una recursión del tipo lineal por cola para estudiar registro por registro hasta encontrar el que corresponde. En el caso de que no exista dicho registro, se retorna null.

Dominio: Lista de registros diagnósticos entregado como parámetro.

INPUT:

Nombre del diagnostico.

Lista de registros de diagnósticos.

OUTPUT:

Registro del diagnostico que contiene el nombre ingresado.

Función extraer_Registros_Pacientes

Descripción: Función utilizada para retornar una lista de pacientes construidos a partir de los registros contenidos en la base de datos Pacientes.txt. Se basa en el uso de una aplicación de recursión lineal, no por cola, para poder extraer cada uno de los registros y construir con ellos los diferentes pacientes contenidos en la base de datos. Estos son agregados a una lista, a la cual se retorna tras finalizar la ejecución de la función en el caso de que todo salga bien. Por otro lado, si algún error ocurre durante el proceso, se devuelve una lista vacia (null).

Dominio: Registros extraídos a partir de la base de datos Pacientes.txt.

INPUT:

Lista de registros contenidos en la base de datos Pacientes.txt.

OUTPUT:

Lista de registros pacientes.

Función obtener_ID_Paciente_segun_RUT

Descripción: Función utilizada para retornar el identificador de un paciente contenido en una lista de registros cuyo rut corresponda al que es entregado como parámetro. Para poder analizar registro por registro, se recurre a una recursión del tipo lineal por cola hasta encontrar el que se esta buscando. En el caso de que dicho registro no se encuentre en la lista, se retorna -1 como una forma de indicar que no existe tal identificador.

Dominio: Lista de registros pacientes entregado como parámetro.

INPUT:

Rut del paciente.

Lista de registros de pacientes.

OUTPUT:

Identificador del paciente.

Función obtener_Registro_Paciente_segun_RUT

Descripción: Función utilizada para retornar el registro de un paciente comparando un rut ingresado con el rut de todos los registros de los pacientes. Para realizar dichas comparaciones, se requiere hacer uso de una recursión por cola hasta encontrar el registro buscado. En caso de que no exista, se retorna una lista vacía (null).

Dominio: Lista de registros pacientes entregado como parámetro.

INPUT:

Rut del paciente.

Lista de registros de pacientes.

OUTPUT:

Registro del paciente.

Función obtener_Registro_Paciente_segun_Email

Descripción: Función utilizada para retornar el registro de un paciente comparando un email ingresado con el email de todos los registros de los pacientes. Para realizar dichas comparaciones, se requiere hacer uso de una recursión por cola hasta encontrar el registro buscado. En caso de que no existas, se retorna una lista vacía (null).

Dominio: Lista de registros pacientes entregado como parámetro.

INPUT:

Email del paciente.

Lista de registros de pacientes.

OUTPUT:

Registro del paciente.

Función obtener_Registro_Paciente_segun_ID

Descripción: Función utilizada para retornar el registro de un paciente comparando un identificador paciente ingresado con el identificador paciente de todos los registros de los pacientes. Para realizar dichas comparaciones, se requiere hacer uso de una recursión por cola hasta encontrar el registro buscado. En caso de que no exista, se retorna una lista vacia (null).

Dominio: Lista de registros pacientes entregado como parámetro.

INPUT:

Identificador del paciente.

Lista de registros de pacientes.

OUTPUT:

Registro del paciente.

Función obtener_Registro_Paciente_segun_Nombre_Apellido

Descripción: Función utilizada para retornar el registro de un paciente relacionado al nombre y apellido entregado como parámetro de entrada. Para ello recurre una recursión lineal del tipo no por cola, analizando registro por registro hasta encontrar el deseado.

Dominio: Lista de registros pacientes entregado como parámetro.

INPUT:

Nombre del paciente.

Apellido del paciente.

Lista de registros de pacientes.

OUTPUT:

Registro del paciente.

Función verificar_Email_En_Registro_Paciente

Descripción: Función utilizada para retornar Verdadero o Falso en caso de que un email exista o no, respectivamente, dentro de una lista de registros de tipo paciente.

Dominio: Lista de registros pacientes entregado como parámetro.

INPUT:

Listas de registros pacientes.

Mail.

OUTPUT:

Verdadero o Falso.

Función modificar_Email_En_Registro_Paciente

Descripción: Función utilizada para retornar la lista de registros, cambiando el email de uno de ellos. Asumiendo que el cambio es valido, es decir, que ya se sabe que el email antiguo existe en la lista de registros y que el email nuevo puede reemplazar. Para poder construir una lista de todos los registros pacientes, incluyendo aquel que fue modificado, se emplea una recursión del tipo lineal no por cola.

Dominio: Lista de registros pacientes entregado como parámetro.

INPUT:

Listas de registros pacientes.

Mail Antiguo.

Mail Nuevo.

OUTPUT:

Listas de registros pacientes modificadas.

8.1.3 DiagnosticoPaciente.rkt

Función extraer_Registros_DiagnosticoPacientes

Descripción: Función utilizada para retornar una lista de diagnosticoPacientes construidos a partir de los registros contenidos en la base de datos DiagnosticoPaciente.txt. Se basa en el uso de una aplicación de recursión lineal, no por cola, para poder extraer cada uno de los registros y construir con ellos los diferentes diagnosticoPacientes contenidos en la base de datos. Estos son agregados a una lista, a la cual se retorna tras finalizar la ejecución de la función en el caso de que todo salga bien. Por otro lado, si algún error ocurre durante el proceso, se devuelve una lista vacía (null).

Dominio: Registros extraídos a partir de la base de datos DiagnosticoPaciente.txt.

INPUT:

Lista de registros contenidos en la base de datos DiagnosticoPaciente.txt.

OUTPUT:

Lista de diagnosticoPacientes.

Función obtener_Identificadores_Doctor_Alta

Descripción: Función utilizada para retornar una lista con todos los identificadores de los doctores que dan el alta y que están contenidos en los registros diagnosticoPaciente. Se utiliza una recursión del tipo lineal no por cola para poder unir todos los identificadores que cumplan con la condición mencionada a medida que se recorren los elementos de una lista de registros.

Dominio: Lista de registros diagnosticoPacientes entregado como parámetro.

INPUT:
Lista de registros diagnosticoPaciente.
OUTPUT:
Lista de identificadores doctor alta.

Función obtener_Registros_DiagnosticoPaciente_Segun_IDPaciente

Descripción: Función utilizada para retornar todos los registros relacionados con el identificador de un paciente. Para ello se utiliza una recursión del tipo lineal no por cola para analizar registro por registro y en listar todos los que contengan al identificador ingresado como entrada.

Dominio: Lista de registros diagnosticoPacientes entregado como parámetro.

INPUT:
Identificador del paciente.
Lista de registros diagnosticoPaciente.
OUTPUT:
Lista de registros relacionados con el paciente.

Función obtener_Registros_DiagnosticoPaciente_Segun_IDDoctorAlta_IDDiagnostico

Descripción: Función utilizada para retornar todos los registros relacionados con el identificador de un médico que da el alta y el identificador de un diagnostico. Para ello se utiliza una recursión del tipo lineal no por cola para analizar registro por registro y en listar todos los que contengan al identificador ingresado como entrada.

Dominio: Lista de registros diagnosticoPacientes entregado como parámetro.

INPUT:
Identificador del medico de alta.
Identificador del diagnostico.

Lista de registros diagnosticoPaciente.

OUTPUT:

Lista de registros relacionados con el medico de alta y diagnostico.

Función obtener_Registros_DiagnosticoPaciente_Segun_FechaAlta

Descripción: Función utilizada para retornar todos los registros relacionados con una fecha de alta. Para ello se utiliza una recursión del tipo lineal por cola para analizar registro por registro y en listar aquel contenga la fecha ingresada como parámetro de entrada. Como un paciente no puede ser dado de alta dos veces en un mismo día, se asume que solo hay una fecha de alta para el registro DiagnosticoPaciente de un paciente en específico.

Dominio: Lista de registros diagnosticoPacientes entregado como parámetro.

INPUT:

Fecha de alta.

Lista de registros diagnosticoPaciente.

OUTPUT:

Lista de registros relacionados con la fecha de alta.

Función modificar_IDDoctorAlta_Registro_DiagnosticoPaciente_Segun_Registro

Descripción: Función utilizada para retornar una lista de todos los registros, con la diferencia de que el doctor de alta de uno de ellos sea modificado. Para acceder registro por registro y reconstruir la lista se recurre a una recursión lineal no por cola.

Dominio: Lista de registros diagnosticoPacientes entregado como parámetro.

INPUT:

Identificador Medico reemplazante.

Registro a modificar.

Lista de registros diagnosticoPaciente.

OUTPUT:

Lista de registros modificada.

8.1.4 Doctor.rkt

Función extraer_Registros_Doctores

Descripción: Función utilizada para retornar una lista de doctores construidos a partir de los registros contenidos en la base de datos Doctor.txt. Se basa en el uso de una aplicación de recursión lineal, no por cola, para poder extraer cada uno de los registros y construir con ellos los diferentes doctores contenidos en la base de datos. Estos son agregados a una lista, a la cual se retorna tras finalizar la ejecución de la función en el caso de que todo salga bien. Por otro lado, si algún error ocurre durante el proceso, se devuelve una lista vacía (null).

Dominio: Registros extraídos a partir de la base de datos Doctor.txt.

INPUT:

Lista de registros contenidos en la base de datos Doctor.txt.

OUTPUT:

Lista de doctores.

Función obtener_ID_Doctor_segun_RUT

Descripción: Función utilizada para retornar el identificador de un doctor comparando un rut ingresado con el rut de todos los registros de los doctores. Utiliza una recursión del tipo lineal por cola para analizar registro por registro hasta encontrar el registro buscado. En el caso de que no sea posible encontrarlo, se retorna una lista vacía (null).

Dominio: Lista de registros doctores entregado como parámetro.

INPUT:

Rut del doctor.

Lista de registros de doctores.

OUTPUT:

Identificador del doctor.

Función obtener_Registro_Doctor_segun_RUT

Descripción: Función utilizada para retornar el registro de un doctor comparando un rut ingresado con el rut de todos los registros de los doctores. Recurre a una recursión del tipo lineal por cola para analizar registro por registro hasta encontrar el registro buscado. En el caso de que no sea posible encontrarlo, se retorna una lista vacía (null).

Dominio: Lista de registros doctores entregado como parámetro.

INPUT:
Rut del doctor.
Lista de registros de doctores.
OUTPUT:
Registro del doctor.

Función obtener_Registro_Doctor_segun_Email

Descripción: Función utilizada para retornar el registro de un doctor comparando un email ingresado con el email de todos los registros de los doctores. Recurre a una recursión del tipo lineal por cola para analizar registro por registro hasta encontrar el registro buscado. En el caso de que no sea posible encontrarlo, se retorna una lista vacía (null).

Dominio: Lista de registros doctores entregado como parámetro.

INPUT:
Email del doctor.
Lista de registros de doctores.
OUTPUT:
Registro del doctor.

Función obtener_RUT_Nombre_Apellido_Doctor_segun_Identificador

Descripción: Función utilizada para retornar el rut, nombre y apellido de un doctor relacionado con un identificador ingresado como entrada. Recurre a una recursión del tipo lineal por cola para analizar registro por registro hasta encontrar el que corresponde. En el caso de que no pueda ser encontrado dicho registro, se retorna una lista vacía (null).

Dominio: Lista de registros doctores entregado como parámetro.

INPUT:

Identificador.

Lista de registros de doctores.

OUTPUT:

Rut, nombre y apellido de un doctor como una cadena de caracteres de la forma "rut,nombre apellido".

Función obtener_RUT_Nombre_Apellido_Doctor_segun_Identificador_V2

Descripción: Función utilizada para retornar el rut, nombre y apellido de un doctor relacionado con un identificador ingresado como entrada. Recurre a una recursión del tipo lineal por cola para analizar registro por registro hasta encontrar el que corresponde. En el caso de que no pueda ser encontrado dicho registro, se retorna una lista vacía (null).

Dominio: Lista de registros doctores entregado como parámetro.

INPUT:

Identificador.

Lista de registros de doctores.

OUTPUT:

Rut, nombre y apellido de un doctor como una cadena de caracteres de la forma "rut nombre apellido".

Función obtener_RUT_Nombre_Apellido_Doctores_segun_Identificadores

Descripción: unción utilizada para retornar el rut, nombre y apellido de los doctores relacionados con ciertos identificadores ingresado como entrada. Recurre a una recursión del tipo lineal no por cola para analizar registro por registro, haciendo uso de la función obtener_RUT_Nombre_Apellido_Doctor_segun_Identificador.

Dominio: Lista de registros doctores entregado como parámetro.

INPUT:

Lista de identificadores.

Lista de registros de doctores.

OUTPUT:

Lista con el Rut, nombre y apellido de cada doctor, escrito cada uno como una cadena de caracteres de la forma "rut,nombre apellido".

8.1.5 Tratamiento.rkt

Función extraer_Registros_Tratamientos

Descripción: Función utilizada para retornar una lista de tratamientos construidos a partir de los registros contenidos en la base de datos Tratamiento.txt. Se basa en el uso de una aplicación de recursión lineal, no por cola, para poder extraer cada uno de los registros y construir con ellos los diferentes tratamientos contenidos en la base de datos. Estos son agregados a una lista, a la cual se retorna tras finalizar la ejecución de la función en el caso de que todo salga bien. Por otro lado, si algún error ocurre durante el proceso, se devuelve una lista vacía (null).

Dominio: Registros extraídos a partir de la base de datos Tratamiento.txt.

INPUT:

Lista de registros contenidos en la base de datos Tratamiento.txt.

OUTPUT:

Lista de tratamientos.

Función obtener_Registro_Tratamiento_Segun_Identificador

Descripción: Función que utiliza una recursión por cola para encontrar y retornar un registro de tipo tratamiento que cumpla con la siguiente condición: El registro debe contener el mismo identificador que el que es ingresado como argumento de la función. En el caso de que no exista registro que cumpla con ello, se retorna null

Dominio: Lista de registros tratamientos entregado como parametro.

INPUT:

Identificador del tratamiento

Lista de registros de tratamientos

OUTPUT:

Registro Tratamiento relacionado con el Identificador de entrada

Función obtener_Nombre_Tratamiento_Segun_Identificador

Descripción: Función que utiliza una recursión por cola para encontrar y retornar el dato "nombre" de un registro de tipo tratamiento que cumpla con la siguiente condición: El registro debe contener el mismo identificador que el que es ingresado como argumento de la función. En el caso de que no cumpla con ello, se retorna null.

Dominio: Lista de registros tratamientos entregado como parámetro.

INPUT:

Identificador del tratamiento.

Lista de registros tratamiento.

OUTPUT:

Nombre del tratamiento.

Función obtener_Nombres_Tratamientos_Segun_Identificadores

Descripción: Función utilizada para retornar una lista de nombres de los registros de tipo tratamiento que están relacionados con una lista de identificadores entregados como parametro de la función. Para cumplir con dicho propósito, la función requiere hacer uso de una recursión del tipo lineal, no por cola, para construir la lista requerida.

Dominio: Lista de registros tratamientos entregado como parámetro.

INPUT:

Lista de identificadores.

Lista de registros tratamiento.

OUTPUT:

Lista de nombres de tratamiento.

Función obtener_Registros_Tratamientos_Segun_Identificadores

Descripción: Función utilizada para retornar una lista de tratamientos relacionados con una lista de identificadores de tratamiento. Se recurre a una recursión del tipo lineal no por cola para unir cada uno de los registros que cumplen con la condición señalada en una lista.

Dominio: Lista de registros tratamientos entregado como parámetro.

INPUT:

Lista de identificadores.

Lista de registros de tratamiento.

OUTPUT:

Lista de tratamientos que contienen un nivel de riesgo en especifico.

Función obtener_Registros_Tratamientos_Segun_Nivel

Descripción: Función utilizada para retornar una lista de tratamientos que contienen un nivel de riesgo especifico. Se utiliza una recursión del tipo lineal no por cola para unir cada uno de los registros que cumplen con la condición señalada en una sola lista.

Dominio: Lista de registros tratamientos entregado como parámetro.

INPUT:

Lista de registros de tratamientos.

Nivel de riesgo.

OUTPUT:

Lista de tratamientos que contienen un nivel de riesgo en especifico

Función obtener_Nombres_Tratamientos_Segun_Lista

Descripción: Función utilizada para retornar una cadena de texto de nombres de tratamientos contenidos en una lista de tratamientos. Se recurre a una recursión del tipo lineal no por cola para poder unir en un string los nombres contenidos en la lista ingresada.

Dominio: Lista de registros tratamientos entregado como parámetro.

INPUT:

Lista de tratamientos.

OUTPUT:

Lista de nombres de tratamientos.

8.1.6 tratamientoDiagnostico.rkt

Función extraer_Registros_TratamientoDiagnosticos.

Descripción: Función utilizada para retornar una lista de tratamientoDiagnosticos construidos a partir de los registros contenidos en la base de datos TratamientoDiagnostico.txt. Se basa en el uso de una aplicación de recursión lineal, no por cola, para poder extraer cada uno de los registros y construir con ellos los diferentes tratamientoDiagnosticos contenidos en la base de datos. Estos son agregados a una

lista, a la cual se retorna tras finalizar la ejecución de la función en el caso de que todo salga bien. Por otro lado, si algún error ocurre durante el proceso, se devuelve una lista vacía (null).

Dominio: Registros extraídos a partir de la base de datos TratamientoDiagnostico.txt.

INPUT:
Lista de registros contenidos en la base de datos TratamientoDiagnostico.txt.
OUTPUT:
Lista de TratamientoDiagnosticos.

Función obtener_IDTratamientos

Descripción: Función utilizada para retornar una lista de todos los identificadores tratamientos contenidos en los registros. Recurre a una recursión del tipo lineal no por cola para obtener cada uno de los identificadores de cada uno de los registros y unirlos en una lista.

Dominio: Lista de registros tratamientoDiagnosticos entregado como parámetro.

INPUT:
Lista de registros tratamientoDiagnosticos.
OUTPUT:
Lista de identificadores Tratamiento.

Función obtener_IDTratamientos_Segun_IDDiagnostico

Descripción: Función utilizada para retornar una lista de identificadores tratamientos relacionados con cierto identificador diagnostico. Utiliza una recursión del tipo lineal no por cola para unir cada uno de los identificadores obtenidos en una lista.

Dominio: Lista de registros tratamientoDiagnosticos entregado como parámetro.

INPUT:
Identificador Diagnostico.
Lista de registros tratamientoDiagnosticos.

OUTPUT:

Lista de Tratamientos relacionados con el identificador Diagnostico ingresado.

8.1.7 TratamientoDiagnosticoPaciente.rkt:

Función extraer_Registros_TratamientoDiagnosticoPacientes

Descripción: Función utilizada para retornar una lista de tratamientoDiagnosticoPacientes construidos a partir de los registros contenidos en la base de datos TratamientoDiagnosticoPaciente.txt. Se basa en el uso de una aplicación de recursión lineal, no por cola, para poder extraer cada uno de los registros y construir con ellos los diferentes tratamientoDiagnosticoPacientes contenidos en la base de datos. Estos son agregados a una lista, a la cual se retorna tras finalizar la ejecución de la función en el caso de que todo salga bien. Por otro lado, si algún error ocurre durante el proceso, se devuelve una lista vacía (null).

Dominio: Registros extraídos a partir de la base de datos TratamientoDiagnosticoPaciente.txt.

INPUT:

Lista de registros contenidos en la base de datos TratamientoDiagnosticoPaciente.txt.

OUTPUT:

Lista de tratamientoDiagnosticoPacientes.

Función obtener_Registro_TratamientoDiagnosticoPaciente_segun_IDDiagnosticoPaciente

Descripción: Función utilizada para obtener el registro TratamientoDiagnosticoPaciente relacionado con el identificador DiagnosticoPaciente entregado como parámetro, recurriendo a una recursión de tipo lineal, no por cola, para comparar registro por registro hasta encontrar una similitud.

Dominio: Lista de registros TratamientoDiagnosticoPaciente entregado como parámetro.

INPUT:

Identificador DiagnosticoPaciente.

Registros TratamientoDiagnosticoPaciente.

OUTPUT:

Registro TratamientoDiagnosticoPaciente.

Función obtener_Registros_TratamientoDiagnosticoPaciente_segun_IDTratamiento

Descripción: Función utilizada para obtener el registro TratamientoDiagnosticoPaciente relacionado con el identificador identificador Tratamiento entregado como parámetro, recurriendo a una recursión de tipo lineal, no por cola, para comparar registro por registro hasta encontrar una similitud.

Dominio: Lista de registros TratamientoDiagnosticoPaciente entregado como parámetro.

INPUT:

Identificador Tratamiento.

Registros TratamientoDiagnosticoPaciente.

OUTPUT:

Registro TratamientoDiagnosticoPaciente.

Función obtener_Registro_TratamientoDiagnosticoPaciente_segun_Fecha

Descripción: Función utilizada para obtener el registro TratamientoDiagnosticoPaciente relacionado con la fecha de inicio entregado como parámetro, recurriendo a una recursión lineal por cola para comparar registro por registro hasta encontrar una similitud.

Dominio: Lista de registros TratamientoDiagnosticoPaciente entregado como parámetro.

INPUT:

Fecha.

Registros TratamientoDiagnosticoPaciente.

OUTPUT:

Registro TratamientoDiagnosticoPaciente.

Función modificar_Resultado_Registro_TratamientoDiagnosticoPaciente

Descripción: Función utilizada para devolver los registros ingresados inicialmente, con la diferencia de que uno de los resultados de sus registros ha sido modificado. Utiliza una recursión del tipo lineal, no por cola, para unir cada uno de los registros en una lista, incluyendo aquel que fue modificado.

Dominio: Lista de registros TratamientoDiagnosticoPaciente entregado como parámetro.

INPUT:

Nuevo Resultado.

Registro a modificar.

Registros TratamientoDiagnosticoPaciente.

OUTPUT:

Registros TratamientoDiagnosticoPaciente con la modificación señalada.

8.1.8 FuncionesGenerales.rkt

Función verificarElementoEnLista_V1

Descripción: Función que hace uso de una recursión linea por cola para verificar la existencia de cierto en una lista.

Dominio: Elementos de la lista ingresada como parámetro.

INPUT:

Elemento cuya existencia en la lista se desea verificar.

Lista sobre al cual se pretende trabajar.

OUTPUT:

#f Si el elemento no se encuentra en la lista.

#t Si el elemento se encuentra en la lista.

Función contarElementoEnLista

Descripción: Función utilizada para contar la cantidad de veces que se repite un elemento en una lista, utilizando para ello una recursión del tipo lineal, no por cola.

Dominio: Elementos de la lista ingresada como parámetro.

INPUT:

Elemento a contar.

Lista.

OUTPUT:

Cantidad de veces que se repite el elemento en la lista.

Función listar_Y_Contar_Elementos_En_Listas_V1

Descripción: Función que aplica una recursión lineal no por cola para contar los diferentes elementos de una lista y devolver otra construida de la forma que se indica a continuación:

'('(elemento-1 ctdad-1) '(elemento-2 ctdad-2) ... ... '(elemento-n ctdad-n))

Dominio: Elementos de la lista ingresada como parámetro.

INPUT

Lista cuyos elementos se desea contar.

Lista de retorno (Inicialmente vacia).

OUTPUT

Lista de elementos contados.

Función concatenar_Listas_De_Texto

Descripción: Función que recurre a una recursión del tipo lineal, no por cola, para concatenar todos los elementos del tipo string contenidos en una lista.

Dominio: Elementos de la lista ingresada como parámetro.

INPUT:

Lista cuyos elementos son todos string.

OUTPUT:

String con todos los elementos de la lista concatenados.

Función obtener_Maximo_Lista

Descripción: Función que utiliza una recursión del tipo lineal por cola para poder obtener el elemento (numérico) de mayor valor en una lista.

Dominio: Elementos de la lista ingresada como parámetro.

INPUT:

Lista de elementos numéricos.

Elemento de mayor valor (inicialmente el primer elemento de la lista).

OUTPUT:

Elemento de mayor valor.