Trabajo Práctico 3. Programación Funcional

Puntuación

Puntaje Total: 100 puntos

Aprobación: 60 puntos

Fecha de entrega: **14/12/2022 - 23:55 hs.**

Condiciones de entrega

1. El presente trabajo práctico deberá resolverse en grupo de hasta 3 (tres) personas.
2. Entrega: Se realizará por medio del Campus Virtual de la UTN, en la tarea correspondiente al TP 3. La extensión del archivo será .zip o .rar, de acuerdo al programa de compresión usado. El nombre del archivo se consigue concatenando un prefijo del número del TP con los apellidos de los integrantes separados por guiones (Ej: Pérez y Abdala, el nombre será `tp3-abdala-perez.zip`). Note que no hay espacios en blanco ni acentos en el nombre de archivo. Dentro del archivo de entrega, deben constar los siguientes:
    - Archivo con el código fuente Scheme denominado `TP3.rkt` debidamente comentado.
    - Los casos de prueba se entregarán en un archivo de texto, no deben ser capturas de pantalla. Deberán cubrir diferentes resultados que puedan obtenerse de la evaluación de las funciones solicitadas. Se enfatiza que se adjunten casos de prueba que sean claros, válidos y suficientes para poder verificar el trabajo. Entregar este archivo con el nombre `casos-de-prueba.txt`.
    - Archivo de texto (`integrantes.txt`) con una línea para cada integrante, en la cual figure el nombre del alumno/a y su dirección de email.
3. Penalización por entrega fuera de término: Si el trabajo práctico se entrega después de la fecha indicada, y hasta una semana tarde, tendrá una quita de 15 puntos. No serán recibidos trabajos luego de una semana de la fecha de entrega. Los trabajos que se deban rehacer/corregir fuera de la fecha de entrega tienen una quita de 30 puntos.

Enunciado

Se solicita que escriba las funciones que permiten gestionar calendarios y citas según la lista que sigue a continuación. La biblioteca de gestión de calendarios tiene tres tipos de elementos: calendarios, citas, y momentos en el tiempo.

Usted puede seleccionar la representación interna de cada elemento, sujeto a que las siguientes funciones se definen y trabajan como en los siguientes ejemplos:

`(momento-nuevo un-dia un-mes un-año una-hora un-minuto)`
Retorna un momento. Todos los argumentos son números enteros.
Puede asumir que los datos corresponden a un momento válido. Por ejemplo, asuma que no se ingresará como fecha el día 31 y el mes 11 (noviembre), que la hora será un valor entre 0 y 23, los minutos entre 0 y 59, que los días 29 de febrero sólo serán ingresados para años bisiestos, etc.

```
(cita-nueva
      un-texto-descriptivo ; es un string
      un-momento-de-inicio  ; es un momento
      un-momento-de-finalización) ; es un momento
```
Retorna una cita.

`(citas-solapadas? una-cita otra-cita)`
Retorna verdadero si las citas se solapan en el tiempo, y falso en caso contrario. Note que pueden solaparse de diversas maneras. Si una cita inicia en el mismo momento que la otra finaliza, entonces no solapan.

`(citas-iguales? una-cita otra-cita)`
Retorna verdadero si las citas tienen la misma descripción y los mismos momentos de inicio y fin.

`(calendario-nuevo un-nombre)`
Retorna un calendario sin citas. El nombre del calendario es un string.

`(calendario-agrega-cita un-calendario una-cita)`
Retorna un nuevo calendario, que tiene **`una-cita`** además de las existentes en **`un-calendario`**.

`(calendario-elimina-cita un-calendario una-cita)`
Retorna un calendario, en el cual no existe **`una-cita`** pero están las otras existentes en **`un-calendario`**.

`(calendarios-solapados? un-calendario otro-calendario)`

Retorna verdadero si los calendarios tienen al menos una cita solapada en el tiempo, y falso en caso contrario.

`(encuentra-cita-mas-temprana un-calendario un-pred)`
Encuentra y devuelve la primera cita —en orden de fechas, la más temprana— que hay en el calendario **`un-calendario`** y que hace verdadera la evaluación de **`un-pred`**. La función **`un-pred`** toma como argumentos el calendario **`un-calendario`** y una cita que pertenece al calendario. Escriba **`encuentra-cita-mas-tardia`** de manera que recorra las citas del calendario y las pase a **`un-pred`** una por una. Si no existe ninguna cita en el calendario que cumpla lo solicitado, devuelva **`#f`**.

`(encuentra-cita-mas-tardia un-calendario un-pred)`
Similar a la función anterior, pero encuentra y devuelve la última —en orden de fechas, la más tardía— o, si no hay ninguna que satisfaga **`un-pred`**, devuelve **`#f`**.

Todas las funciones anteriores deben escribirse como “funciones puras”, o sea que no tienen efectos laterales: no escriben en variables globales, no escriben en pantalla, no toman entrada del teclado.

La siguiente función que debe escribir, usa a la primitiva **`display`** para escribir en la salida estándar (pantalla):

`(calendario-muestra un-calendario)`
Retorna el calendario que va de argumento. Tiene el efecto de escribir en la pantalla el calendario con el siguiente formato: el nombre del calendario en un renglón y a continuación una cita por renglón, ordenadas desde las más tempranas a las más tardías. Cada cita consta de tres partes: el momento de inicio, el momento final y el texto descriptivo. Las fechas que corresponden a los momentos se deben mostrar en el formato ISO8601 que adopta la forma: Año-Mes-DiaTHora:Minutos, ej: 2022-10-18T15:23.

Las funciones auxiliares siguientes le ayudarán a escribir la anterior:
`(cita-muestra una-cita)` para mostrar una cita. No es una función pura pues escribe en la pantalla.

`(momento->string-iso8601 un-momento)` que toma la estructura de momento (fecha) interna del TP y devuelve un string en formato ISO8601. Es una función pura.

Ejemplo de uso

Se bosqueja a continuación un ejemplo de las clases de consultas que se podrían escribir con estas primitivas que usted definió de acuerdo al enunciado.

Este ejemplo no está completo, si desea hacerlo funcionar, deberá escribir las funciones faltantes que se indican en el texto. No es necesario resolver el ejemplo para entregar el TP. La resolución del ejemplo no tiene ningún puntaje.

Supongamos que necesitamos averiguar cuál es la siguiente cita de ir al gimnasio que sea posterior a la cita de ir al odontólogo, la semana que viene.

Vamos a usar la evaluación de **(encuentra-cita-mas-temprana un-calendario un-pred):**

```
(define (gimnasio-despues-odontologo-semana-que-viene cal cita)
  (encuentra-cita-mas-temprana
   (calendario-con-citas-semana-que-viene cal)
    cita-gimnasio-despues-odontologo?))
```

Y las funciones auxiliares serán:

```
(define (cita-gimnasio-despues-odontologo? cal cita)
  (and
    (es-cita-gimnasio? cita)
    (hay-cita-odontologo-antes? cal cita)))

(define (hay-cita-odontologo-antes? cal cita)
  (let ((odonto (encuentra-cita-mas-tardia cal es-cita-odontologo?)))
    (and odonto (cita-antecede-a? odonto cita))))
```

Nota: se deja la definición de **cita-antecede-a?** al lector. Devuelve verdadero cuando el momento de la cita que es el primer argumento es previo al momento de la cita que es el segundo argumento.

Nota: queda la definición de **es-cita-odontologo?** para que el lector la construya, similar a **es-cita-gimnasio?** Uno debe buscar en el texto descriptivo de la cita la palabra “odontologo” y “gimnasio”, respectivamente. Más abajo se presenta un boceto de **es-cita-gimnasio?**

```
(define (es-cita-gimnasio? cita)
  (descripcion-cita-contiene? cita “gimnasio”))
```

La más complicada, usando las primitivas de crear un calendario nuevo y buscar/eliminar cita de un calendario, se filtran recursivamente las citas para construir un calendario con sólo las citas de la semana que viene.

Note que el calendario no es una lista de citas solamente.

```
(define (calendario-con-citas-semana-que-viene cal)
  (let ((una-cita
         (encuentra-cita-mas-tardia
          cal
          (lambda (ca ci) (cita-pertenece-a-semana-que-viene? ci)))))
    (if una-cita
        (calendario-agrega-cita
         (calendario-con-citas-semana-que-viene
          (calendario-elimina-cita una-cita) una-cita))
        (calendario-nuevo (calendario-nombre cal)))))
```

Aquí hemos buscado una cita, la más tardía de la semana que viene. Si esa cita existe, entonces la agrego al calendario que se construye recursivamente; si no puedo conseguir una cita de la semana que viene, entonces devuelvo un calendario nuevo (vacío) que tiene el mismo nombre que el calendario que fue pasado como argumento. El resultado será un calendario del mismo nombre, pero que solamente tiene las citas del viejo calendario que corresponden a la semana que viene.

Nota: se deja la definición de **`calendario-nombre`** al lector, debe retornar el nombre del calendario argumento.

Nota: se deja la definición de **`descripcion-cita-contiene?`** al lector. Puede ser tan sencillo como una comparación de igualdad entre la descripción de la cita y el argumento string..

Nota: se deja la definición de **`cita-pertenece-a-semana-que-viene?`** al lector. Para la cita argumento se debe verificar el momento de la misma, si se encuentra dentro del rango de momentos de la semana que viene. Para simplificar, puede usar valores fijos para representar el comienzo y final de la semana que viene.