

```

1  /*
2   * Programming Challenge 4
3   */
4  #include <cassert>
5  #include <cstdlib>
6  #include <iostream>
7  #include <sstream>
8  using namespace std;
9
10 string makeString (string label, double value, char separator);
11 char stringToChar (string value);
12 int stringToInt (string value);
13 double stringToDouble (string value);
14 bool stringToBool (string value);
15
16 /* helper functions -- do not alter */
17 void clearScreen ();
18
19 /* for unit testing -- do not alter */
20 template <typename X, typename A>
21 void btassert(A assertion);
22 void interactiveTest ();
23 void unittest ();
24
25 int main (int argc, char* argv[])
26 {
27     string input;
28     cout << "[1] Do interactive test, [2] Skip to unit test: ";
29     getline(cin, input);
30     if (stringToInt(input) == 1)
31         interactiveTest();
32
33     unittest();
34
35     return 0;
36 }
37
38
39 /*
40  * Return a string comprised of a label, followed by a space,
41  * followed by a separator character, followed by a space, followed
42  * by a floating-point value. For example, label="Temperature", value=41.7,
43  * separator=':' will return "Temperature : 41.7". Uses stringstream.
44  * @param label the label for the value
45  * @param value a double containing the value associated with the label
46  * @param separator the character that will separate the label and the value
47  * @return a string comprised of a label, followed by a space,
48  *         followed by a separator character, followed by a space, followed
49  *         by a floating-point value
50  */

```

```

51 string makeString (string label, double value, char separator)
52 {
53     stringstream ss;
54
55     ss << label << " " << separator << " " << value;
56
57     return ss.str();
58 }
59
60 /*
61  * Useful when accepting input from stdin using the getline function.
62  * Return the first character of a length 1 string. If the value is of
63  * length 0 or of length > 1, return the null character ('\0').
64  * @param value a string containing an expected single character
65  * @return the first character of the string or null character ('\0')
66  *         when value is length 0 or value is length > 1
67  */
68 char stringToChar (string value)
69 {
70
71     if( value.length() == 0 || value.length() > 1 ) { return '\0'; }
72
73     return value.at(0);
74
75 }
76
77 /*
78  * Useful when accepting input from stdin using the getline function.
79  * Convert a string containing an expected integer value (such
80  * as a string captured from stdin) into an integer. If value is
81  * not valid as an integer, return 0.
82  * @param value a string containing an expected integer value
83  * @return an integer representing the value, or 0 on failure
84  */
85 int stringToInt (string value)
86 {
87     // THIS FUNCTION PROVIDED AS AN EXAMPLE
88     int ivalue = 0;
89     stringstream converter(value);
90     converter.exceptions(ios_base::failbit);
91
92     try
93     {
94         converter >> ivalue;
95     }
96     catch (ios_base::failure f) {}
97
98     return ivalue;
99 }
100

```

```

101  /*
102  * Useful when accepting input from stdin using the getline function.
103  * Convert a string containing an expected floating-point value (such
104  * as a string captured from stdin) into a double. If value is
105  * not valid as an double, return 0.
106  * @param value a string containing an expected floating-point value
107  * @return an double representing the value, or 0 on failure
108  */
109  double stringToDouble (string value)
110  {
111
112      double dblConvertedValue = 0.0;
113
114      stringstream converter( value );
115      converter.exceptions( ios_base::failbit );
116
117      try {
118
119          converter >> dblConvertedValue;
120
121      } catch( ios_base::failure f ) {    }
122
123      return dblConvertedValue;
124  }
125
126  /*
127  * Useful when accepting input from stdin using the getline function.
128  * Convert a string containing an boolean value (such
129  * as a string captured from stdin) into a bool. Return true if the first
130  * character is 'T' (case-insensitive), false if the first character is 'F'
131  * (case-insensitive), and false on anything else.
132  * @param value a string expected to start with either 'T' or 'F'
133  * @return an bool if the first character is 'T' (case-insensitive), false
134  *         if the first character is 'F' (case-insensitive), and false on
135  *         anything else.
136  */
137
138  bool stringToBool (string value)
139  {
140
141      if( value.length() == 0 )    {    return false;    } else {
142
143          char charDecision    = value.at(0);
144
145          if( charDecision == 't' || charDecision == 'T' )    {    return true;    } else {    return false;    }
146
147      }
148
149  }
150

```

```

151  /*
152  * Unit testing functions. Do not alter.
153  */
154
155  void interactiveTest ()
156  {
157      cout << "\nSTARTING INTERACTIVE TEST\n\n";
158
159      bool quit = false;
160      char c = 'z';
161      string input;
162
163      while (!quit)
164      {
165          cout << "! TRY EVERYTHING YOU CAN TO BREAK THESE ON BAD INPUT !\n\n";
166
167          cout << "Enter a char ('z' to stop interactive test): ";
168          getline(cin, input);
169          c = stringToChar(input);
170          if (c == '\0')
171              cout << input << " not a valid char\n";
172          else if (c == 'z')
173              break;
174          else
175              cout << "Char input: " << c << endl;
176
177          cout << "\nEnter an integer: ";
178          getline(cin, input);
179          cout << "Integer input: " << stringToInt(input) << endl;
180
181          cout << "\nEnter a double: ";
182          getline(cin, input);
183          cout << "Double input: " << stringToDouble(input) << endl;
184
185          cout << "\nEnter TRUE or FALSE: ";
186          getline(cin, input);
187          cout << "Boolean input: " << boolalpha << stringToBool(input) << endl;
188
189          cout << "\nHIT ENTER TO CONTINUE";
190          getline(cin, input);
191          clearScreen();
192      }
193
194      cout << "\nINTERACTIVE TEST COMPLETE\n\n";
195  }
196
197  void unittest ()
198  {
199      cout << "\nSTARTING UNIT TEST\n\n";
200

```

```

201     try {
202         btassert<bool>(makeString("Temperature", 42.6, ':') == "Temperature : 42.6");
203         cout << "Passed TEST 1: makeString(Temperature, 42.6, ':')\n";
204     } catch (bool b) {
205         cout << "# FAILED TEST 1 makeString(Temperature, 42.6, ':') #\n";
206     }
207
208     try {
209         btassert<bool>(makeString("", 75, ',') == " , 75");
210         cout << "Passed TEST 2: makeString(\"\", 75, ',')\n";
211     } catch (bool b) {
212         cout << "# FAILED TEST 2 makeString(\"\", 75, ',') #\n";
213     }
214
215     try {
216         btassert<bool>(makeString("Total", 100.05, '=') == "Total = 100.05");
217         cout << "Passed TEST 3: makeString(Total, 100.05, '=')\n";
218     } catch (bool b) {
219         cout << "# FAILED TEST 3 makeString(Total, 100.05, '=') #\n";
220     }
221
222     try {
223         btassert<bool>(stringToChar("") == '\0');
224         cout << "Passed TEST 4: stringToChar(\"\")\n";
225     } catch (bool b) {
226         cout << "# FAILED TEST 4 stringToChar(\"\") #\n";
227     }
228
229     try {
230         btassert<bool>(stringToChar("yn") == '\0');
231         cout << "Passed TEST 5: stringToChar(yn)\n";
232     } catch (bool b) {
233         cout << "# FAILED TEST 5 stringToChar(yn) #\n";
234     }
235
236     try {
237         btassert<bool>(stringToChar("X") == 'X');
238         cout << "Passed TEST 6: stringToChar(X)\n";
239     } catch (bool b) {
240         cout << "# FAILED TEST 6 stringToChar(X) #\n";
241     }
242
243     try {
244         btassert<bool>(stringToInt("42") == 42);
245         cout << "Passed TEST 7: stringToInt(42)\n";
246     } catch (bool b) {
247         cout << "# FAILED TEST 7 stringToInt(42) #\n";
248     }
249
250     try {

```

```

251     btassert<bool>(stringToInt("hello") == 0);
252     cout << "Passed TEST 8: stringToInt(hello)\n";
253 } catch (bool b) {
254     cout << "# FAILED TEST 8 stringToInt(hello) #\n";
255 }
256
257 try {
258     btassert<bool>(stringToInt("") == 0);
259     cout << "Passed TEST 9: stringToInt(\"\")\n";
260 } catch (bool b) {
261     cout << "# FAILED TEST 9 stringToInt(\"\") #\n";
262 }
263
264 try {
265     btassert<bool>(stringToDouble("") == 0);
266     cout << "Passed TEST 10: stringToDouble(\"\")\n";
267 } catch (bool b) {
268     cout << "# FAILED TEST 10 stringToDouble(\"\") #\n";
269 }
270
271 try {
272     btassert<bool>(stringToDouble("3.14") == 3.14);
273     cout << "Passed TEST 11: stringToDouble(3.14)\n";
274 } catch (bool b) {
275     cout << "# FAILED TEST 11 stringToDouble(3.14) #\n";
276 }
277
278 try {
279     btassert<bool>(stringToDouble("hello") == 0);
280     cout << "Passed TEST 12: stringToDouble(hello)\n";
281 } catch (bool b) {
282     cout << "# FAILED TEST 12 stringToDouble(hello) #\n";
283 }
284
285 try {
286     btassert<bool>(stringToBool("") == false);
287     cout << "Passed TEST 13: stringToBool(\"\")\n";
288 } catch (bool b) {
289     cout << "# FAILED TEST 13 stringToBool(\"\") #\n";
290 }
291
292 try {
293     btassert<bool>(stringToBool("TrUe") == true);
294     cout << "Passed TEST 14: stringToBool(TrUe)\n";
295 } catch (bool b) {
296     cout << "# FAILED TEST 14 stringToBool(TrUe) #\n";
297 }
298
299 try {
300     btassert<bool>(stringToBool("FALSE") == false);

```

```
301         cout << "Passed TEST 15: stringToBool(FALSE)\n";
302     } catch (bool b) {
303         cout << "# FAILED TEST 15 stringToBool(FALSE) #\n";
304     }
305
306     cout << "\nUNIT TEST COMPLETE\n\n";
307 }
308
309 template <typename X, typename A>
310 void btassert (A assertion)
311 {
312     if (!assertion)
313         throw X();
314 }
315
316 void clearScreen ()
317 {
318     #ifdef WIN32
319         system("cls");
320     #else
321         system("clear");
322     #endif
323 }
```