Implementación de un banco

Queremos implementar una aplicación para gestionar un **banco**. El banco contendrá una **lista de clientes**, donde cada **cliente** lleva asociado un *NIF* y una *lista* de **cuentas bancarias**. Cada cuenta bancaria consta de un *número de cuenta* y un *saldo*. La lista de clientes *estará ordenada por los NIF de los clientes*, mientras que la lista de cuentas de cada cliente *estará ordenada por su número de cuenta.*

Las operaciones que se pueden realizar en el banco son las siguientes:

- **Realizar un ingreso**. Dado el NIF de un cliente, un número de cuenta y una cantidad: si no existe el NIF en el banco, entonces se añade un cliente nuevo con ese NIF, y una única cuenta con dicho número y el saldo igual a la cantidad a ingresar. En caso de que exista el cliente, es decir, exista el NIF, entonces:
  - Si existe el número de cuenta se hace el ingreso añadiendo la cantidad al saldo actual de la cuenta.
  - Si no existe el número de cuenta se crea una nueva cuenta para ese cliente con saldo inicial igual a la cantidad a ingresar.
- **Realizar un reintegro**. Dado el NIF de un cliente, un número de cuenta y una cantidad: si existe el NIF en el banco, actualizamos el saldo de esa cuenta restándole la cantidad a sacar. Hay que tener en cuenta que:
  - Si al hacer el reintegro el saldo de la cuenta es menor o igual a 0, debemos eliminar la cuenta bancaria de ese cliente. Si al eliminar la cuenta al cliente, este cliente se queda sin cuentas en el banco, entonces se procederá a eliminar el cliente.
- **Mostrar los clientes del banco.** Muestra la información de los clientes del banco. El formato podrás verlo en el ejemplo de ejecución.

La aplicación comenzará cargando los clientes del banco de la entrada estándar (`cin`), que en ese momento contendrá información sobre las cuentas de los clientes. La primera línea indicará cuántas cuentas se describen a continuación, y en las siguientes líneas aparece esa información. Por ejemplo:

```
11
20A 1 2000
10B 5 1000
20A 3 4000
10B 3 1000
20A 2 10000
30C 3 4000
40X 8 20000
40Z 1 20000
30K 5 14000
40Z 5 30000
30K 4 25000
```

Observa por ejemplo que el cliente de NIF `20A` tiene asociadas tres cuentas bancarias. La primera con número de cuenta `1` y saldo `2000`, la segunda con número de cuenta `3` y saldo `4000` y la tercera con número de cuenta `2` y saldo `10000`. La información no está ordenada. Tendrás que ordenarla tú. Por otro lado tendrás también que agrupar todas las cuentas bancarias de un cliente en su lista asociada.

Para implementar la aplicación necesitarás implementar las clases que se describen a continuación.

Clase CuentaBancaria

Contiene como atríbutos un número de cuenta (`int`) y un saldo (`int`). Ofrece los siguientes métodos y constructoras públicas:

- `CuentaBancaria()`: constructora por defecto.
- `CuentaBancaria(nc)`: constructora con un argumento para inicializar el número de cuenta. El saldo inicial será `0`.
- `~CuentaBancaria()`: destructora.
- `cargar_cuenta()`: carga la información de la cuenta de la entrada estándar, `cin`.
- `mostrar_cuenta()`, muestra por consola la información de la cuenta. Por ejemplo:

```
Numero de cuenta: 3    Saldo: 1000
```

- `num_cuenta()`: devuelve el número de cuenta.
- `saldo()`: devuelve el saldo.
- `pon_saldo(s)`: pone como saldo actual de la cuenta el valor `s`.

Clase Cliente

Un cliente estará identificado por su NIF, y contendrá una lista ordenada (por el número de cuenta) de **punteros** a cuentas bancarias. La lista tendrá tamaño máximo `MAX_CUENTAS=5`. Además contendrá un atributo que contiene el saldo total de todas las cuentas del cliente. Esta clase ofrece los siguientes métodos públicos y constructoras:

- `Cliente()`: constructora por defecto. El número de cuentas será `0`.
- `Cliente(NIF)`: inicializa el NIF del cliente. Su número de cuentas será `0`.
- `~Cliente()`: libera la memoria creada por el cliente.
- `cargar_cuenta()`: carga (desde la entrada estándar) la información de una única cuenta del cliente, y la inserta de forma ordenada en su lista de cuentas.
- `mostrar_cliente()`: muestra por consola la información de un cliente. Por ejemplo:

```
NIF del cliente: 20A
Saldo total del cliente: 16000
  Relacion de cuentas:
         Numero de cuenta: 1    Saldo: 2000
         Numero de cuenta: 2    Saldo: 10000
         Numero de cuenta: 3    Saldo: 4000
```

- `nif()`: devuelve el valor del NIF del cliente.
- `num_cuentas()`: devuelve el número de cuentas del cliente.
- `buscar_cuenta(numC, pos)`: devuelve `true` si y solo si el cliente contiene el número de cuenta `numC`. Además en `pos` devuelve la posición en la que está dicha cuenta. En caso de que no exista la cuenta, devuelve `false` y en `pos` la posición en la que debería estar dicha cuenta.
- `realizar_ingreso(pos, cantidad)`: añade `cantidad` al saldo de la cuenta situada en la posición `pos` de la lista de cuentas del cliente.
- `realizar_reintegro(pos, cantidad)`: decrementa en `cantidad` el saldo de la cuenta que ocupa la posición `pos` de la lista de cuentas del cliente.
- `nueva_cuenta(nc, saldo)`: añade una nueva cuenta con número de cuenta `nc` y saldo `saldo` a la lista de cuentas del cliente.
- `eliminar_cuenta(pos)`: elimina la cuenta que ocupa la posición `pos`.
- `saldo(pos)`: devuelve el saldo de la cuenta que ocupa la posición `pos`.

Clase Banco

El banco contiene la lista de clientes implementada a través de un **array dinámico de punteros** a clientes. El tamaño inicial del array será `1` (aunque luego aumentará siempre que haga falta). Ofrece los siguientes métodos y constructoras públicas:

- `Banco()`: constructora por defecto.
- `~Banco()`: destruye la memoria creada por el banco.
- `cargar_clientes()`: carga la lista de clientes de la entrada estándar (`cin`).
- `buscar_cliente(NIF, pos)`: devuelve `true` si y solo si el cliente con `NIF` existe en la lista. En `pos` devuelve la posición en la que se encuentra el cliente. Si el cliente no existe, entonces devuelve `false` y la posición en la que debería aparecer dicho cliente.
- `insertar_cliente(NIF, pos)`: devuelve `false` si el cliente con `NIF` no existe en la lista. Además lo inserta de forma ordenada y devuelve en `pos` la posición en la que lo ha insertado. En caso de que exista, devuelve `true`, no lo inserta, y en `pos` devuelve la posición en la que se encuentra el cliente.
- `eliminar_cliente(pos)`: elimina de la lista el cliente que ocupa la posición `pos`.
- `cliente(pos)`: Devuelve una referencia al cliente que ocupa la posición `pos`.
- `mostrar_clientes`: muestra por consola la información de los clientes del banco. Por ejemplo:

```
NIF del cliente: 10B
Saldo total del cliente: 2000
  Relacion de cuentas:
         Numero de cuenta: 3    Saldo: 1000
         Numero de cuenta: 5    Saldo: 1000

NIF del cliente: 20A
Saldo total del cliente: 16000
  Relacion de cuentas:
         Numero de cuenta: 1    Saldo: 2000
         Numero de cuenta: 2    Saldo: 10000
         Numero de cuenta: 3    Saldo: 4000

NIF del cliente: 30C
Saldo total del cliente: 4000
  Relacion de cuentas:
         Numero de cuenta: 3    Saldo: 4000

NIF del cliente: 30K
Saldo total del cliente: 39000
  Relacion de cuentas:
         Numero de cuenta: 4    Saldo: 25000
         Numero de cuenta: 5    Saldo: 14000

NIF del cliente: 40X
Saldo total del cliente: 20000
  Relacion de cuentas:
         Numero de cuenta: 8    Saldo: 20000

NIF del cliente: 40Z
Saldo total del cliente: 50000
  Relacion de cuentas:
         Numero de cuenta: 1    Saldo: 20000
         Numero de cuenta: 5    Saldo: 30000
```

Función main

En la función `main()` primero se cargará la información sobre todas las cuentas de los clientes y después se ejecutarán las operaciones que aparezcan en la entrada estándar.

Las operaciones pueden ser:

- `INGRESAR NIF cuenta cantidad`. Si no existe un cliente con ese `NIF` en el bando, se añade con una cuenta con identificador `cuenta` y saldo `cantidad`. Si el cliente existe, pero no tiene una cuenta con ese identificador, se le añadirá esa cuenta. En otro caso, se añadirá la cantidad a la cuenta existente.

- `SACAR NIF cuenta cantidad`. Si no existe un cliente en el banco con ese `NIF` se mostrará una línea con el mensaje `ERROR: NO EXISTE EL CLIENTE` seguido del `NIF`. Si existe el cliente pero no tiene una cuenta con identificador `cuenta` se mostrará en una línea el

mensaje de error `ERROR: EL CLIENTE NIF NO TIENE UNA CUENTA cuenta`, donde el `NIF` y la `cuenta` serán las particulares de esta operación. Tras los mensajes de error se mostrará una línea vacía.

Si el cliente y la cuenta existen, se restará `cantidad` a su saldo. Si la `cantidad` es mayor o igual al saldo actual de la cuenta, se eliminará la cuenta de la lista de cuentas de este cliente. Si tras esta eliminación el cliente se queda sin cuentas, se eliminará al cliente de la lista de clientes del banco.

- `MOSTRAR_CLIENTES`. Muestra por la salida estándar (`cout`) la información de todos los clientes, utilizando el método `mostrar_clientes()` de la clase `Banco`.
- `SALIR`. Termina el programa.

Ejemplo de ejecución

Como parte del material del examen tienes un fichero `entrada.txt` con un ejemplo de entrada: primero la información de las cuentas y luego una serie de operaciones. El fichero `salida.txt` contiene la salida que tiene que mostrarse al ejecutar el programa con los datos del fichero de entrada.

Las siguientes instrucciones que aparecen en la función `main()` hacen que cuando hagas las pruebas locales la entrada estándar (`cin`) lea en realidad del fichero `clientes.txt`, en vez de leer del teclado:

```
  // Si estás ejecutando el programa en tu ordenador, las siguientes líneas
  // redirigirán cualquier lectura de cin al fichero 'entrada.txt'. Esto es
  // útil para no tener que teclear los casos de prueba por teclado cada vez
  // que ejecutas el programa.
  //
  // Si prefieres teclear la entrada por teclado en tu ordenador,
  // comenta las líneas comprendidas entre los #ifndef y #endif
#ifndef DOMJUDGE
  std::ifstream in("entrada.txt");
  auto cinbuf = std::cin.rdbuf(in.rdbuf());
#endif
```