Introducción a la Programación – UNQ – 2do semestre de 2011

Recu del Segundo Parcial – Call Center

Aclaraciones:

- *Esta evaluación es a libro abierto. Se pueden usar todo lo visto en la práctica y en la teórica, aclarando la referencia.*
- *No se olvide de poner nombre, nro. de legajo, nro. de hoja y cantidad total de hojas en cada hoja.*
- *Le recomendamos leer el enunciado en su totalidad y organizar sus ideas antes de comenzar la resolución.*
- *Recuerde que la intención es medir cuánto comprende usted del tema. Por ello, no dude en escribir todo lo que sabe, en explicar lo que se propone antes de escribir código, en probar su código con ejemplos, etc.*

Un *Call Center* está integrado por un batallón de *telefonistas* entrenados en el arte del Zen, discursos de contención y cría de ovejas. Las telefonistas reciben *reclamos* por parte de *clientes*. Un cliente realiza un reclamo llamando por teléfono al *Call Center*. El primer telefonista libre lo atiende y una vez que termina de hablar con el cliente declara si el reclamo fue satisfecho o no. En el tiempo durante el cual un telefonista atiende a un cliente, no está libre para atender a otro.

Este parcial tiene por objetivo modelar el *Call Center* en CGobstones. Para ello se utilizarán las siguientes estructuras de datos. Un *Call Center* se representa como un par que consiste de una lista de telefonistas y un número que indica el tamaño del *Call Center* (el tamaño no es más que el tamaño de la lista de telefonistas).

```
struct CallCenter {
  List<Telefonista> ts;
  int tamaño;
};
```

Una telefonista tiene un nombre (representado con un entero para simplificar) y atiende una lista de *reclamos*.

```
struct Telefonista {
  int nombre;
  List<Reclamo> reclamos;
};
```

Un *reclamo* consiste del número que llama, un indicador de si el llamado está actualmente en curso (i.e. la telefonista lo está atendiendo en ese momento) y un indicador del resultado del reclamo. El resultado del reclamo se conocerá una vez que la telefonista termina de hablar con el cliente y se cierre la comunicación.

```
struct Reclamo {
  int numeroTel;
  bool enCurso;
  bool resultadoReclamo;
};
```

La llegada de nuevos llamados por parte de los clientes para realizar reclamos, como así también la finalización de esos llamados se va a representar como una lista de *eventos*. Un evento puede ser de uno de dos tipos:

- Nuevo llamado (Verde): aquí se indica el número que llama.
- Fin de llamado (Rojo): aquí se indica el número que llama y si el resultado del reclamo fue satisfecho o no.

```
struct Evento {
  Color tipo;
  int numeroTel;
  bool resultadoReclamo;
};
```

Se solicita resolver los siguientes ejercicios[1].

Ejercicio 1 *Escribir* `crearReclamo`*, una función que dado un número de teléfono, cree un nuevo reclamo indicando que el mismo está en curso (no hace falta asignar nada al resultado).*

Ejercicio 2 *Escribir*

[1] Por abuso de notación utilizaremos el término *función* para referirnos a un procedimiento que devuelve un valor.

a) *una función* `hayTelefonistaLibre` *que dado una lista de telefonistas retorna un booleano indicando si hay una telefonista libre. Una telefonista está libre si no se encuentra actualmente atendiendo una llamada.*

b) *una función* `primerTelefonistaLibre` *que dado una lista de telefonistas retorna el primero que está libre. Puede suponer que hay al menos una libre.*

Ejercicio 3 *Escribir* `ProcesarNuevoLlamado`*, un procedimiento que dado un evento de nuevo llamado y un* Call Center*, retorna un nuevo* Call Center *que refleja que se está atendiendo el nuevo reclamo. Para ello debe crearse el reclamo, indicar que está en curso y asignarse a la primera telefonista libre.* ***Nota:*** *Puede asumir que hay al menos una telefonista libre.*

Ejercicio 4 *Escribir* `ProcesarFinDeLlamado`*, un procedimiento que dado un evento de fin de llamado y un* Call Center*, retorna un nuevo* Call Center *que refleja que el llamado finalizó y el reclamo se resolvió o no.* ***Nota:*** *Puede asumir que el llamado a finalizar existe y corresponde a un reclamo actualmente en curso.*

Ejercicio 5 *Escribir una función* `simularCallCenter` *que dado un* Call Center *y una lista de eventos, procese todos los eventos y retorne el* Call Center *resultante. En el caso de que haya un nuevo llamado y no haya telefonistas libres, el evento debe pasarse al final de la lista actual de eventos*[2]*.*

Ejercicio 6 *Escribir la función* `efectividad`*, que dado un* Call Center *retorne la cantidad de reclamos que se satisficieron.*

Ejercicio 7 *Escribir la función* `tomarPrimerasN` *que dado una lista de telefonistas y un número positivo n, retorna las primeras n telefonistas.* ***Nota:*** *Puede asumir que n es menor o igual que la longitud de la lista de telefonistas.*

Ejercicio 8 *Escribir la función* `minNroDeTelefonistasNecesarias`*, que dado un* Call Center *y una lista de eventos, retorne la lista con menor cantidad de telefonistas (de entre aquellas del* Call Center*) necesarias que no empeoren la efectividad. El proceso a seguir se ejemplifica de la siguiente manera. Supongamos que* `[t1,t2,t3]` *es la lista de telefonistas. Primero determinar la efectividad resultante de simular el Call Center teniendo a la lista* `[t1]` *como telefonistas. Luego con* `[t1,t2]`*. Y finalmente con* `[t1,t2,t3]`*. En el momento en que el agregado de una telefonista no modifica la efectividad, el proceso se detiene y se reporta la lista resultante. Por ejemplo, si para* `[t1]` *la efectividad es 2, para* `[t1,t2]` *es 3 y para* `[t1,t2,t3]` *es también 3, entonces debe retornarse* `[t1,t2]` *(la tercer telefonista no le da mayor efectividad al Call Center para esa lista de eventos).*

[2] Si bien esto no representa una estrategia justa, simplifica el análisis del ejercicio.