```
#pragma once

#include <algorithm> // for sort
#include <cstdlib> // for NULL
#include <iostream> // for operator<<
#include <string>
#include <vector>
using namespace std;

struct Item
{
    Item (string newName="noname", unsigned int newValue=0, unsigned int newQuantity=0)
    : name(newName), value(newValue), quantity(newQuantity)
    {
    }

    friend ostream& operator<< (ostream& outs, const Item& src)
    {
        outs << src.name;
        return outs;
    }

    string name;
    unsigned int value;
    unsigned int quantity;
};

class TreasureChest
{
    public:

        /*
         * Add an item to the end of the chest.
         * @param newItem the item to be added to the end of the chest
         */
        void addItem (const Item& newItem);

        /*
         * Insert an item at the specified zero-indexed position in the chest.
         * If position is not valid for the chest, add the item to
         * the end of the chest.
         * @param newItem the item to be inserted into the chest
         * @param position the zero-indexed position where the insertion
         *        is to take place
         */
        void insertItem (const Item& newItem, unsigned int position);

        /*
         * Get a pointer to an item at a specified zero-indexed position in the chest.
         * @param position the zero-indexed position of the item
```

```
51          * @return a pointer to the item if position is valid, else NULL
52          */
53         const Item* getItem (unsigned int position);
54
55         /*
56          * Remove an item from the chest at a specified zero-indexed position.
57          * @param position the zero-indexed position of the item
58          * @return a copy of the Item removed from the chest
59          * @throw string("ERROR: attempting remove at invalid position") if
60          *        position is not valid
61          */
62         Item removeItem (unsigned int position) throw (string);
63
64         /*
65          * Clear the chest of all items.
66          */
67         void clear ();
68
69         /*
70          * Check to see if the chest is empty.
71          * @return true if the chest is empty, else false
72          */
73         bool empty () const;
74
75         /*
76          * Get the size/number of items currently in the chest.
77          * @return an unsigned integer containing the current size of the chest
78          */
79         unsigned int getSize () const;
80
81         /*
82          * Sort the items in the chest by name, using the sort function
83          * from the C++ standard algorithm library.
84          */
85         void sortByName ();
86
87         /*
88          * Sort the items in the chest by value, using the sort function
89          * from the C++ standard algorithm library.
90          */
91         void sortByValue ();
92
93         /*
94          * Sort the items in the chest by quantity, using the sort function
95          * from the C++ standard algorithm library.
96          */
97         void sortByQuantity ();
98
99         /*
100         * Place the names of the items in the chest on the specified stream,
```

```
         * formatted as ITEM_NAME,ITEM_NAME,...ITEM_NAME
         */
        friend ostream& operator<< (ostream& outs, const TreasureChest& src);

    private:

        vector<Item> chest;
};

/*
 * Compare two items by name.
 * @return true if lsrc.name < rsrc.name, else false
 */
bool compareItemsByName (const Item& lsrc, const Item& rsrc);

/*
 * Compare two items by value.
 * @return true if lsrc.value < rsrc.value, else false
 */
bool compareItemsByValue (const Item& lsrc, const Item& rsrc);

/*
 * Compare two items by quantity.
 * @return true if lsrc.quantity < rsrc.quantity, else false
 */
bool compareItemsByQuantity (const Item& lsrc, const Item& rsrc);

```