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

Segundo parcial – Vuela vuela

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.*

El presente parcial tiene por finalidad simular el tráfico aéreo, en la visión de una Torre de Control que lleva a cabo el seguimiento de cada nave. Una *Torre de Control* consiste de una lista de vuelos y una lista que asocia un plan de vuelos con cada par de la forma (origen,destino). La lista de vuelos va a ir variando a medida que los vuelos despegan y aterrizan. Sin embargo, la lista que asocia planes de vuelo para cada par (origen,destino) siempre es la misma: si estando en origen se quiere ir a destino, entonces habrá un único plan de vuelo que permita conectar esas dos coordenadas.

El *plan de vuelos* consiste de una lista de direcciones N, S, E u O. Un *vuelo* consiste de un número de vuelo (único), un destino, la etapa actual del plan de vuelo en la que se encuentra la nave y el estado. La etapa del plan de vuelo va entre 1 y la longitud del plan de vuelo. El estado de un vuelo puede ser

- Saliente (Verde): Avión listo para despegar.
- En ruta (Azul): Avión en vuelo.
- Aterrizado (Negro): Avión llegó a destino.

Los siguientes registros modelan estos datos.

```
struct TorreDeControl
{
    List<Vuelo> vs;
    List<PlanDestino> ps;
}

struct Vuelo
{
    int numero;
    Coord origen;
    Coord destino;
    Color estado;
    int etapaEnPlanDeVuelo;
}

struct PlanDestino
{
     Coord origen;
     Coord destino;
     List<Dir> planDeVuelo;
};
```

Se solicita resolver los siguientes ejercicios. Tener en cuenta que ningún ejercicio involucra el uso del tablero CGobstones con la *excepción* del último.

Ejercicio 1 `Vuelo hallarVuelo(List<Vuelo>vs, Color estado)`

Propósito: *Retorna el primer vuelo en la lista de vuelos que tiene el estado dado como parámetro.*

Precondición: *Existe al menos un vuelo con ese estado.*

Ejercicio 2 `TorreDeControl despegarVuelo(TorreDeControl t)`

Propósito: *Hace despegar al primer vuelo en la lista de vuelos que esté "listo para despegar". Para ello debe cambiar su estado a "en vuelo" e indicar que se encuentra en la etapa 1 del plan de vuelo.*

Precondición: *Existe al menos un vuelo en estado de "listo para despegar".*

Ejercicio 3 `TorreDeControl aterrizarVuelo(TorreDeControl t)`

Propósito: *Hace aterrizar el primer vuelo en la lista de vuelos que esté "en ruta" y que haya llegado a destino (la etapa coincide con la longitud del plan de vuelo). Para ello debe cambiar su estado a "aterrizado".*

Precondición: *Existe al menos un vuelo en condiciones de aterrizar.*

Ejercicio 4 `Coord coordenadaEnEtapa_i(TorreDeControl t, Vuelo v, int i)`

Propósito: *Retorna la coordenada sobre la que estará sobrevolando el vuelo* `v` *dentro de* `i` *etapas, contando desde la etapa actual.*

Precondición: *La etapa actual más* `i` *no supera la longitud del plan de vuelo.*

Ejercicio 5 `bool hayColisionEnEtapa_i(TorreDeControl t, int i)`

Propósito: *Indica si hay aviones que colisionan en la etapa* `i`*, contando desde la etapa actual. Dos aviones colisionan en etapa* `i` *si pasan por la misma coordenada en esta etapa.*

Ejercicio 6 `bool hayColision(TorreDeControl t)`

Propósito: *Indica si hay aviones que colisionan en alguna etapa. Tener en cuenta que las etapas van desde 1 hasta la máxima etapa posible (la longitud del plan de vuelos más largo posible).*

Nota: El siguiente ejercicio involucra el uso del tablero.

Ejercicio 7 `void Radar(TorreDeControl t)`

Propósito: *Dibuja la ubicación actual de cada vuelo. Los vuelos deben codificarse con tantas bolitas verdes como indica su número de vuelo. Cada origen con una bolita azul y cada destino con una bolita negra (el origen y destino debe codificarse a lo sumo una vez en el tablero).*

Precondición: *Puede asumir la existencia de una función* `bool enTablero(Coord c)` *que indica si una coordenada cae dentro del tablero o no como así también el procedimiento* `void PonerEnCoord(Coord crd, Color color)` *que pone una bolita del color indicado en la coordenada indicada (se asume que la coordenada cae dentro del tablero).*