PEP 1 Paradigmas de Programación (Abril 2022)

NOMBRE: **PROFESOR:** **RUT:**

Instrucciones: En **AMBAS PREGUNTAS** **(1)** hacer un uso adecuado del paradigma funcional (según corresponde use funciones anónimas, orden superior, currificación, programación declarativa, etc.); **(2)** use pseudo-scheme **(3)** documentar; **(4)** Responder preguntas en hojas separadas indicando en cada una (preguntas y respuestas) su nombre, rut y profesor; **(5)** entregar todas las hojas (preguntas y respuestas) al final; **(6)** no es necesario comprobar tipos; y **(7)** solo puede usar car, cdr, if, cond, define, let, null?, =, >, <, eqv?, eq? equal?, +, *, /, -, null, lambda, not, and, or, cons, list, length, map, filter, apply (agrega lista en base a operación(ej: (apply + '(1 2 3)) => 6), string-length (largo string). Cualquier otra función no indicada en esta lista deberá ser implementada.

Pregunta 1 (30 pts):

Considere la especificación e implementación parcial del TDA documento, el cual forma parte de un sistema de ofimática como Word o GoogleDocs.

a. (1.5 pts) ;retorna la primera palabra de un documento.
dom: document ; rec: word
firstWord(document):word

b. (1.5 pts) ;retorna un documento con las siguientes palabras después de la primera.
dom: document ; rec: document
nextWords(document):document

c. (5 pts) ;localiza las posiciones donde se ubica una palabra en el documento. El retorno corresponde a una lista con todas las posiciones donde figura la palabra.
dom: document X word ; rec: position List
find(document,word):position List

d. (5 pts) ;reemplaza todas las ocurrencias de la palabra especificada en word, por newWord. El retorno es un nuevo documento con los cambios efectuados.
dom: document X word X word, rec: document
replaceAll(document,word,newWord):document

e. (7 pts) ;contabiliza la cantidad de caracteres en el documento sin considerar los espacios (se asume que entre palabras existen espacios simples).
dom: document ; rec:integer+{0}
charactersCountWithoutSpace(document):integer+{0}

f. (10 pts) ;busca frases (ej: "esta es una frase") en un documento. El retorno es una lista con las posiciones en la lista donde comienzan las frases que coinciden.
dom:document X string , rec: position List
findPhrase(document,phrase):position List

Respecto de la implementación parcial del TDA documento, se conoce su representación.

documento = null | string X documento

y el constructor base llamado buildDoc, tiene dom: string y rec: string list. Su uso queda expresado de la siguiente forma: (buildDoc "hola mundo") => ("hola" " " "mundo").

A partir de los antecedentes provistos, implemente cada una de las operaciones del TDA señaladas en los ítems (a) al (f).

PEP 1 Paradigmas de Programación (Abril 2022)

NOMBRE: **PROFESOR:** **RUT:**

Pregunta 2 (30 pts):

Se planea construir un sistema para un servicio de transporte urbano (como Uber o Lyft) que permita vincular usuarios clientes con usuarios conductores a través de viajes. En particular, un **usuario cliente** registrado con un correo electrónico puede solicitar un **viaje** ingresando una dirección origen y una dirección destino. Respecto del **viaje**, este consta de un identificador, el correo electrónico del conductor que acepta el viaje, el correo del cliente que solicita el viaje, las direcciones de origen y destino, distancia total a recorrer representada en kilómetros (km), tarifa por km recorrido, estado del viaje (ej: asignado, rechazado, terminado), duración del viaje en minutos y finalmente el precio a pagar CLP.

En relación al **usuario conductor,** éste está registrado con un correo electrónico y cuenta con un registro histórico de todos los viajes que ha realizado y aquellos que se le han asignado (no necesariamente concretado). Estos últimos se van actualizando conforme el conductor acepta o rechaza los viajes que se le asignen.

A partir de estos antecedentes:

a) (**3 pts**) Especificar el o los TDA(s) para abordar la construcción del sistema. En la especificación de cada TDA (listado de operaciones), sólo considere lo absolutamente necesario para cubrir los requerimientos de esta pregunta.
b) (**6 pts**) Implementar el o los TDA(s) identificados. Procure señalar la representación escogida para todos los TDAs. Luego implementación las funciones expresadas en (a) (Solo aquellas que es absolutamente necesario para responder a los siguientes ítems).
c) (**2 pts**) Implementar una función que permita obtener el total de km recorridos de un conductor considerando solo los viajes realizados.
d) (**4 pts**) Implementar una función que permita obtener el precio total de un viaje. Esto calculado a partir de los kilómetros recorridos y la tarifa.
e) (**10 pts**) Implementar una función usando explícitamente recursión natural (no puede usar funciones de scheme que realicen esta operación de manera directa) que permita a un cliente determinar el o los viajes que reúnan ciertas condiciones (ej: el más largo que X, el más costoso que X, el más barato que X, etc.) El criterio se define como un criterio de entrada de la función.
f) **(5 pts)** Cree una función currificada que permita determinar si el precio de un viaje es mayor que un valor dado. Luego ejemplifique el uso de esta función en conjunto con la función implementada en (e) para obtener los viajes de un cliente C1 (se asume su existencia) cuyo valor sea mayor a $20.000.

Para autoevaluación recuerde evaluar cada ítem de acuerdo a la siguiente escala de apreciación:

Ptje	Descripción
0	No responde pregunta. Respuesta no corresponde con la pregunta.
.0.25	Respuesta incompleta con errores mayores pero que revela cierta orientación hacia la solución del problema.
0.5	Respuesta incompleta con errores mayores. Demuestra entendimiento del problema, sin embargo la aplicación del paradigma y el lenguaje es imprecisa o incorrecta. Ej: omite casos base, descomposición recursiva, no aplica lo solicitado para la pregunta, etc.
0.75	Respuesta completa con errores menores. Si bien la respuesta es completa, presenta algunas imprecisiones. Demuestra un correcto uso del paradigma y lenguaje.
1	Respuesta completa sin errores

Luego la nota se calcula de la siguiente forma:

$$Nota = 1 + (\sum_{i} Pi * PtjeTotalITEMi)/10 \quad ; Pi\ es\ el\ ptje\ asignado\ en\ la\ evaluación\ para\ el\ item\ i$$

Pauta Pregunta 1

a. (1.5 pts) ;retorna la primera palabra de un documento
dom: document, rec: word
(define firstWord car)

b. (1.5 pts) ;retorna un documento con las siguientes palabras después de la primera
dom: document, rec: document
(define nextWords cdr)

c. (5 pts) ;permite localizar las posiciones donde se ubica una palabra en el documento. El retorno corresponde a una lista con todas las posiciones donde figura la palabra
dom: document X word, rec: position List
recursión natural

```
(define find (lambda (document word)
               (define findAux (lambda (document word pos)
                    (if (null? document)
                        null
                        (if (equal? word (firstWord document))
                            (cons pos (findAux (nextWords document) word (+ pos 1)))
                            (findAux (nextWords document) word (+ pos 1)))))
               (findAux document word 0))
```

d. (5 pts) ;reemplaza todas las ocurrencias de la palabra especificada en word, por newWord. El retorno es un nuevo documento con los cambios efectuados.
dom: document X word X word, rec: document
(define replaceAll (lambda (document word newWord)

```
            (map (lambda (w) (if (equal? word w) newWord w)) document)))
```

e. (7 pts) ;función que permite contabilizar la cantidad de caracteres en el documento sin considerar los espacios (se asume que entre palabras existen espacios simples).
dom: document rec:integer+{0}

```
(define charactersCountWithoutSpace (lambda (document)
        (apply + (map (lambda (w) (if (not (equal? w " ")) (string-length w) 0)) document))))
```

f. (10 pts) ;función que permite buscar frases (ej: “esta es una frase”) en un documento. El retorno es una lista con las posiciones en la lista donde comienzan las frases que coinciden.
dom:document X string , rec: position LIst

```
;recursión natural
(define findPhrase (lambda (document phrase)
        (define findPhraseAux (lambda (document phrase pos)
                (if (null? document)
                    null
                 (if (phrase? phrase document #t)
                    (cons pos (findPhraseAux (nextWords document) phrase (+ pos 1)))
                        (findPhraseAux (nextWords document) phrase (+ pos 1))))))
      (findPhraseAux document (buildDoc phrase) 0)))

;recursión cola
(define phrase? (lambda (phrase document result)
                (if (null? phrase)
                   result
                   (if (null? document)
                      #f
                      (if (equal? (firstWord phrase) (firstWord document))
                          (phrase? (nextWords phrase) (nextWords document) #t)
                          (phrase? null null  #f))))))
```

Pauta Pregunta 2

a) (**3 pts**) Especificar el o los TDA(s) para abordar la construcción del sistema. En la especificación de cada TDA (listado de operaciones), sólo considere lo absolutamente necesario para cubrir los requerimientos de esta pregunta.

TDA sistema
TDA cliente
TDA clientes
TDA conductor
- selectores: getViajes(conductor):viajes
- otras: filterViajesCliente(criterio,cliente)

TDA conductores
TDA viaje
- selectores: getKmRecorridos(viaje):kms
- getTarifa(viaje):tarifa

TDA viajes
- Selectores: firstTrip, nextTrips, getEstado

b) (**6 pts**) Implementar el o los TDA(s) identificados. Procure señalar la representación escogida para todos los TDAs. A continuación, implementar las funciones constructoras, selectoras, modificadores, pertenencia que haya expresado como necesarias en el item (a) de esta pregunta

TDA sistema. Representación: Clientes X Conductores
TDA cliente: Representación Correo (String) X Viajes
TDA conductor: Representación: Correo (String) X Viajes
-

TDA viaje: Representación: id (Integer) X Cliente X Conductor X EstadoAceptación (boolean) X Origen (String) X Destino (String) X DistanciaTotal (Integer) X Tarifa (Integer) X Duración (Integer) X Precio (Integer)
- Selector (define getKmRecorridos caddddddddr) ; o su correspondiente con (car (cdr (cdr (cdr ….)
- Selector (define getTarifa cadddddddddr) ;o su correspondiente con (car (cdr …..)
- Selector (define getEstado cadddr) ; o su correspondiente
- Otras: precioTotalViaje (ver implementación en j)

TDA viajes: null | viaje X viajes
-Selectores
(define firstTrip car)
(define nextTrips cdr)

TDA clientes: null | cliente X clientes
TDA conductores: null | conductor X conductores|

c) (**4 pts**) Implementar una función que permita obtener el total de km recorridos de un conductor considerando solo los viajes realizados.

```
;selector   ;dom: viaje   ; rec: km (integer)
(define getTotalKmRecorridos (lambda (conductor)
            (let ([criterio (lambda (v) (equal? (getEstado v) “terminado”))])
                  (apply + (map getKmRecorridos  (filter criterio (getViajes conductor)))))))
```

d) (**2 pts**) Implementar una función que permita obtener el precio total de un viaje. Esto calculado a partir de los kilómetros recorridos y la tarifa.

```
(define precioTotalViaje (lambda (viaje) (* (getKmRecorridos viaje) (getTarifa viaje)))
```

e) (**5 pts**) Implementar una función usando explícitamente recursión natural que permita a un cliente determinar el o los viajes que reúnan ciertas condiciones (ej: el más largo que X, el más costoso que X, el más barato que X, etc.) El criterio se define como un criterio de entrada de la función.

```
(define filterViajesCliente (lambda criterio cliente)
                  (define filterViajesAux (lambda criterio viajes)
                              (if (null? viajes)
                                    null
                                    (if (criterio (firstTrip viajes)
                                          (cons (firstTrip viajes) (filterViajesAux criterio (nextTrips viajes)))
                                          (filterViajesAux criterio (nextTrips viajes)))))
                  (filterViajesAux criterio (getViajes cliente)))
```

f) **(2 pts)** Cree una función currificada que permita determinar si el precio de un viaje es mayor que un valor dado. Luego ejemplifique el uso de esta función en conjunto con la función implementada en (e) para obtener los viajes de un cliente C1 (se asume su existencia) cuyo valor sea mayor a $20.000.

```
(define viajesGreaterThan (lambda (threshold)
            (lambda (viaje)
                  (> (precioTotalViaje viaje) threshold))))

            (filterViajesCliente   (viajesGreaterThan 20000)     C1)
```