Ejercicio 3

3 puntos

Bienvenidos al restaurante *La caracena*, donde el cocinero es tan meticuloso que elabora los pedidos en el mismo orden en el que le llegan. ¡Ningún pedido se cuela! El restaurante dispone de una gran cantidad de mesas, cada una de ellas identificada por un número. Los comensales de las mesas pueden pedir platos a lo largo de la noche. La cocina va recibiendo los pedidos de todas las mesas y va sirviendo los platos, siempre comenzando por el pedido más antiguo que esté pendiente de servir.

Se pide implementar un TAD `Restaurante` que implemente las siguientes operaciones:

- `void nueva_mesa(int num)`

  Registra que un grupo de comensales ha llegado a la mesa `num`. A partir de ese momento, se podrán realizar pedidos desde esa mesa. Si la mesa `num` ya tenía comensales, se lanza una excepción `domain_error` con el mensaje `Mesa ocupada`.

- `void nuevo_pedido(int mesa, const string &plato)`

  Indica a la cocina que la `mesa` cuyo identificador se pasa como primer parámetro ha pedido el `plato` pasado como segundo parámetro. Si la `mesa` no tiene comensales, se lanza una excepción `domain_error` con el mensaje `Mesa vacia`. Ten en cuenta que una mesa puede pedir varias veces el mismo plato a lo largo de la noche.

- `void cancelar_pedido(int mesa, const string &plato)`

  Cancela el último pedido (esto es, el más reciente) del `plato` indicado por parte de la `mesa` pasada como primer parámetro. Solo pueden cancelarse los pedidos que estén en cocina, esto es, pedidos que no hayan sido servidos aún. Si la mesa había pedido varias veces el plato indicado, se cancela solamente el pedido más reciente, pudiéndose cancelar el resto de pedidos mediante sucesivas llamadas a `cancelar_pedido`. Si la `mesa` no tiene comensales, se lanza una excepción `domain_error` con el mensaje `Mesa vacia`. Si la cocina no tiene ningún pedido pendiente del `plato` indicado para esa `mesa`, se lanzará un excepción `domain_error` con el mensaje `Producto no pedido por la mesa`.

- `pair<int, string> servir()`

  Obtiene el pedido más antiguo que esté pendiente de servir, y lo registra como servido en la mesa correspondiente. Devuelve un `pair` con el identificador de la mesa y el plato correspondientes al pedido. Si la cocina no tiene pedidos pendientes de servir, se lanza una excepción `domain_error` con el mensaje `No hay pedidos pendientes`.

- `vector<string> que_falta(int mesa) const`

  Devuelve una lista con aquellos platos pedidos por la `mesa` pasada como parámetro que aún no han sido servidos. La lista resultante debe estar ordenada de manera ascendente según el nombre de los platos, utilizando el orden lexicográfico. Si la mesa tiene varios pedidos pendientes de un mismo plato, el plato solamente debe aparecer una vez en la lista devuelta. Si la mesa no tiene comensales, se lanza una excepción `domain_error` con el mensaje `Mesa vacia`.

La implementación de las operaciones debe ser lo más eficiente posible. Por tanto, debes elegir una representación adecuada para el TAD, implementar las operaciones y justificar la complejidad resultante.

Los métodos del TAD no deben mostrar nada por pantalla. El manejo de la entrada y salida de datos se realizará en funciones externas al TAD.

Entrada

La entrada consta de una serie de casos de prueba. Cada caso está formado por una serie de líneas, en las que se muestran las operaciones a llevar a cabo, una por cada línea: el nombre de la operación seguido de sus argumentos. La palabra `FIN` en una línea indica el final de cada caso.

Salida

Las operaciones `nueva_mesa`, `nuevo_pedido`, `cancelar_pedido` no imprimen nada, salvo en caso de error. Con respecto al resto de operaciones:

- `servir` imprime una línea con el plato servido y el número de mesa que lo pidió, ambos separados por un espacio.
- `que_falta X` debe imprimir una línea con el mensaje `En la mesa X falta:` (donde `X` es el número de mesa pasado como parámetro). Después deben imprimirse los elementos de la lista devuelta por la operación, cada uno en una línea y precedido por dos espacios.

Si una operación produce un error, entonces se escribirá una línea con el mensaje `ERROR:`, seguido del mensaje de la excepción que lanza la operación, y no se escribirá nada más para esa operación.

Cada caso termina con una línea con tres guiones (`---`).

Entrada de ejemplo

```
nueva_mesa 1
nuevo_pedido 1 bravas
nueva_mesa 7
nuevo_pedido 7 gazpacho
nuevo_pedido 7 croquetas
servir
servir
que_falta 7
que_falta 1
nuevo_pedido 7 ensalada
nuevo_pedido 7 croquetas
que_falta 7
cancelar_pedido 7 croquetas
que_falta 7
cancelar_pedido 7 croquetas
que_falta 7
servir
servir
FIN
nueva_mesa 1
nueva_mesa 1
nuevo_pedido 2 croquetas
nuevo_pedido 1 croquetas
cancelar_pedido 1 bravas
servir
servir
cancelar_pedido 1 croquetas
FIN
```

Salida de ejemplo

```
bravas 1
gazpacho 7
En la mesa 7 falta:
  croquetas
En la mesa 1 falta:
En la mesa 7 falta:
  croquetas
  ensalada
En la mesa 7 falta:
  croquetas
  ensalada
En la mesa 7 falta:
  ensalada
ensalada 7
ERROR: No hay pedidos pendientes
---
ERROR: Mesa ocupada
ERROR: Mesa vacia
ERROR: Producto no pedido por la mesa
croquetas 1
ERROR: No hay pedidos pendientes
ERROR: Producto no pedido por la mesa
---
```