PautaAuxiliar8

November 13, 2020

1 Auxiliar 8 - Mutación y Aliasing

1.0.1 P1. ONU

La ONU requiere su ayuda manteniendo un registro actualizado de los mandatarios de todos los países del mundo. Para esto desarrollaremos una función que permita cambiar el nombre del presidente de un país.

1.0.2 A)

Cree la estructura **no mutable** llamada 'Pais' que tenga los atributos nombre, continente y presidente_actual.

[ ]:
```
import estructura
# Pais: nombre(str) continente(str) nombre_presidente(str)
estructura.crear("Pais", "nombre continente nombre_presidente")
```

1.0.3 B)

Cree la estructura **mutable** PaisMut con los atributos nombre, continente y presidente_actual.

[ ]:
```
# PaisMut: nombre(string) continente(string) nombre_presidente(str)
estructura.mutable("PaisMut", "nombre continente nombre_presidente")
```

1.0.4 C)

Defina la función cambioDeMando, que recibe un **Pais** y el nombre de un presidente, y retorne un Pais con el nuevo nombre del presidente.

[ ]:
```
#cambioDeMando: Pais, str -> Pais
#Recibe un Pais (pais) y el nombre del nuevo presidente (n),
# y retorna un Pais que cambia el nombre del presidente de pais por el dado␣
 ↪como parametro n.
#Ej: cambioDeMando(Pais("EEUU", "America", "Donald Trump "), "Joe Biden")␣
 ↪entrega Pais("EEUU", "America", "Joe Biden")
def cambioDeMando(pais, n):
  return Pais(pais.nombre, pais.continente, n)
#test
```

```
assert cambioDeMando(Pais("EEUU", "America", "Donald Trump "), "Joe Biden") ==␣
 ↪("EEUU", "America", "Joe Biden")
```

1.0.5 D)

Defina la función cambioDeMandoMut, que recibe un **PaisMut** y el nombre de un presidente y actualiza el nombre del presidente de PaisMut en cuestión. Esta función debe retornar None.

[ ]:
```
#cambioDeMandoMut: PaisMut, str -> None
#efecto: Cambia el atributo nombre_presidente de pais por n
#Ej: cambioDeMandoMut(Pais("EEUU", "America", "Donald Trump "), "Joe Biden")␣
 ↪muta pais a Pais("EEUU", "America", "Joe Biden")
def cambioDeMandoMut(pais, n):
  pais.nombre_presidente = n

#test
pais = PaisMut("EEUU", "America", "Donald Trump ")
cambioDeMandoMut(pais,"Joe Biden")
assert pais == PaisMut("EEUU", "America", "Joe Biden")
```

1.0.6 P2. Convivencia

Cuando acabe la pandemia organizaremos una convivencia del curso. Para esto crearemos un programa que nos permita llevar un registro de los participantes y las cosas que traerá cada uno.

Para resolver el ejercicio, utilice la funcion buscar y las estructuras mutables lista, producto y participante definidas a continuación.

[ ]:
```
import estructura

# participante: nombre(str) confirmacion(str) nombre_producto(str)
estructura.mutable('participante', 'nombre confirmacion nombre_producto')
# producto: nombre(str) cantidad(int)
estructura.mutable('producto', 'nombre cantidad')
# lista: valor(producto o participante) siguiente(lista)
estructura.mutable('lista', 'valor siguiente')

# buscar: lista(producto o participante ), string -> producto o participante
# Recibe una lista de las estructuras producto o participante y un parámetro␣
 ↪nombre.
# Busca el elemento de la lista cuyo atributo nombre sea igual al parámetro␣
 ↪nombre.
# Ej:
# producto1 = producto("chips pop", 5)
# producto2 = producto("bebidas", 0)
# L1 = lista(producto1,(lista(producto2, None)))
# buscar(L1, "chips pop")
# assert buscar(L1, "chips pop") == producto1
```

```
# participante1 = participante("Camila", "no", None)
# participante2= participante("José", "no", None)
# L2 = lista(participante1,lista(participante2, None))
# assert buscar(L2, "José") == participante2

def buscar(L, nombre, value=None):
  if L== None:
    return None
  if L.valor.nombre==nombre:
    value= L.valor
    return value
  else:
    return buscar(L.siguiente, nombre)


producto1 = producto("chips pop", 5)
producto2 = producto("bebidas", 0)
L1 = lista(producto1,(lista(producto2, None)))
buscar(L1, "chips pop")
assert buscar(L1, "chips pop") == producto1

participante1 = participante("Camila", "no", None)
participante2= participante("José", "no", None)
L2 = lista(participante1,lista(participante2, None))
assert buscar(L2, "José") == participante2
```

1.0.7 A)

Defina e inicialice con None las variables de estado participantes e inventario. La variable participantes será una lista de elementos de tipo participante, y nos permitirá llevar un registro de quienes irán o no a la convivencia. Por otro lado, inventario será una lista de elementos de tipo producto en la que iremos agregando las donanciones de cada participante.

[ ]:
```
# Variables de estado :
# participantes: lista(participante)
# para guardar a los participantes de la convivencia guardando
# su nombre, el estado de su participación y el nombre del producto que traerán

# participantes: lista(producto)
# para guardar los productos donados a la convivencia
# indicando el nombre de estos y la cantidad que se ha recolectado

participantes=None
inventario=None
```

1.0.8 B)

Defina la función editarConfirmacionParticipantes. Esta recibirá el nombre de la persona (nombre) y la decisión de si irá o no a la convivencia (confirmacion). Esta función debe editar el atributo confirmacion de la persona con el nombre dado. Además, si la persona no estaba registrada, se agrega a la lista de participantes con el estado indicado en confirmacion.

Esta función no debe retornar nada. Pero debe imprimir el valor anterior y el nuevo de confirmacion. Para el caso en el que se agregue un nuevo participante se debe imprimir el nombre de quien se agregó y el valor actual de su confirmacion.

Ejemplos de mensajes:

i) Caso en el que existe el participante:

```
La confirmación cambió, antes era no
ahora es sí
```

ii) Caso en el que se agrega a alguien:

```
Se agregó a Juanita con confirmación sí
```

Utilice la función buscar y la variable global participantes. Además asuma que los nombres de los asistentes son únicos.

[ ]:
```
# editarConfirmacionParticipantes: str, str -> None
# efecto: cambia el atributo confirmacion del elemento de la lista␣
 ↪participantes cuyo nombre es dado.
# Y en el caso en el que nadie tenga el nombre dado, se agrega un nuevo␣
 ↪participante al inicio de la lista participantes
# con el nombre y la confirmacion dados.

def editarConfirmacionParticipantes(nombre, confirmacion):
  global participantes
  p=buscar(participantes, nombre)
  if p!=None:
    print("La confirmación cambió, antes era ", p.confirmacion )
    p.confirmacion=confirmacion
    print("ahora es ", p.confirmacion )
  else:
    p=participante(nombre, confirmacion, None)
    participantes= lista(p, participantes)
    print("Se agregó a ", p.nombre, "con confirmación ", p.confirmacion)
#test  (participantes parte en None, al final lo devolveremos al estado␣
 ↪original)
participantes=None
editarConfirmacionParticipantes("Ana", "sí")
assert participantes==lista(participante("Ana", "sí", None), None)
editarConfirmacionParticipantes("Ana", "no")
assert participantes==lista(participante("Ana", "no", None), None)
participantes=None
```

```
Se agregó a  Ana con confirmación  sí
La confirmación cambió, antes era  sí
ahora es  no
```

1.0.9 C)

Defina la función editarInventario. Esta recibirá el nombre de un producto de la lista inventario y un parámetro delta, el que puede ser 1 o -1. En el primer caso indicará que se aumenta en 1 la cantidad de productos y en caso de ser -1 indicará que este se debe disminuir en uno.

Si al disminuir los productos se obtiene cero o un valor negativo, no se debe eliminar el producto, basta con que el parametro cantidad sea cero. Por otra parte si el producto no se encuentra se debe agregar a la lista inventario.

Esta función no debe retornar nada. Pero debe imprimir el nombre del producto, junto con el valor anterior y el nuevo de cantidad. Para el caso en el que se agregue un nuevo producto se debe imprimir el nombre del producto que se agregó. En el caso de que se disminuyan los productos a 0 se debe indicar el nombre del producto que se acabó.

Ejemplos de mensajes:

i) Caso en el que existe el producto y este se aumenta:

```
La cantidad de papas cambió, antes era 1
ahora es 2
```

ii) Caso en el que se agrega un nuevo producto:

```
Se agregó el producto papas
```

iii) Caso en el que se acaba un producto:

```
Ya no quedan papas
```

Utilice la función buscar y la variable global inventario. Además asuma que los nombres de los productos son únicos.

[ ]:
```
# editarInventario: str, int -> None
# efecto: cambia en uno el atributo cantidad del elemento de la lista␣
 ↪inventario cuyo nombre es dado.
# Si al disminuir el parametro cantidad se obtiene cero o un valor negativo, el␣
 ↪parametro cantidad se dejará en cero.
# Si el producto no se encuentra se debe agregar a la lista inventario con el␣
 ↪parametro cantidad en 1.

def editarInventario(nombre, delta):
  global inventario
  p=buscar(inventario, nombre)
  if p==None:
    p=producto(nombre,1)
    inventario=lista(p, inventario)
    print("Se agregó el producto ", p.nombre)
  else:
```

```
    if p.cantidad+delta<=0:
      p.cantidad=0
      print("Ya no quedan ", p.nombre)
    else:
      print("La cantidad de ", p.nombre  ,"cambió, antes era ", p.cantidad )
      p.cantidad=p.cantidad+delta
      print("ahora es ", p.cantidad )
#test (inventario parte en None, al final lo devolveremos al estado original)
inventario=None
editarInventario("chips pop", 1)
assert inventario==lista(producto("chips pop", 1), None)
editarInventario("chips pop", 1)
assert inventario==lista(producto("chips pop", 2), None)
editarInventario("chips pop", -1)
assert inventario==lista(producto("chips pop", 1), None)
editarInventario("chips pop", -1)
assert inventario==lista(producto("chips pop", 0), None)
editarInventario("chips pop", -1)
assert inventario==lista(producto("chips pop", 0), None)
inventario=None
```

```
Se agregó el producto  chips pop
La cantidad de  chips pop cambió, antes era  1
ahora es  2
La cantidad de  chips pop cambió, antes era  2
ahora es  1
Ya no quedan  chips pop
Ya no quedan  chips pop
```

1.0.10 D)

Utilizando las variables globales participantes e inventario y las funciones buscar, editarConfirmacionParticipantes y editarInventario, cree un programa interactivo que siga el siguiente diálogo.

En negritas están los mensajes nuevos que debes agregar al usuario, estos sólo son de tipo input. El resto de los mensajes ya deben haber sido programados en las funciones que definimos anteriormente.

£Salir? no
£Cómo te llamas? María
£Vendrás a la convivencia? sí
Se agregó a María con confirmación sí
£Qué traerás? papas
Se agregó el producto papas
£Salir? no
£Cómo te llamas? Javier
£Vendrás a la convivencia? sí
Se agregó a Javier con confirmación sí
£Qué traerás? papas

La cantidad de papas cambió, antes era 1
ahora es 2
£Salir? no
£Cómo te llamas? Pepe
£Vendrás a la convivencia? no
Se agregó a Pepe con confirmación no
£Salir? no
£Cómo te llamas? María
£Vendrás a la convivencia? no
La confirmación cambió, antes era sí
ahora es no
La cantidad de papas cambió, antes era 2
ahora es 1
£Salir? si

[ ]:
```
# None -> None
# efecto: cambia las variables globales participantes e inventario de acuerdo␣
 ↪con las respuestas dadas por el usuario.

def inscripcion():
  resp=input("£Salir? ")
  if resp!="si":
    global participantes
    global inventario
    nombre=input("£Cómo te llamas? ")
    confirmacion=input("£Vendrás a la convivencia? ")
    editarConfirmacionParticipantes(nombre, confirmacion)
    persona=buscar(participantes, nombre)
    if confirmacion=="no":
      if persona.nombre_producto!=None: #si la persona se habia comprometido a␣
 ↪traer algo debemos actualizar nuestro inventario
        editarInventario(persona.nombre_producto, -1)
        persona.nombre_producto=None
    else:
      if persona.nombre_producto==None:
        nombre_producto=input("£Qué traerás? ")
        editarInventario(nombre_producto, 1)
        persona.nombre_producto=nombre_producto
    inscripcion()
inscripcion()
```

```
£Salir? si
```