Facultad de Informática – Universidad Complutense de Madrid
Fundamentos de la programación I – Grupos D,G,I,DG

Examen de la Convocatoria Ordinaria 22-23

Tiempo disponible: 3 horas

Se quiere construir un programa en C++ que gestione las **carteras** de los clientes de un *bróker* (gestor de acciones bursátiles). La cartera de cada cliente se compone de un conjunto de **acciones** de una o de varias **empresas** (un determinado número de cada una). Cada día se realizan una serie de **operaciones** de compra y venta: se **venden** algunas de las acciones que un cliente tiene de una determinada empresa o se **compran**, para un cliente, unas acciones de una empresa. El programa deberá procesar las operaciones, actualizando, convenientemente, las carteras de los clientes.

El bróker gestiona hasta **5** clientes. Cada cliente se identifica con un código de tres caracteres y dispone de una cartera de acciones de hasta **3** empresas. De cada empresa se mantiene su identificador (tres letras) y el número de acciones.

Por otro lado, tenemos una serie de operaciones de compra y venta de acciones de determinadas empresas para las carteras de determinados clientes. Las operaciones se encuentran en un archivo `broker.txt` como el de la derecha. Para cada operación se indica, en una línea y separados por un espacio, el código del cliente, si se trata de una compra (B) o venta (S) de acciones, las siglas de la empresa y el número de acciones que se compran o venden. La última línea tiene XXX como código de cliente (centinela). El programa procesará las operaciones.

`broker.txt`

```
C02 B IBM 10
C04 B IBM 50
C02 B AMZ 40
C02 S IBM 10
C01 B AMZ 10
C06 B IBM 20
C04 S IBM 10
...
C05 S APL 20
XXX
```

Detalles de implementación

En el programa necesitamos los siguientes tipos **[1,5 puntos]**:

- Cartera de acciones (`tCartera`): Una lista de hasta 3 empresas de las que se tienen acciones. De cada empresa se mantiene su nombre y el número de acciones que tiene el cliente.
- Lista de clientes (`tBroker`): Una lista de hasta 5 clientes, cada uno con su código y con su cartera de acciones.
- Operación (`tOperacion`): su `tipo` (B si es compra o S si es venta), el `código` del cliente que compra o vende, la `empresa` de la que se compran o venden acciones, y el número de `acciones` que se compran o venden.

- Lista de operaciones (`tOpList`): Una lista de hasta 20 operaciones para guardar las operaciones rechazadas (operaciones que no se puedan realizar).

El programa principal comenzará con una lista de clientes vacía y procesará las operaciones del archivo `broker.txt`. Para cada operación, mostrará en la pantalla qué se quiere comprar o vender (*ver el ejemplo de ejecución*) y localizará el cliente sobre cuya cartera se quiere realizar la compra o venta. Si se trata de una operación de compra, el cliente todavía no existe en la lista y hay sitio en ella, añadirá un nuevo cliente con ese código. Si es una venta y el cliente no existe, se ignora la operación. Una vez localizado el cliente (nuevo o existente), procede a realizar la operación, mostrando a continuación cómo queda la lista de clientes tras la operación (*ver el ejemplo de ejecución*). Las operaciones que no puedan realizarse se guardan en una lista de operaciones rechazadas que se mostrará al final. **[1,5 puntos]**

Implementa, al menos, los siguientes subprogramas:

- `buscarCliente()`: dada la lista de clientes y un `código`, devuelve el índice en el que se encuentra, en el array de clientes, uno con ese `código`, o `-1` si no hay ningún cliente con ese código. **[0,5 puntos]**
- `buscarActivo()`: dado un cliente y una empresa, devuelve el índice en el que se encuentra, en el array de la cartera de acciones del cliente, esa empresa, o `-1` si no está esa empresa. **[0,75 puntos]**
- `eliminar()`: dado un cliente y una empresa, elimina el activo de esa empresa de la cartera del cliente (se llama siempre existiendo ese activo). **[0,75 puntos]**
- `nuevoCliente()`: dada la lista de clientes y un código, añade un nuevo cliente al final de la lista, devolviendo el índice en el que se ha creado el nuevo cliente, o -1 si la lista ya está llena. **[0,75 puntos]**
- `nuevaOp()`: dada la lista de operaciones no válidas y una operación, añade esa operación al final de la lista. **[0,5 puntos]**
- `operar()`: Dada una operación, un cliente y la lista de operaciones rechazadas, realiza, si es posible, la operación de compra o venta de acciones en la cartera del cliente. Si no es posible incorpora la operación a la lista de operaciones rechazadas. Para implementar esta operación se deben utilizar las operaciones implementadas anteriormente. **[2 puntos]**

  Comenzará localizando la empresa en la cartera del cliente.

  - Compra de acciones:
    - Si existe la empresa en la cartera del cliente, se añaden las acciones.
    - Si no existe y hay espacio en la cartera del cliente, se añade al final de la cartera del cliente la empresa con ese número de acciones.

- Si no existe y no hay espacio en la cartera del cliente, se añade la operación a la lista de operaciones rechazadas, siempre que haya espacio en esta lista. Si no hay espacio en la lista de operaciones rechazadas la operación se pierde.

- Venta de acciones:
  - Si no existe la empresa en la cartera del cliente, se añade la operación a la lista de operaciones que no se han podido realizar, siempre que haya espacio en esta lista.
  - Si existe se descuentan las acciones a vender hasta el máximo de las acciones que se tengan, y si no se tienen suficientes se añaden las que no se han podido vender a la lista de operaciones rechazadas. Si el número de acciones termina siendo 0, se elimina esta empresa de la cartera del cliente.

- `mostrarOp()`: dada una operación, la muestra en la pantalla. Se valorará que la operación se implemente utilizando sobrecarga de operadores. **[0,25 puntos]**
- `mostrarCli()`: dado un cliente muestra su código y su lista de activos (cartera) Se valorará que la operación se implemente utilizando sobrecarga de operadores. **[0,5 puntos]**
- `mostrar()`: muestra la lista de clientes de un bróker con sus carteras. Se valorará que la operación se implemente utilizando sobrecarga de operadores. **[0,5 puntos]**
- `guardar()`: guarda la lista de operaciones no válidas en un archivo `rechazadas.txt`, con el mismo formato que el fichero de entrada broker.txt. **[0,5 puntos]**

Ejemplo de ejecución

```
B 10 IBM C02

C02: 10xIBM


B 50 IBM C04

C02: 10xIBM
C04: 50xIBM


B 40 AMZ C02

C02: 10xIBM 40xAMZ
C04: 50xIBM


S 10 IBM C02

C02: 40xAMZ
C04: 50xIBM


B 10 AMZ C01

C02: 40xAMZ
C04: 50xIBM
C01: 10xAMZ


B 20 IBM C06

C02: 40xAMZ
C04: 50xIBM
C01: 10xAMZ
C06: 20xIBM
```

S 10 IBM C04

C02: 40xAMZ
C04: 40xIBM
C01: 10xAMZ
C06: 20xIBM

B 10 APL C05

C02: 40xAMZ
C04: 40xIBM
C01: 10xAMZ
C06: 20xIBM

C05: 10xAPL

B 20 SSG C02

C02: 40xAMZ 20xSSG
C04: 40xIBM
C01: 10xAMZ
C06: 20xIBM
C05: 10xAPL

B 50 MSF C02

C02: 40xAMZ 20xSSG 50xMSF
C04: 40xIBM
C01: 10xAMZ
C06: 20xIBM
C05: 10xAPL

B 20 AMZ C03

C02: 40xAMZ 20xSSG 50xMSF
C04: 40xIBM
C01: 10xAMZ
C06: 20xIBM
C05: 10xAPL

B 20 AMZ C05

C02: 40xAMZ 20xSSG 50xMSF
C04: 40xIBM
C01: 10xAMZ
C06: 20xIBM
C05: 10xAPL 20xAMZ

B 50 IBM C01

C02: 40xAMZ 20xSSG 50xMSF
C04: 40xIBM
C01: 10xAMZ 50xIBM
C06: 20xIBM
C05: 10xAPL 20xAMZ

...

Operaciones no ejecutadas:
B 20 AMZ C03
B 20 APL C02
S 20 APL C05