Universidad de Málaga
E.T.S.I. Telecomunicación

Laboratorio de Programación 2
junio de 2004
(1º de Ingeniería de Telecomunicación)

Alumno: Grupo: Máquina:

Se desea implementar un sistema para controlar el uso de las pistas de aterrizaje de un aeropuerto. El aeropuerto consta de `MAX_PISTAS` pistas, numeradas desde 0 hasta `MAX_PISTAS-1`. Una pista puede o no estar disponible. Los vuelos estarán esperando utilizar una pista para aterrizar o despegar. Para cada vuelo, el sistema almacenará la siguiente información:

identificador cada vuelo estará identificado por un número entero (`long int`).

acción un vuelo puede estar esperando para despegar o aterrizar. Esta situación se representará por el tipo enumerado `TEspera`.

prioridad Los vuelos tienen asignada una prioridad desde la mínima (0) hasta la máxima (`MAX_PRIO=100`).

El sistema debe almacenar para cada pista el estado en que se encuentra (disponible o no) y, caso de que esté disponible, la lista de vuelos que esperan utilizarla. Las listas de vuelos en espera de cada pista estarán ordenadas por prioridades, de mayor a menor. La figura de abajo muestra la estructura del sistema de control de pistas suponiendo que el aeropuerto tiene 4 pistas (`MAX_PISTAS=4`) y que las letras `A` y `D` representan aterrizar y despegar, respectivamente.

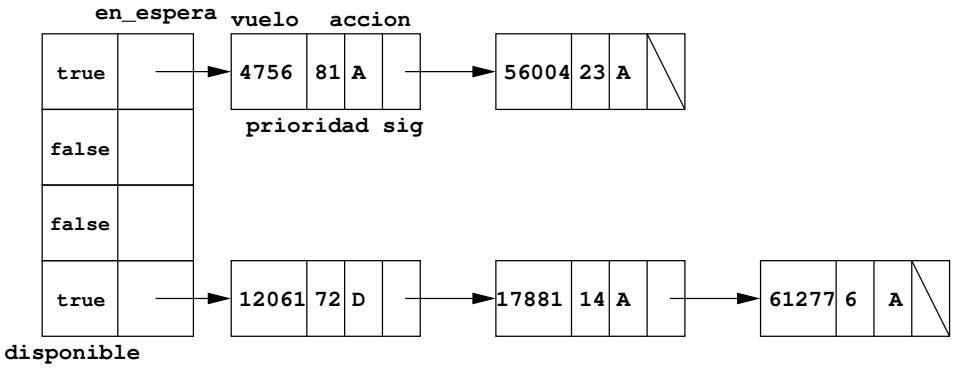

Figura 1: Un aeropuerto con 4 pistas y 5 aviones en espera

En la figura anterior, solo dos pistas están disponibles (la 0 y la 3). Observa que las listas de espera están ordenadas por prioridad. Cuando un vuelo solicite pista para aterrizar o despegar, se le debe asignar la pista disponible en la que quede mejor situado (es decir, con menos vuelos en espera por delante). Por ejemplo, si en la sitación descrita en la figura anterior un vuelo con prioridad 20 pide pista, se le debe asignar la pista 3, ya que queda en segundo lugar de la lista de espera de dicha pista.

El control del uso de las pistas se realizará a través de la clase `CAeropuerto`, definida por el siguiente interfaz:

```
class CAeropuerto
{
   public:

     // constantes y tipos públicos, según convenga

     // métodos públicos

     CAeropuerto();
```

```
~CAeropuerto();

void Asignar_Pista(const long int vuelo, const TEspera accion, const int prioridad);
   // inserta un vuelo, asignándole la ‘‘mejor’’ pista

void Leer_Vuelos(const char[] nombre_fichero);
   // lee un fichero de texto que contiene en cada línea los datos de un
   // vuelo. El primer dato de cada línea es el número de vuelo, el segundo
   // la prioridad y el tercero la acción que quiere realizar.
   // Por ejemplo, el siguiente fichero tiene los datos de 3 vuelos:
   //
   //  47563     3     aterrizar
   //  12051     5     despegar
   //  23406     0     aterrizar

// pre: 0 <= p <= MAX_PISTAS-1
void Aterrizar(const int p);
   // retira de la pista p el vuelo con máxima prioridad que esperaba
   // para aterrizar

// pre: 0 <= p <= MAX_PISTAS-1
void Despegar(const int p);
   // retira de la pista p el vuelo con máxima prioridad que esperaba
   // para despegar

// pre: el vuelo está en la estructura
void Cambiar_Prioridad(const long int vuelo, const int prioridad);
   // cambia la prioridad de un vuelo, y le reasigna pista de acuerdo
   // con su nueva prioridad

// pre: el vuelo está en la estructura
void Pista_y_Puesto(const int vuelo, int& pista, int& puesto) const;
   // devuelve el número de pista y el puesto en la lista de espera
   // de dicha pista que tiene asignado un vuelo. Los puestos se numeran desde 1.

void Vuelo_Mas_Urgente(int& pista, long int& vuelo, TEspera& accion, int& prioridad) const;
   // devuelve los datos del vuelo más prioritario en espera

// pre: 0 <= p <= MAX_PISTAS-1
void Cerrar_Pista(const int p);
   // cierra la pista p, asignando una nueva pista a los vuelos que
   // esperaban utilizar p

void Escribir_Datos(const char[] nombre_fichero) const;
   // escribe un fichero de texto con los datos de los vuelos
   // en espera, incluyendo pista y puesto asignados.
   // lso vuelos deben aparecer ordenados por prioridad.
   // Para el ejemplo de la figura se obtiene el fichero:
   //
   //  4756     81    aterrizar    0    1
   // 12061     72    despegar     3    1
   // 56004     23    aterrizar    0    2
   // 17881     14    aterrizar    3    2
   // 61277      6    aterrizar    3    3

// pre: 0 <= p <= MAX_PISTAS-1
void Abrir_Pista(const int p);
   // abre la pista p y le asigna vuelos en espera que retira
   // de las demás pistas, de acuerdo con la siguiente política:
   //
   // el primero de p será el segundo más prioritario del resto,
```

```
        // el segundo de p será el tercero más prioritario del resto,
        // y así sucesivamente

   private:

     // constantes, tipos, variables y métodos privados, según convenga

};
```

Completa el interfaz anterior (`aeropuerto.hpp`), implementa la clase `CAeropuerto` (`aeropuerto.cpp`) y escribe un programa cliente (`cliente.cpp`) para probar su funcionamiento.

Notas

- Para aprobar es necesario hacer los métodos: `CAeropuerto`, `~CAeropuerto`, `Asignar_Pista`, `Leer_Vuelos`, `Aterrizar`, `Despegar`.