**Pregunta 1**

**A)**

**TDAs**

**TDA mapa**

\*\*\*\*\*ASUMO QUE PARA LA CREACION DE ESTE TDA YA EXISTE UNA LISTA CON LOS DATOS DE CADA ESTACION\*\*\*\* Lista que almacena los TDAs estación, esta almacenará solo donde existan estaciones

Ejemplo: '((Estacion1) (Estacion2) (Estacion3)...)

**TDA estacion**

Contiene los datos registrados por cada las estaciones de monitoreo

Temperatura (entero) temepatura medida por la estacion

I (entero) Representa el valor de la fila donde se encuentra la estacion

J (entero) Representa el valor de la columna donde se encuentra la estacion

Color (string) Color de la estacion en el mapa

Ejemplo: '(25 1 1 "rojo")

**Funciones para TDA**

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* MAPA \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\*\*\*\* Constructor \*\*\*\*

;Asumiendo de que partida existe una lista con las estaciones del mapa (imagen), crea el TDA mapa con las estaciones existentes

;Recibe como argumento la lista inicial de estaciones (lista de listas)

;Retorna un TDA mapa`

;Llamada (crearMapa lista)

(define crearMapa (lambda (listaEstaciones)

(if (null? listaEstaciones)

'()

(if (equal? "blanco" (getColor(car listaEstaciones)))

(crearMapa (cdr listaEstaciones))

(cons (car listaEstaciones) (crearMapa (cdr listaEstaciones)))

)

)

)

)

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ESTACION \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\*\*\*\* GETTER \*\*\*\*

;Entrega la temperatura de la estacion

;Recibe como argumentos un TDA estacion (lista)

;Retorna la temperatura (entero) de la estación

;Llamada: (getTemperatura estacion1)

(define getTemperatura (lambda (estacion)

(car estacion)

)

)

;Entrega el color de la estacion

;Recibe como argumentos un TDA estacion (lista)

;Retorna el color (String) de la estación

;Llamada: (getColor estacion1)

(define getColor (lambda (estacion)

(car(cdr(cdr(cdr estacion))))

)

)

;Entrega la posicion en I de la estacion

;Recibe como argumentos un TDA estacion (lista)

;Retorna la posicion en I (entero) de la estación

;Llamada: (getI estacion1)

(define getI (lambda (estacion)

(car(cdr estacion))

)

)

;Entrega la posicion en J de la estacion

;Recibe como argumentos un TDA estacion (lista)

;Retorna la posicion en J (entero) de la estación

;Llamada: (getJ estacion1)

(define getJ (lambda (estacion)

(car(cdr(cdr estacion)))

)

)

**B)**

;Permite obtener aquellas estaciones que cumplen un creterio C

;Recibe como argumentos un TDA mapa (lista de estaciones) y el criterio C

;Retorna una nueva lista con los elementos que cumplen el criterio

;Llamada: (filtrarEstaciones mapa1 temperaturaMayor)

(define filtrarEstaciones (lambda (mapa criterio)

(if (null? mapa)

'()

(if (criterio (getTemperatura (car mapa)))

(cons (car mapa) (filtrarEstaciones (cdr mapa) criterio))

(filtrarEstaciones (cdr mapa) criterio)

)

)

)

)

**C)**

;Permite obtener aquellas estaciones que estan en la vecindad inmediata de cierto sector S

;Recibe como argumento una estacion y una lista de estaciones

;Retorna una nueva lista con los elementos que se encuentran en su vecinidad inmediata

Llamada: (vecinidad mapa1 estacion1)

(define vecinidad (lambda (lista estacion)

(if (null? lista)

'()

(if (and (= (getI estacion) (getI (car lista))) (= (getJ estacion) (getJ (car lista)))) ;Comprueba que la estacion no se este comparando con ella misma

(vecinidad (cdr lista) estacion)

(if (or

(= (getI estacion) (- (getI (car lista)) 1))

(= (getI estacion) (+ (getI (car lista)) 1))

(= (getJ estacion) (- (getJ (car lista)) 1))

(= (getJ estacion) (- (getJ (car lista)) 1))

(and (= (getI (car lista)) (- (getI (car lista)) 1)) (= (getI (car lista)) (- (getI (car lista)) 1)))

(and (= (getI estacion) (+ (getI (car lista)) 1)) (= (getI estacion) (+ (getI (car lista)) 1)))

(and (= (getI estacion) (+ (getI (car lista)) 1)) (= (getI estacion) (- (getI (car lista)) 1)))

(and (= (getI estacion) (- (getI (car lista)) 1)) (= (getI estacion) (+ (getI (car lista)) 1)))

)

(cons (car lista) (vecinidad (cdr lista) estacion))

(vecinidad (cdr lista) estacion)

)

)

)

)

)

**D)**

;Permite calcular la temperatura de una vecinidad

;Recibe como argumentos un mapa (lista de listas), una estacion centro de la vecindad (lista) y una funcion estadistica

;Retorna la temperatura estimada de la vecindad (entero)

;Llamada: (estimar estación moda)

(define estimar (lambda (lista estacion funcion)

(funcion (vecinidad lista estacion))

)

)

**Pregunta numero 2**

**TDAs**

**A)**

**TDA Usuarios**

Lista de TDAs Usuario

Ejemplo: '((Usuario1) (Usuario2) ...)

**TDA Usuario**

Lista que contiene los siguiente

username(string)

pass("string")

Cartera (entero)

Preferencias(lista de strings)

peliculasAdquiridas (lista de TDAs Pelicula)

Rut(string)

Ejemplo: '("Pepe" "1234" 666 ("romance" "drama") (Pelicula1 Pelicula2) "182345438")

**TDA Peliculas**

Lista de TDAs Peliculas

Ejemplo: '((Pelicula1) (Pelicula2))

**TDA Pelicula**

Lista que contiene los siguentes elementos

id(entero)

nombre (string)

costo(entero)

duracion (entero)

anio(entero)

tags(lista de strings)

Ejemplo: '(1 "Forrest Gump" 1700 142 1994 ("romance" "drama"))

**Funciones para los TDAs**

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* USUARIOS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\*\*\*\* CONSTRUCTOR \*\*\*\*

;Crea un TDA usuarios

;No recibe argumentos

;Retorna un TDA usuarios

;Llamada: (crearUsuarios)

(define (crearUsuarios)

(list )

)

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* USUARIO \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\*\*\*\* CONSTRUCTOR \*\*\*\*

;Crea un usuario

;Recibe como argumentos el nombre de usuario(string), contraseña (string), cartera(entero), prefetencias(list), peliculas(list) y rut(string)

;Retorna un TDA usuario

;Llamada: (crearUsuario Christofer “123” 640 ‘(“romance”) (“Leon”) “202397867”)

(define crearUsuario (lambda (user pass cartera preferencias peliculas rut)

(list user pass cartera preferencias peliculas rut)

)

)

;\*\*\*\* Getter \*\*\*\*

;Entrega un usuario de una lista de usarios

;Recibe como argumentos un TDA Usuarios (lista de usuario)

;Retorna un usuario

;Llamada: (getUsuario listaUsuarios1)

(define getUsuario (lambda (lista)

(car lista)

)

)

;Entrega el nombre que tiene un usuario

;Recibe como argumentos un TDA Usuario

;Retorna el nombre que tiene un usuario

;Llamada: (getNombre usuario1)

(define getNombre (lambda (usuario)

(car usuario)

)

)

;Entrega la contraseña que tiene un usuario

;Recibe como argumentos un TDA Usuario

;Retorna la contraseña que tiene un usuario

;Llamada: (getPass usuario1)

(define getPass (lambda (usuario)

(car(cdr usuario))

)

)

;Entrega el dinero que tiene un usuario

;Recibe como argumentos un TDA Usuario

;Retorna el nombre que tiene un usuario

;Llamada: (getDinero usuario1)

(define getDinero (lambda (usuario)

(car (cdr (cdr usuario)))

)

)

;Entrega las preferencias que tiene un usuario

;Recibe como argumentos un TDA Usuario

;Retorna las preferencias que tiene un usuario

;Llamada: (getPreferencias usuario1)

(define getPreferencias (lambda (usuario)

(car(cdr (cdr (cdr usuario))))

)

)

;Entrega las peliculas adquiridas que tiene un usuario

;Recibe como argumentos un TDA Usuario

;Retorna las pelicualas adquiridas que tiene un usuario

;Llamada: (getPeliculasAdquiridas usuario1)

(define getPeliculasAdquiridas (lambda (usuario)

(car (cdr (cdr (cdr (cdr usuario)))))

)

)

;Entrega el rut que tiene un usuario

;Recibe como argumentos un TDA Usuario

;Retorna el rut que tiene un usuario

;Llamada: (getRut usuario1)

(define getRut (lambda (usuario)

(car (cdr (cdr (cdr (cdr usuario)))))

)

)

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* PELICULAS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\*\*\*\* CONSTRUCTOR \*\*\*\*

;Crea un TDA peliculas

;No recibe argumentos

;Retorna un TDA películas

;Llamada: (crearPeliculas)

(define (crearPeliculas)

(list )

)

;\*\*\*\* Getter \*\*\*\*

;Entrega una pelicula de una lista de pelicula

;Recibe como argumentos un TDA Peliculas

;Retorna la pelicula de una lista de película

;Llamada: (getPelicula listaPeliculas1)

(define getPelicula (lambda (peliculas)

(car peliculas)

)

)

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* PELICULA \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\*\*\*\* CONSTRUCTOR \*\*\*\*

;Crea un TDA Pelicula

;Recibe como argumentos el id(entero), nombre (string), duracion (entero), anio(entero), tags(lista de strings) y costo(entero)

;Retorna un TDA película

;Llamada: (crearPelicula 32 “Leon” 2300 110 1994 ‘(“accion” “drama”))

(define crearPelicula(lambda (id nombre costo duracion anio tags)

(list id nombre duracion anio tags costo)

)

)

;\*\*\*\* Getter \*\*\*\*

;Entrega el precio de una pelicula

;Recibe como argumentos un TDA Pelicula

;Retorna el precio de una película

;Llamada: (getPrecio pelicula1)

(define getPrecio (lambda (pelicula)

(caddr (cdddr pelicula))

)

)

;Entrega el tiempo que dura una pelicula

;Recibe como argumentos un TDA Pelicula

;Retorna el tiempo que dura una película

;Llamada: (getDuracion pelicula1)

(define getDuracion (lambda (pelicula)

(car (cdr (cdr (cdr pelicula))))

)

)

**Respuestas**

**B)**

;Registra a un nuevo usuario

;Recibe como argumentos el nombre de usuario(string), la contraseña(string), RacketFlix (lista de TDAs)

;Retorna la lista de usuarios con el usuario ya registrado

;Llamada: (registrar “Christofer” “123456” listaUsuarios1)

(define registrar (lambda (user pass usuarios)

(if (null? usuarios)

(list (crearUsuario user pass))

(cons (getUsuario) (crearUsuario user pass))

)

)

)

**C)**

;Permite que un usuario compre una pelicula

;Recibe como argumentos un usuario y la pelicula a comprar

;En caso de que se pueda comprar retorna al usuario con la pelicula ya comprada y false en caso de que no se pueda comprar

;Llamada: (comprar pelicula1 usuario1)

(define comprar (lambda (pelicula usuario)

(if (< (getDinero usuario) (getPrecio pelicula))

#f

(crearUsuario (getNombre) (getPass) (- (getDinero usuario) (getPrecio pelicula)) (getPreferencias) (cons (getPeliculasAdquiridas) pelicula) (getRut))

)

)

)

**D)**

;Permite filtrar peliculas de acuerdo a un cierto criterio

;Recibe como argumento la lista de peliculas y una función para filtrar

;Retorna una nueva lista con los elementos que cumplen el criterio

;Llamado: (filtrarPeliculas listaPeliculas2 peliculasAccion)

(define filtrarPeliculas (lambda (listaPeliculas criterio)

(if (null? listaPeliculas)

'()

(if (criterio (car listaPeliculas))

(cons (car listaPeliculas) (filtrarPeliculas (cdr listaPeliculas) criterio))

(filtrarPeliculas (cdr listaPeliculas) criterio)

)

)

)

)

**E) i)**

;Filtra las peliculas que se encuentren dentro de un rago de precio

;Recibe como argumentos la lista de peliculas, el precio minimo y el precio maximo

;Retorna una nueva lista con los elementos que cumplen con el precio

;Llamado: (rangoDuracion listaPeliculas1 60 180)

(define rangoDuracion (lambda (listaPeliculas min max)

(if (null? listaPeliculas)

'()

(if (and (< (getDuracion (car listaPeliculas)) max) (> (getDuracion (car listaPeliculas)) min))

(cons (car listaPeliculas) (rangoDuracion (cdr listaPeliculas) min max))

(rangoDuracion (cdr listaPeliculas) min max)

)

)

)

)

**F)**

;Implementación de la funcion map, aplica la función entregada a las peliculas de una lista de peliculas

;Recibe como argumento una funcion y una lista de peliculas

;Retorna una nueva lista de peliculas con cada pelicula modificada

;Esta función es del tipo de recursion natural

;Llamado: (mapPeliculas aumentarPrecio listaPeliculas1)

(define mapPeliculas (lambda (funcion listaPeliculas)

;caso base

(if (null? listaPeliculas) ;Condición de borde, indica que la lista es nula o que se llego al final del la lista

null

(cons (funcion (car listaPeliculas)) (mapPeliculas funcion (cdr listaPeliculas))) ;Descomposición recursiva para aplicar función a cada elemento de la cola

)

)

)

;Aplica una rebaja a las peliculas que cumplan un cierto criterio

;Recibe como argumentos la lista de peliculas, la funcion de rebaja y el criterio

;Retorna una nueva lista de pelicualas con los precios actualizados

;Llamado: (rebaja listaPeliculas1 mitadPrecio antiguas)

(define rebaja (lambda (listaPeliculas funcion criterio)

(if (null? listaPeliculas)

'()

(if (criterio (car listaPeliculas))

(cons (mapPeliculas (car listaPeliculas)) (filtrarPeliculas (cdr listaPeliculas) criterio))

(filtrarPeliculas (cdr listaPeliculas) criterio)

)

)

)

)