```cpp
/*
 * Programming Challenge 15
 */
#include <cassert>
#include <iostream>
#include <string>
using namespace std;

class ShoppingList
{
    public:

        /*
         * Default constructor.
         * Initializes numItems to 25 and items to size numItems.
         */
        ShoppingList ();

        /*
         * Overloaded constructor.
         * Initializes numItems to newNumItems and items to size numItems.
         * @param newNumItems an unsigned integer containing the desired size of
         *        the items array
         */
        ShoppingList (unsigned int newNumItems);

        /*
         * Destructor.
         * Frees the memory associated with items.
         */
        virtual ~ShoppingList ();

        /*
         * Get the number of items in this ShoppingList.
         * @return an unsigned integer containing the number of items
         *         in this ShoppingList
         */
        unsigned int getNumItems () const;

        /*
         * Retrieve the value of the item at a specified index in the array.
         * @param index an unsigned integer containing the zero-based index
         *        of the item desired
         * @return if index is valid, a string containing the value of the item
         *         at the specified index
         * @throw ArrayException with the message "INVALID ARRAY INDEX" if index is invalid
         */
        string getItem (unsigned int index) const;

        /*
```

```
51          * Place an item as a specified index in the array.
52          * @param theItem a string containing the item to be placed in the array
53          * @param index an unsigned integer containing the zero-based index of the location
54          *        in the array to place theItem
55          * @return true if the location at index is unoccupied (empty string) and theItem
56          *         is successfully stored in the array; false if there is already an item at index
57          * @throw ArrayException with the message "INVALID ARRAY INDEX" if index is invalid
58          */
59         bool setItem (string theItem, unsigned int index);
60
61         /*
62          * Reset the items array by freeing its associated memory and re-allocating to a
63          * specified size.
64          * @param newSize an unsigned integer containing the desired size of items
65          * @throw ArrayException with the message "INVALID ARRAY SIZE" if newSize is less than 1
66          */
67         void reset (unsigned int newSize);
68
69     private:
70
71         string* items;
72         unsigned int numItems;
73 };
74
75 /* for unit testing -- do not alter */
76 struct ArrayException
77 {
78     ArrayException (string newMessage="error")
79     : message(newMessage)
80     {
81     }
82
83     string message;
84 };
85
86 template <typename X, typename A>
87 void btassert(A assertion);
88 void unittest ();
89
90 int main (int argc, char* argv[])
91 {
92     unittest();
93
94     return 0;
95 }
96
97 // CODE HERE -- FUNCTION DEFINITIONS
98
99 ShoppingList::ShoppingList ()
100 {
```

```
    numItems    = 25;
    items       = new string[numItems];
}

ShoppingList::ShoppingList (unsigned int newNumItems)
{
    numItems    = newNumItems;
    items       = new string[numItems];
}

ShoppingList::~ShoppingList ()
{
    delete [] items;
}

unsigned int ShoppingList::getNumItems () const
{
    return numItems;
}

string ShoppingList::getItem (unsigned int index) const
{
    if( index < numItems )
        return items[index];
    else
        throw ArrayException( "INVALID ARRAY INDEX" );
}

bool ShoppingList::setItem (string theItem, unsigned int index)
{
    if( index < numItems )
    {
        if( items[index] == "" )
        {
            items[index] = theItem;
            return true;
        } else {
            return false;
        }
    } else {
        throw ArrayException( "INVALID ARRAY INDEX" );
    }
}

void ShoppingList::reset (unsigned int newSize)
{
    if( newSize < 1 )
    {
        throw ArrayException( "INVALID ARRAY SIZE" );
    } else {
```

```
        delete [] items;

        numItems    = newSize;
        items       = new string[newSize];
    }
}

/*
 * Unit testing functions. Do not alter.
 */

void unittest ()
{
    cout << "\nSTARTING UNIT TEST\n\n";

    ShoppingList* myList = new ShoppingList;

    cout << "* DEFAULT CONSTRUCTOR *\n\n";

    try {
        btassert<bool>(myList->getNumItems() == 25);
        cout << "Passed TEST 1: ShoppingList::getNumItems () \n";
    } catch (bool b) {
        cout << "# FAILED TEST 1: ShoppingList::getNumItems () #\n";
    }

    try {
        btassert<bool>(myList->getItem(0) == "");
        cout << "Passed TEST 2: ShoppingList::getItem (0) \n";
    } catch (bool b) {
        cout << "# FAILED TEST 2: ShoppingList::getItem (0) #\n";
    }

    try {
        myList->getItem(25);
    } catch (ArrayException e) {
        try {
            btassert<bool>(e.message == "INVALID ARRAY INDEX");
            cout << "Passed TEST 3: ShoppingList::getItem(25) EXCEPTION HANDLING \n";
        } catch (bool b) {
            cout << "# FAILED TEST 3: ShoppingList::getItem(25) EXCEPTION HANDLING #\n";
        }
    }

    try {
        btassert<bool>(myList->setItem("apples", 0) == true);
        cout << "Passed TEST 4: ShoppingList::setItem(\"apples\", 0) \n";
    } catch (bool b) {
        cout << "# FAILED TEST 4: ShoppingList::setItem(\"apples\", 0) #\n";
    }
```

```
    try {
        btassert<bool>(myList->getItem(0) == "apples");
        cout << "Passed TEST 5: ShoppingList::getItem(0) \n";
    } catch (bool b) {
        cout << "# FAILED TEST 5: ShoppingList::getItem(0) #\n";
    }

    try {
        btassert<bool>(myList->setItem("oranges", 0) == false);
        cout << "Passed TEST 6: ShoppingList::setItem(\"oranges\", 0) \n";
    } catch (bool b) {
        cout << "# FAILED TEST 6: ShoppingList::setItem(\"oranges\", 0) #\n";
    }

    try {
        btassert<bool>(myList->getItem(0) == "apples");
        cout << "Passed TEST 7: ShoppingList::getItem(0) \n";
    } catch (bool b) {
        cout << "# FAILED TEST 7: ShoppingList::getItem(0) #\n";
    }

    try {
        myList->setItem("oranges", 25);
    } catch (ArrayException e) {
        try {
            btassert<bool>(e.message == "INVALID ARRAY INDEX");
            cout << "Passed TEST 8: ShoppingList::setItem(\"oranges\", 25) EXCEPTION HANDLING \n";
        } catch (bool b) {
            cout << "# FAILED TEST 8: ShoppingList::setItem(\"oranges\", 25) EXCEPTION HANDLING #\n";
        }
    }

    try {
        myList->reset(0);
    } catch (ArrayException e) {
        try {
            btassert<bool>(e.message == "INVALID ARRAY SIZE");
            cout << "Passed TEST 9: ShoppingList::reset(0) EXCEPTION HANDLING \n";
        } catch (bool b) {
            cout << "# FAILED TEST 9: ShoppingList::reset(0) EXCEPTION HANDLING #\n";
        }
    }

    try {
        btassert<bool>(myList->getNumItems() == 25);
        cout << "Passed TEST 10: ShoppingList::getNumItems () \n";
    } catch (bool b) {
        cout << "# FAILED TEST 10: ShoppingList::getNumItems () #\n";
    }
```

```
    try {
        btassert<bool>(myList->getItem(0) == "apples");
        cout << "Passed TEST 11: ShoppingList::getItem(0) \n";
    } catch (bool b) {
        cout << "# FAILED TEST 11: ShoppingList::getItem(0) #\n";
    }

    try {
        btassert<bool>(myList->getNumItems() == 25);
        cout << "Passed TEST 12: ShoppingList::getNumItems () \n";
    } catch (bool b) {
        cout << "# FAILED TEST 12: ShoppingList::getNumItems () #\n";
    }

    try {
        btassert<bool>(myList->getItem(0) == "apples");
        cout << "Passed TEST 13: ShoppingList::getItem(0) \n";
    } catch (bool b) {
        cout << "# FAILED TEST 13: ShoppingList::getItem(0) #\n";
    }

    myList->reset(1);

    try {
        btassert<bool>(myList->getNumItems() == 1);
        cout << "Passed TEST 14: ShoppingList::getNumItems () \n";
    } catch (bool b) {
        cout << "# FAILED TEST 14: ShoppingList::getNumItems () #\n";
    }

    try {
        btassert<bool>(myList->setItem("apples", 0) == true);
        cout << "Passed TEST 15: ShoppingList::setItem(\"apples\", 0) \n";
    } catch (bool b) {
        cout << "# FAILED TEST 15: ShoppingList::setItem(\"apples\", 0) #\n";
    }

    try {
        btassert<bool>(myList->getItem(0) == "apples");
        cout << "Passed TEST 16: ShoppingList::getItem(0) \n";
    } catch (bool b) {
        cout << "# FAILED TEST 16: ShoppingList::getItem(0) #\n";
    }

    try {
        btassert<bool>(myList->setItem("oranges", 0) == false);
        cout << "Passed TEST 17: ShoppingList::setItem(\"oranges\", 0) \n";
    } catch (bool b) {
        cout << "# FAILED TEST 17: ShoppingList::setItem(\"oranges\", 0) #\n";
```

```
    }

    try {
        btassert<bool>(myList->getItem(0) == "apples");
        cout << "Passed TEST 18: ShoppingList::getItem(0) \n";
    } catch (bool b) {
        cout << "# FAILED TEST 18: ShoppingList::getItem(0) #\n";
    }

    try {
        myList->setItem("oranges", 25);
    } catch (ArrayException e) {
        try {
            btassert<bool>(e.message == "INVALID ARRAY INDEX");
            cout << "Passed TEST 19: ShoppingList::setItem(\"oranges\", 25) EXCEPTION HANDLING \n";
        } catch (bool b) {
            cout << "# FAILED TEST 19: ShoppingList::setItem(\"oranges\", 25) EXCEPTION HANDLING #\n";
        }
    }

    delete myList;
    myList = new ShoppingList(1);

    cout << "\n* OVERLOADED CONSTRUCTOR *\n\n";

    try {
        btassert<bool>(myList->getNumItems() == 1);
        cout << "Passed TEST 20: ShoppingList::getNumItems () \n";
    } catch (bool b) {
        cout << "# FAILED TEST 20: ShoppingList::getNumItems () #\n";
    }

    try {
        btassert<bool>(myList->setItem("apples", 0) == true);
        cout << "Passed TEST 21: ShoppingList::setItem(\"apples\", 0) \n";
    } catch (bool b) {
        cout << "# FAILED TEST 21: ShoppingList::setItem(\"apples\", 0) #\n";
    }

    try {
        btassert<bool>(myList->getItem(0) == "apples");
        cout << "Passed TEST 22: ShoppingList::getItem(0) \n";
    } catch (bool b) {
        cout << "# FAILED TEST 22: ShoppingList::getItem(0) #\n";
    }

    try {
        myList->setItem("oranges", 1);
    } catch (ArrayException e) {
        try {
```

```
            btassert<bool>(e.message == "INVALID ARRAY INDEX");
            cout << "Passed TEST 23: ShoppingList::setItem(\"oranges\", 1) EXCEPTION HANDLING \n";
        } catch (bool b) {
            cout << "# FAILED TEST 23: ShoppingList::setItem(\"oranges\", 1) EXCEPTION HANDLING #\n";
        }
    }

    try {
        myList->getItem(1);
    } catch (ArrayException e) {
        try {
            btassert<bool>(e.message == "INVALID ARRAY INDEX");
            cout << "Passed TEST 24: ShoppingList::getItem(1) EXCEPTION HANDLING \n";
        } catch (bool b) {
            cout << "# FAILED TEST 24: ShoppingList::getItem(1) EXCEPTION HANDLING #\n";
        }
    }

    cout << "\nUNIT TEST COMPLETE\n\n";
}

template <typename X, typename A>
void btassert (A assertion)
{
    if (!assertion)
        throw X();
}
```