Parque de atracciones

Se desea desarrollar una aplicación que permita gestionar un parque de atracciones. En el parque hay *atracciones*, cada una identificada a través de un nombre único, y *niños*, cuyos nombres también son únicos. Cuando un niño llega al parque por primera vez se registra su visita. El niño puede quedarse dando vueltas por el parque, sin subirse en ninguna atracción, o bien ponerse en la cola de una atracción y esperar a que le toque montar. Eso sí, no puede estar en la cola de más de una atracción a la vez. Por último, se pueden consultar las atracciones en las que un niño ha montado o el número de veces que lo ha hecho en una atracción concreta. La implementación de la aplicación hará uso de un TAD `ParqueDeAtracciones` que debe proporcionar las siguientes operaciones:

- `crea()`: Crea un parque de atracciones vacío.
- `an_atraccion(nombre_atraccion)`: Añade una nueva atracción, con nombre `nombre_atraccion`, al parque, de forma que pasa a estar disponible para que los niños se monten. Si ya existe una atracción con ese nombre señala un error (excepción `EAtraccionYaExiste`).
- `an_niño(nombre_niño)`: Registra en el parque la visita de un nuevo niño, de nombre `nombre_niño`. Si ya está registrado un niño con dicho nombre señala un error (excepción `ENinioYaRegistrado`)
- `poner_en_fila(nombre_niño, nombre_atraccion)`: Registra el acceso del niño de nombre `nombre_niño` a la atracción de nombre `nombre_atraccion`, es decir, el niño pasa a estar al final de la cola de dicha atracción. Para poderse realizar la acción el niño debe estar registrado, la atracción debe estar disponible, y el niño no debe estar ya esperando para montar en una atracción; en caso contrario la operación señala un error (excepción `EAccesoAtraccion`).
- `avanzar(nombre_atraccion)`: Registra el disfrute de la atracción de nombre `nombre_atraccion` por parte del primer niño en la cola de la misma. Como resultado el niño abandona la cola de dicha atracción y está en disposición de montar nuevamente en alguna atracción. En caso de que la atracción no exista la operación señala un error (excepción `EAtraccionNoExiste`) La operación no tiene efecto si la cola de la atracción está vacía.
- `atracciones_visitadas(nombre_niño)->lista`: Devuelve una lista con los nombres de las atracciones en las que ha montado el niño de nombre `nombre_niño`, ordenadas alfabéticamente. En caso de que el niño no esté registrado señala un error (excepción `ENinioNoRegistrado`).
- `numero_visitas(nombre_niño, nombre_atraccion)->numero`: Devuelve el número de veces que el niño de nombre `nombre_niño` ha montado en la atracción de nombre `nombre_atraccion`. En caso de que la atracción no exista o el niño no esté registrado señala un error (excepción `EConsultaNumViajes`)

Trabajo a realizar

Se debe completar un programa que lea una serie de líneas, cada una de las cuáles contiene un *comando* para invocar una operación, ejecute los comandos leídos, y muestre el resultado. Los comandos tienen el mismo nombre y propósito que las operaciones del TAD. Su formato es el siguiente:

- `an_atraccion` <*nombre atracción*>
- `an_ninio` <*nombre_ninio*>
- `poner_en_fila` <*nombre_ninio*> <*nombre_atraccion*>
- `avanzar` <*nombre_atraccion*>
- `atracciones_visitadas` <*nombre_ninio*>
- `numero_visitas` <*nombre_ninio*> <*nombre atracción*>

La salida generada para cada comando es como sigue:

- `an_atraccion`: se imprime `OK` si todo ha ido bien; `ATRACCION_YA_EXISTE` si la atracción dada ya existe.
- `an_ninio`: se imprime `OK` si todo ha ido bien; `NINIO_YA_REGISTRADO` si el niño dado ya está registrado.
- `poner_en_fila`: se imprime `OK` si todo ha ido bien; `ERROR_ACCESO_ATRACCION` si la operación no se ha podido ejecutar.

- `avanzar`: se imprimr `OK` si todo ha ido bien; `ATRACCION_NO_EXISTE` si la atracción dada no existe.
- `atracciones_visitadas`: si todo ha ido bien, se imprime la lista de las atracciones visitadas ordenada alfabéticamente; `NINIO_NO_REGISTRADO` en otro caso.
- `numero_visitas`: si todo ha ido bien, se imprime el número de veces que el niño ha disfrutado la atracción; `ERROR_CONSULTA_NUMERO_VIAJES` en otro caso.

Ejemplo de entrada / salida

Entrada	Salida
`an_atraccion globos_locos`	`OK`
`an_atraccion la_jungla`	`OK`
`an_ninio pepito`	`OK`
`an_ninio juanita`	`OK`
`poner_en_fila pepito la_jungla`	`OK`
`poner_en_fila juanita globos_locos`	`OK`
`avanzar la_jungla`	`OK`
`poner_en_fila pepito globos_locos`	`OK`
`avanzar globos_locos`	`OK`
`avanzar globos_locos`	`OK`
`atracciones_visitadas pepito`	`globos_locos la_jungla`
`atracciones_visitadas juanita`	`globos_locos`
`numero_visitas pepito la_jungla`	`1`
`numero_visitas juanita la_jungla`	`0`

Se proporciona el archivo `main.cpp` en el que se implementa la lógica de entrada/salida necesaria. El código proporcionado no debe modificarse.

Se proporciona, así mismo, el archivo `ParqueDeAtracciones.h`, con la definición de las operaciones de TAD. Dicho archivo debe completarse eligiendo una representación apropiada para gestionar la información del parque de atracciones. Pueden, asimismo, definirse todos los métodos privados adicionales en `ParqueDeAtracciones` que se estimen oportunos.

Las operaciones del TAD deben implementarse, en el archivo `ParqueDeAtracciones.cpp`.