

```

1  /*
2   * Programming Challenge 15
3   */
4  #include <cassert>
5  #include <iostream>
6  #include <string>
7  using namespace std;
8
9  class ShoppingList
10 {
11     public:
12
13         /*
14          * Default constructor.
15          * Initializes numItems to 25 and items to size numItems.
16          */
17         ShoppingList ();
18
19         /*
20          * Overloaded constructor.
21          * Initializes numItems to newNumItems and items to size numItems.
22          * @param newNumItems an unsigned integer containing the desired size of
23          *       the items array
24          */
25         ShoppingList (unsigned int newNumItems);
26
27         /*
28          * Destructor.
29          * Frees the memory associated with items.
30          */
31         virtual ~ShoppingList ();
32
33         /*
34          * Get the number of items in this ShoppingList.
35          * @return an unsigned integer containing the number of items
36          *       in this ShoppingList
37          */
38         unsigned int getNumItems () const;
39
40         /*
41          * Retrieve the value of the item at a specified index in the array.
42          * @param index an unsigned integer containing the zero-based index
43          *       of the item desired
44          * @return if index is valid, a string containing the value of the item
45          *       at the specified index
46          * @throw ArrayException with the message "INVALID ARRAY INDEX" if index is invalid
47          */
48         string getItem (unsigned int index) const;
49
50         /*

```

```

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    {

```

```

101     numItems    = 25;
102     items       = new string[numItems];
103 }
104
105 ShoppingList::ShoppingList (unsigned int newNumItems)
106 {
107     numItems    = newNumItems;
108     items       = new string[numItems];
109 }
110
111 ShoppingList::~ShoppingList ()
112 {
113     delete [] items;
114 }
115
116 unsigned int ShoppingList::getNumItems () const
117 {
118     return numItems;
119 }
120
121 string ShoppingList::getItem (unsigned int index) const
122 {
123     if( index < numItems )
124         return items[index];
125     else
126         throw ArrayException( "INVALID ARRAY INDEX" );
127 }
128
129 bool ShoppingList::setItem (string theItem, unsigned int index)
130 {
131     if( index < numItems )
132     {
133         if( items[index] == "" )
134         {
135             items[index] = theItem;
136             return true;
137         } else {
138             return false;
139         }
140     } else {
141         throw ArrayException( "INVALID ARRAY INDEX" );
142     }
143 }
144
145 void ShoppingList::reset (unsigned int newSize)
146 {
147     if( newSize < 1 )
148     {
149         throw ArrayException( "INVALID ARRAY SIZE" );
150     } else {

```

```

151         delete [] items;
152
153         numItems    = newSize;
154         items       = new string[newSize];
155     }
156 }
157
158 /*
159  * Unit testing functions. Do not alter.
160  */
161
162 void unittest ()
163 {
164     cout << "\nSTARTING UNIT TEST\n\n";
165
166     ShoppingList* myList = new ShoppingList;
167
168     cout << "* DEFAULT CONSTRUCTOR *\n\n";
169
170     try {
171         btassert<bool>(myList->getNumItems() == 25);
172         cout << "Passed TEST 1: ShoppingList::getNumItems () \n";
173     } catch (bool b) {
174         cout << "# FAILED TEST 1: ShoppingList::getNumItems () #\n";
175     }
176
177     try {
178         btassert<bool>(myList->getItem(0) == "");
179         cout << "Passed TEST 2: ShoppingList::getItem (0) \n";
180     } catch (bool b) {
181         cout << "# FAILED TEST 2: ShoppingList::getItem (0) #\n";
182     }
183
184     try {
185         myList->getItem(25);
186     } catch (ArrayException e) {
187         try {
188             btassert<bool>(e.message == "INVALID ARRAY INDEX");
189             cout << "Passed TEST 3: ShoppingList::getItem(25) EXCEPTION HANDLING \n";
190         } catch (bool b) {
191             cout << "# FAILED TEST 3: ShoppingList::getItem(25) EXCEPTION HANDLING #\n";
192         }
193     }
194
195     try {
196         btassert<bool>(myList->setItem("apples", 0) == true);
197         cout << "Passed TEST 4: ShoppingList::setItem(\"apples\", 0) \n";
198     } catch (bool b) {
199         cout << "# FAILED TEST 4: ShoppingList::setItem(\"apples\", 0) #\n";
200     }

```

```
201
202     try {
203         btassert<bool>(myList->getItem(0) == "apples");
204         cout << "Passed TEST 5: ShoppingList::getItem(0) \n";
205     } catch (bool b) {
206         cout << "# FAILED TEST 5: ShoppingList::getItem(0) #\n";
207     }
208
209     try {
210         btassert<bool>(myList->setItem("oranges", 0) == false);
211         cout << "Passed TEST 6: ShoppingList::setItem(\"oranges\", 0) \n";
212     } catch (bool b) {
213         cout << "# FAILED TEST 6: ShoppingList::setItem(\"oranges\", 0) #\n";
214     }
215
216     try {
217         btassert<bool>(myList->getItem(0) == "apples");
218         cout << "Passed TEST 7: ShoppingList::getItem(0) \n";
219     } catch (bool b) {
220         cout << "# FAILED TEST 7: ShoppingList::getItem(0) #\n";
221     }
222
223     try {
224         myList->setItem("oranges", 25);
225     } catch (ArrayException e) {
226         try {
227             btassert<bool>(e.message == "INVALID ARRAY INDEX");
228             cout << "Passed TEST 8: ShoppingList::setItem(\"oranges\", 25) EXCEPTION HANDLING \n";
229         } catch (bool b) {
230             cout << "# FAILED TEST 8: ShoppingList::setItem(\"oranges\", 25) EXCEPTION HANDLING #\n";
231         }
232     }
233
234     try {
235         myList->reset(0);
236     } catch (ArrayException e) {
237         try {
238             btassert<bool>(e.message == "INVALID ARRAY SIZE");
239             cout << "Passed TEST 9: ShoppingList::reset(0) EXCEPTION HANDLING \n";
240         } catch (bool b) {
241             cout << "# FAILED TEST 9: ShoppingList::reset(0) EXCEPTION HANDLING #\n";
242         }
243     }
244
245     try {
246         btassert<bool>(myList->getNumItems() == 25);
247         cout << "Passed TEST 10: ShoppingList::getNumItems () \n";
248     } catch (bool b) {
249         cout << "# FAILED TEST 10: ShoppingList::getNumItems () #\n";
250     }
```

```
251
252     try {
253         btassert<bool>(myList->getItem(0) == "apples");
254         cout << "Passed TEST 11: ShoppingList::getItem(0) \n";
255     } catch (bool b) {
256         cout << "# FAILED TEST 11: ShoppingList::getItem(0) #\n";
257     }
258
259     try {
260         btassert<bool>(myList->getNumItems() == 25);
261         cout << "Passed TEST 12: ShoppingList::getNumItems () \n";
262     } catch (bool b) {
263         cout << "# FAILED TEST 12: ShoppingList::getNumItems () #\n";
264     }
265
266     try {
267         btassert<bool>(myList->getItem(0) == "apples");
268         cout << "Passed TEST 13: ShoppingList::getItem(0) \n";
269     } catch (bool b) {
270         cout << "# FAILED TEST 13: ShoppingList::getItem(0) #\n";
271     }
272
273     myList->reset(1);
274
275     try {
276         btassert<bool>(myList->getNumItems() == 1);
277         cout << "Passed TEST 14: ShoppingList::getNumItems () \n";
278     } catch (bool b) {
279         cout << "# FAILED TEST 14: ShoppingList::getNumItems () #\n";
280     }
281
282     try {
283         btassert<bool>(myList->setItem("apples", 0) == true);
284         cout << "Passed TEST 15: ShoppingList::setItem(\"apples\", 0) \n";
285     } catch (bool b) {
286         cout << "# FAILED TEST 15: ShoppingList::setItem(\"apples\", 0) #\n";
287     }
288
289     try {
290         btassert<bool>(myList->getItem(0) == "apples");
291         cout << "Passed TEST 16: ShoppingList::getItem(0) \n";
292     } catch (bool b) {
293         cout << "# FAILED TEST 16: ShoppingList::getItem(0) #\n";
294     }
295
296     try {
297         btassert<bool>(myList->setItem("oranges", 0) == false);
298         cout << "Passed TEST 17: ShoppingList::setItem(\"oranges\", 0) \n";
299     } catch (bool b) {
300         cout << "# FAILED TEST 17: ShoppingList::setItem(\"oranges\", 0) #\n";
```

```

301     }
302
303     try {
304         btassert<bool>(myList->getItem(0) == "apples");
305         cout << "Passed TEST 18: ShoppingList::getItem(0) \n";
306     } catch (bool b) {
307         cout << "# FAILED TEST 18: ShoppingList::getItem(0) #\n";
308     }
309
310     try {
311         myList->setItem("oranges", 25);
312     } catch (ArrayException e) {
313         try {
314             btassert<bool>(e.message == "INVALID ARRAY INDEX");
315             cout << "Passed TEST 19: ShoppingList::setItem(\"oranges\", 25) EXCEPTION HANDLING \n";
316         } catch (bool b) {
317             cout << "# FAILED TEST 19: ShoppingList::setItem(\"oranges\", 25) EXCEPTION HANDLING #\n";
318         }
319     }
320
321     delete myList;
322     myList = new ShoppingList(1);
323
324     cout << "\n* OVERLOADED CONSTRUCTOR *\n\n";
325
326     try {
327         btassert<bool>(myList->getNumItems() == 1);
328         cout << "Passed TEST 20: ShoppingList::getNumItems () \n";
329     } catch (bool b) {
330         cout << "# FAILED TEST 20: ShoppingList::getNumItems () #\n";
331     }
332
333     try {
334         btassert<bool>(myList->setItem("apples", 0) == true);
335         cout << "Passed TEST 21: ShoppingList::setItem(\"apples\", 0) \n";
336     } catch (bool b) {
337         cout << "# FAILED TEST 21: ShoppingList::setItem(\"apples\", 0) #\n";
338     }
339
340     try {
341         btassert<bool>(myList->getItem(0) == "apples");
342         cout << "Passed TEST 22: ShoppingList::getItem(0) \n";
343     } catch (bool b) {
344         cout << "# FAILED TEST 22: ShoppingList::getItem(0) #\n";
345     }
346
347     try {
348         myList->setItem("oranges", 1);
349     } catch (ArrayException e) {
350         try {

```

```

351         btassert<bool>(e.message == "INVALID ARRAY INDEX");
352         cout << "Passed TEST 23: ShoppingList::setItem(\"oranges\", 1) EXCEPTION HANDLING \n" ;
353     } catch (bool b) {
354         cout << "# FAILED TEST 23: ShoppingList::setItem(\"oranges\", 1) EXCEPTION HANDLING #\n" ;
355     }
356 }
357
358 try {
359     myList->getItem(1);
360 } catch (ArrayException e) {
361     try {
362         btassert<bool>(e.message == "INVALID ARRAY INDEX");
363         cout << "Passed TEST 24: ShoppingList::getItem(1) EXCEPTION HANDLING \n" ;
364     } catch (bool b) {
365         cout << "# FAILED TEST 24: ShoppingList::getItem(1) EXCEPTION HANDLING #\n" ;
366     }
367 }
368
369 cout << "\nUNIT TEST COMPLETE\n\n";
370 }
371
372 template <typename X, typename A>
373 void btassert (A assertion)
374 {
375     if (!assertion)
376         throw X();
377 }
378

```