Introducción a la Programación
Tercer Entregable de laboratorio

Ejercicio 1

La notación polaca inversa es un método algebraico alternativo de introducción de datos. En una expresión con esta notación, primero están los operandos y después viene el operador que va a realizar los cálculos sobre ellos. Por ejemplo, la expresión ‘(2 * 5) + 7‘ se escribe en notación polaca inversa como ‘2 5 * 7 +‘. Otro ejemplo: ‘5 + ((1+2) * 4) - 3‘ en esta notación es ‘5 1 2 + 4 * + 3 -‘. Notar que no es necesario agregar paréntesis.

Se pide implementar la función `calcular_expresion(expr: str) ->float` , cuya especificación es la siguiente:

`problema calcular_expresion` (in expr: $seq\langle Char\rangle$) : $\mathbb{R}$ {
`requiere:` {expr es no vacío}
`requiere:` {los operandos/operadores están separados por un único espacio}
`requiere:` {expr comienza con un operando y finaliza con un operador (no hay espacios al inicio ni al final de expr)}
`requiere:` {Los operadores en expr pueden ser solamente ‘*‘, ‘+‘, ‘/‘, ‘-‘}
`requiere:` {expr es una expresión en notación polaca inversa válida}
`requiere:` {La evaluación de expr no se indefine}
`asegura:` {res es igual al resultado de evaluar expr}
}

Por ejemplo:

- dado el input ”2 5 * 7 +”, el resultado esperado es 17.0.
- dado el input ”2 3.5 -”, es resultado esperado es -1.5.

Resolverlo utilizando el TAD Pila.

Introducción a la Programación
Tercer Entregable de laboratorio

Ejercicio 2

Se pide implementar la función `unir_diccionarios(a_unir: List[Dict[str,List[str]]]) ->Dict[str, str]` que tome una lista de diccionarios como argumento y devuelva un nuevo diccionario que sea la unión de todos ellos. Si hay claves duplicadas en los diccionarios, el valor correspondiente en el nuevo diccionario debe ser una lista con todos los valores asociados a esa clave en los diccionarios originales. El orden de las claves y valores debe ser según el orden de los diccionarios de entrada.

Por ejemplo, para el input [{'a': 1, 'b': 2}, {'b': 3, 'c': 4}, {'a': 5}], el resultado esperado es {'a': [1, 5], 'b': [2, 3], 'c': [4]}.

Introducción a la Programación
Tercer Entregable de laboratorio

Ejercicio 3

En una tienda de comestibles en línea, se desea implementar un sistema de procesamiento de pedidos. Los clientes realizan sus pedidos en línea y los pedidos se procesan en el orden en que se reciben. Cada pedido se representa como un diccionario con la siguiente estructura:

pedido = { 'id': 1, 'cliente': 'Juan', 'productos': {'Manzana': 2, 'Pan': 4, 'Factura': 6} }

Notar que 'productos' es a su vez un diccionario donde la clave es el nombre del producto, y su valor representa la cantidad de ese producto a solicitar. En el ejemplo: el cliente 'Juan' cuyo 'id' es 1, está solicitando 2 unidades de 'Manzana', 4 unidades de 'Pan' y 6 unidades de 'Factura'.
El procesamiento de pedidos se realiza de la siguiente manera:

- Se toma el primer pedido de la cola.
- Se verifica el stock de cada producto en el pedido. Se descuenta del stock disponible los productos solicitados en un pedido.
- Se calcula el precio total del pedido.
- Se marca el pedido como 'completo' o 'incompleto' dependiendo si había stock suficiente o no para **todos** los productos. Es decir, si no había stock suficiente de algún producto, el estado será 'incompleto' aunque se procesarán todos los productos.
- Se pasa al siguiente pedido en la cola y se repiten los pasos anteriores hasta que no queden más pedidos en la cola.

El stock y los precios de los productos se obtienen de diccionarios con la siguiente estructura:

stock_productos = { "Manzana": 10, "Leche": 5, "Pan": 3, "Factura": 0 }
precio_productos = { "Manzana": 3.5, "Leche": 5.5, "Pan": 3.5, "Factura": 2.75 }

Se pide implementar la función `procesar_pedidos(in pedidos: Queue[Dict[str, Union[int, str, Dict[str, int]]]], inout stock_productos: Dict[str, int], in precios_productos: Dict[str,float]) ->List[Dict[str, Union[int, str, float, Dict[str, int]]]]` que retorne una lista de diccionarios con los pedidos procesados.

Por ejemplo, para el siguiente input de 'pedidos':

```
pedidos = [{
 'id': 21,
 'cliente': 'Gabriela',
 'productos': {'Manzana': 2}
}, {
 'id': 1,
 'cliente': 'Juan',
 'productos': {'Manzana': 2, 'Pan': 4, 'Factura': 6}
}]
```

y los inputs 'stock_productos' y 'precio_productos' que figuran anteriormente, el resultado esperado es:

```
pedido_procesado = [{
 'id': 21,
 'cliente': 'Gabriela',
 'productos': {'Manzana': 2}
```

```
 'precio_total': 7.0,
 'estado': 'completo'
 }, {
 'id': 1,
 'cliente': 'Juan',
 'productos': {'Manzana': 2, 'Pan': 3, 'Factura': 0}
 'precio_total': 17.5,
 'estado': 'incompleto'
}]
```

Aclaraciones:

- pedido: Dict[str, Union[int, str, Dict[str, int]]]: Significa que el tipo de dato de 'pedido' es un diccionario donde su clave es de tipo *str* y su valor de algunos de los siguientes tipos: *int*, *str* ó *Dict[str, int]*. Es decir, puede haber valores de tipo *int*, otros de tipo *str* y/o otros de tipo *Dict[str, int]*. Sin embargo, esta notación no es soportada en el CMS, por lo tanto, verán en el template que descargan que su anotación es simplemente 'pedidos: Queue'.
- Se puede asumir que todos los productos en 'pedidos' existen tanto en 'stock_productos' (pueden tener cantidad 0) como en 'precio_productos'.

Introducción a la Programación
Tercer Entregable de laboratorio

Ejercicio 4

Implementar la función `def avanzarFila(inout fila: Queue, in min: int)`, que modela una fila del banco. El funcionamiento del banco es el siguiente:

- La gente puede comenzar a hacer la fila desde antes de que abra el banco. Cada persona que llega a la fila recibe un número, comenzando por el número 1. Se otorgan números consecutivos.
- La fila empieza con las n personas que llegaron antes de que abra el banco. Cuando abre (a las 10), cada 4 minutos llega una nueva persona a la fila (la primera entra a las 10:00, la siguiente a las 10:04, etc)
- El banco tiene 3 cajas que atienden gente de la siguiente manera:
  - Caja1: Empieza a atender 10:01, y atiende a una persona cada 10 minutos
  - Caja2: Empieza a atender 10:03, atiende a una persona cada 4 minutos
  - Caja3: Empieza a atender 10:02, y atiende una persona cada 4 minutos, pero no le resuelve el problema y la persona debe volver a la fila (se va al final y tarda 3 min en llegar. Es decir, la persona que fue atendida 10:02 vuelve a entrar a la fila a las 10:05)
- En caso que haya más de una caja disponible para atender en el mismo minuto, primero atiende la Caja1, luego la Caja2 y finalmente la Caja3.

Se desea modelar este comportamiento de la fila del banco y que el parámetro $fila$ contenga la fila resultante después de min minutos.

1. Ejemplo 1

Fila inicial: [1,2,3]

- 10:00 llega una nueva persona. Fila: [1,2,3,4]
- 10:01 caja 1 atiende al primero. Fila: [2,3,4]
- 10:02 caja 3 atiende al primero, que va a volver más adelante. Fila: [3,4]
- 10:03 caja 2 atiende al primero. Fila: [4]
- 10:04 llega una nueva persona. Fila: [4,5]
- 10:05 vuelve a la fila el número 2. Fila: [4,5,2]

Test 1

Si se pasa como parámetro $min = 0$, la fila resultado debería ser [1,2,3,4].

Test 2

Si se pasa como parámetro $min = 5$, la fila resultado debería ser [4,5,2].

2. Ejemplo 2

Fila inicial: []

- 10:00 llega una nueva persona. Fila: [1]
- 10:01 caja 1 atiende al primero. Fila: []
- 10:04 llega una nueva persona. Fila: [2]

Test 1

Si se pasa como parámetro $min = 0$, la fila resultado debería ser [1].

Test 2

Si se pasa como parámetro $min = 3$, la fila resultado debería ser [].

3. Aclaración

Se puede usar la función 'qsize()->int' que devuelve la cantidad de elementos de una cola. Por ejemplo si 'c' es una cola con los elementos [1,2,3], entonces c.qsize() devuelve 3.