1. Imagine que está escribiendo un programa para manejar listas de compras. Defina el tipo **ShoppingList**, con capacidad de mantener entidades del tipo **ShoppingItem** que constan de un nombre de ítem (**std::string**) y su cantidad asociada (**double**). En la interfase de **ShoppingList** defina e implemente métodos para:

```
// Agregar items a la lista. Retorna la cantidad integrada
// resultante para ese ítem:
double addItem(const string &itemName, double cant);

// recupera la cantidad asociada a un ítem determinado
double getItem(const string &itemName);

// elimina un ítem dado. Retorna true/false de acuerdo al resultado
bool removeItem(const string &itemName);

// Imprime la ShoppingList sobre la salida dada
void print(ostream &out);
```

   Implemente la interfaz solicitada de una manera independiente al tipo de contenedor utilizado. Verifique que puede cambiar un **std::vector** por un **std::list** sin afectar una aplicación de usuario ni tampoco la implementación de la interfase. Más adelante veremos que hay contenedores más adecuados para este tipo de requerimientos (contenedores asociativos).

   Pruebe todo el diseño desde una función main.

   Se anima a generalizar el diseño anterior de manera que el **ShoppingList** sea una clase template parametrizada en el tipo contenedor a utilizar y función de comparación para comparar ítems dentro del contenedor.

2. Modifique el problema 1 haciendo:
   - Agregue el tipo **ShoppingItem** el operador **double operator ()** que termine retornando la cantidad asociada a un ítem determinado.
   - Defina un tipo de datos, interno a **ShoppingList**, que se comporte como un iterador sobre ítems de la lista.
   - De interfaz de **begin()**, **end()** para iniciar un recorrido del iterador.
3. Diseñe un tipo adecuado para mantener una lista de precios (debe mantener un conjunto de pares **<itemName, precio>**).
4. Diseñe un tipo **Cajero** con miembros que permitan calcular el valor total de una compra dada una lista de precios (manejada internamente por el **Cajero**) y una instancia de **ShoppingList**.

5. Instancie STL deque. Ejercite los métodos disponibles.
6. Instancie un STL queue utilizando list y utilizando deque como underlying container.
7. Ejercite los métodos disponibles sobre una queue.