Introducción a la Programación – UNQ – 1er semestre de 2013

Segundo parcial – WhileMart

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.*

Los directivos del supermercado WhileMart desean optimizar el número de cajas de sus instalaciones, de tal modo que sean mínimas para reducir los costos, pero suficientes para evitar la acumulación y demora de los clientes. Para ello se dispone de la información de los clientes de una sucursal en un día típico, que se modela mediante los siguientes registros:

```
struct Cliente {
    int horaIngresoCola;
    int cantProductos;
};

struct Caja {
    int nroCaja;
    List<Cliente> clientesEsperando;
};

struct Super {
    int horaActual;
    List<Caja> lineaDeCajas;
    List<Cliente> clientesComprando;
};
```

Para cada cliente, se conoce la hora a la que empezó a hacer cola (`horaIngresoCola`) y la cantidad de productos que compró (`cantProductos`). Cada caja del supermercado se identifica por un número de caja (`nroCaja`) e incluye una lista de los clientes que están haciendo cola en esa caja (`clientesEsperando`).

Para simular un supermercado, se cuenta con el registro `Super`, que indica la hora actual en la simulación del supermercado (`horaActual`), la lista de todas las cajas (`lineaDeCajas`) y la lista de todos los clientes que todavía no comenzaron a hacer cola (`clientesComprando`).

Se asumen los siguientes hechos sobre el funcionamiento del supermercado:

- Cuando un cliente termina de comprar y comienza a hacer cola, elige la caja en la que haya menos personas esperando. En caso de empate, elige la caja que tenga el menor número de caja.
- La lista de clientes esperando en una caja representa una cola. El primer elemento de la lista representa el cliente que está siendo atendido.
- Las cajas procesan un producto por unidad de tiempo.
- Cuando una caja termina de procesar todos los productos del cliente que está siendo atendido, dicho cliente se retira y se pasa al siguiente cliente esperando en la cola, si lo hay.
- Se asumirá siempre que las líneas de cajas no contienen cajas con números repetidos.

Se solicita resolver los siguientes ejercicios. Tener en cuenta que *ninguno* de ellos involucra el tablero de CGobstones.

Ejercicio 1 `Caja cajaMenosOcupada (List<Caja> lineaDeCajas)`

Propósito: *Denota la caja de la línea de cajas que está* menos ocupada*, es decir, aquella en la que hay menos clientes haciendo cola. En caso de empate, se elige la caja que esté menos ocupada y que tenga el número más chico. Por ejemplo, si*

la caja 1 está atendiendo un cliente, y las cajas 2 y 3 están libres, se elige la caja 2.

Precondición: *La lista* `lineaDeCajas` *debe ser no vacía.*

Ejercicio 2 `List<Caja> agregarCliente (List<Caja> lineaDeCajas, Cliente c)`

Propósito: *Denota la línea de cajas después de que el cliente* `c` *comience a hacer cola. El cliente se ubica en la caja que esté menos ocupada, de acuerdo con el criterio del ejercicio anterior.*

Precondición: *La lista* `lineaDeCajas` *debe ser no vacía.*

Ejercicio 3 `List<Caja> avanzarCaja (List<Caja> lineaDeCajas, int nroCaja)`

Propósito: *Denota la línea de cajas después de que la caja identificada por el número* `nroCaja` *procese un producto. Si la caja en cuestión no tiene clientes, la línea de cajas debe quedar tal como estaba. Si ya se procesaron todos los productos del cliente que está siendo atendido, dicho cliente se retira de la caja.*

Precondición: *Debe haber una caja identificada por el número* `nroCaja`.

Ejercicio 4 `List<Caja> avanzarLineaDeCajas (List<Caja> lineaDeCajas)`

Propósito: *Denota la línea de cajas después de que todas y cada una de las cajas procesen un producto, con las mismas observaciones que en el ejercicio anterior.*

Precondición: *No tiene.*

Ejercicio 5 `Super simularFinalDeLasCompras (Super s)`

Propósito: *Toma todos los clientes que están terminando de hacer sus compras en el supermercado y los ubica en la cola que les corresponda. Dicho de otro modo, denota el supermercado después de que hayan empezado a hacer cola todos aquellos clientes cuya* `horaIngresoCola` *coincide con la hora actual del supermercado. Cuando un cliente ingresa a una cola lo hace según el criterio ya establecido (elige la caja menos ocupada).* Precondición: *No tiene.*

Nota: Puede asumir la existencia de `List<Cliente>clientesQueIngresanALas(List<Cliente>, int hora)` *y* `List<Cliente>clientesQueNoIngresanALas(List<Cliente>, int hora)` *que retornan los valores esperados.*

Ejercicio 6 `Super simularPasoDelTiempo (Super s)`

Propósito: *Denota el supermercado después de que pase una unidad de tiempo. Lo que se debe hacer es, en orden:*

- *Hacer que los clientes que terminaron sus compras en la hora actual pasen a estar esperando en las cajas.*
- *Procesar un producto en cada una de las cajas de la línea de cajas.*
- *Avanzar el reloj una unidad de tiempo.*

Precondición: *No tiene.*

Ejercicio 7 `Super supermercadoVacio (Super s)`

Propósito: *Denota Verdadero si el supermercado está completamente vacío, es decir, si no hay clientes comprando y no hay clientes esperando en ninguna caja.* Precondición: *No tiene.*

Ejercicio 8 `int horaEnQueQuedaVacio (Super s)`

Propósito: *Denota la hora en la que el supermercado queda completamente vacío.* Nota: *Para determinar ese valor, simular el paso del tiempo hasta que el supermercado quede vacío.* Precondición: *No tiene.*