

```

1  /*
2   * Programming Challenge 3
3   */
4  #include <cassert>
5  #include <cmath>
6  #include <iostream>
7  #include <sstream>
8  #include <string>
9  using namespace std;
10
11 string goldilocks (string item, int number);
12 int rockScissorPaper (char playerOne, char playerTwo);
13 string charWithAsciiValueAsString (char c);
14 string toLower (string input);
15 string toUpper (string input);
16 char getCharacter (string input, int charIndex);
17
18 /* for unit testing -- do not alter */
19 template <typename X, typename A>
20 void btassert(A assertion);
21 void unittest ();
22
23 int main (int argc, char* argv[])
24 {
25     unittest();
26
27     return 0;
28 }
29
30 /*
31  * Tell the story of Goldilocks. For example,
32  * if item is "bed" and number is 1, the story
33  * will say "This bed is too soft". "item" parameter must be passed
34  * in all lowercase characters -- the function is case-sensitive.
35  * <ul>
36  * <li>item: "porridge", number: 1, return "This porridge is too hot"</li>
37  * <li>item: "porridge", number: 2, return "This porridge is too cold"</li>
38  * <li>item: "porridge", number: 3, return "This porridge is just right"</li>
39  * <li>item: "chair", number: 1, return "This chair is too big"</li>
40  * <li>item: "chair", number: 2, return "This chair is too big"</li>
41  * <li>item: "chair", number: 3, return "This chair is just right"</li>
42  * <li>item: "bed", number: 1, return "This bed is too hard"</li>
43  * <li>item: "bed", number: 2, return "This bed is too soft"</li>
44  * <li>item: "bed", number: 3, return "This bed is just right"</li>
45  * </ul>
46  * @param item a string representing the item in the story ("porridge", "chair", and
47  *           "bed" are the only legal values -- will default to "bed" on invalid argument)
48  * @param number which item (1, 2, and 3 are the only legal values -- will default to 3
49  *           on invalid argument)
50  * @return the output specified in the documentation above

```

```
51  */
52  string goldilocks (string item, int number)
53  {
54
55      if( item == "porridge" )    {
56
57          switch( number )    {
58
59              case 1:
60                  return "This porridge is too hot";
61
62              case 2:
63                  return "This porridge is too cold";
64
65              case 3:
66                  return "this porridge is just right";
67
68          }
69      }
70
71      if( item == "chair" )    {
72
73          switch( number )    {
74
75              case 1:
76                  return "This chair is too big";
77
78              case 2:
79                  return "This chair is too big";
80
81              case 3:
82                  return "This chair is just right";
83
84          }
85      }
86
87      if( item == "bed" ) {
88
89          switch( number )    {
90
91              case 1:
92                  return "This bed is too hard";
93
94              case 2:
95                  return "This bed is too soft";
96
97              case 3:
98                  return "This bed is just right";
99
100             return "This bed is just right";
```

```

101
102     }
103
104 }
105
106 }
107
108 /*
109  * Compute the outcome of a round of a rock-scissor-paper game. Lowercase or uppercase
110  * values for playerOne and playerTwo arguments are acceptable.
111  * Possible inputs: 'R' rock, 'S' scissor, 'P' paper
112  * <ul>
113  * <li>rocks beats scissors</li>
114  * <li>scissors beats paper</li>
115  * <li>paper beats rock</li>
116  * </ul>
117  * @param playerOne a char representing player one's "play" ('R', 'S', or 'P')
118  * @param playerTwo a char representing player two's "play" ('R', 'S', or 'P')
119  * @return 1 if player one wins, 2 if player two wins, 3 on a draw
120  */
121 int rockScissorPaper (char playerOne, char playerTwo)
122 {
123     switch( playerOne ) {
124
125         case 'r':
126         case 'R':
127             if( playerTwo == 's' || playerTwo == 'S' ) {
128
129                 return 1;
130
131             } else if( playerTwo == 'p' || playerTwo == 'P' ) {
132
133                 return 2;
134
135             } else if( playerTwo == 'r' || playerTwo == 'R' ) {
136
137                 return 3;
138
139             }
140
141         case 's':
142         case 'S':
143             if( playerTwo == 'p' || playerTwo == 'P' ) {
144
145                 return 1;
146
147             } else if( playerTwo == 'r' || playerTwo == 'R' ) {
148
149                 return 2;
150

```

```

151         } else if( playerTwo == 's' || playerTwo == 'S' ) {
152
153             return 3;
154
155         }
156
157     case 'p':
158     case 'P':
159         if( playerTwo == 'r' || playerTwo == 'R' ) {
160
161             return 1;
162
163             } else if( playerTwo == 's' || playerTwo == 'S' ) {
164
165                 return 2;
166
167             } else if( playerTwo == 'p' || playerTwo == 'P' ) {
168
169                 return 3;
170
171             }
172
173     }
174
175 }
176
177 /*
178  * Return a string that contains a character (taken from the parameter
179  * c) and its ASCII integer value. For example, If the char passed in is 'A',
180  * the function will return "A 65"
181  * @param c the char from which an ASCII value will be taken
182  * @return a string containing the original character argument, followed by a space,
183  *         followed by the ASCII integer value of the char
184  */
185 string charWithAsciiValueAsString (char c)
186 {
187
188     int intCharacterValue = static_cast<int>( c );
189     stringstream ss;
190
191     ss << c << ' ' << intCharacterValue;
192
193     return ss.str();
194
195 }
196
197 /*
198  * Return the input string with all characters converted to lowercase.
199  * @param input the string that will be converted to all lowercase characters.
200  * @return a string containing the input string, converted to all lowercase characters.

```

```

201     */
202 string toLower (string input)
203 {
204     for( int count = 0; count < input.length(); count++ )    {
205         input.at(count) = tolower( input.at(count) );
206     }
207     return input;
208 }
209
210
211
212
213 }
214
215 /*
216  * Return the input string with all characters converted to uppercase.
217  * @param input the string that will be converted to all uppercase characters.
218  * @return a string containing the input string, converted to all uppercase characters.
219  */
220 string toUpper (string input)
221 {
222     for( int count = 0; count < input.length(); count++ )    {
223         input.at(count) = toupper( input.at(count) );
224     }
225     return input;
226 }
227
228
229
230
231 }
232
233 /*
234  * Return the character from the input string at index charIndex.
235  * @param input the string from which the character will be taken
236  * @param charIndex an integer containing the zero-indexed position of the character
237  *         to return; charIndex must be >= 0 and < length of input string
238  * @return a char containing the character from the input string at charIndex position
239  */
240 char getCharacter (string input, int charIndex)
241 {
242     return input.at( charIndex );
243 }
244
245 /*
246  * Unit testing functions. Do not alter.
247  */
248 void unittest ()
249 {
250     cout << "\nSTARTING UNIT TEST\n\n";

```

```
251
252     try {
253         btassert<bool>(goldilocks("porridge", 2) == "This porridge is too cold");
254         cout << "Passed TEST 1: goldilocks (porridge, 2)\n";
255     } catch (bool b) {
256         cout << "# FAILED TEST 1 goldilocks (porridge, 2) #\n";
257     }
258
259     try {
260         btassert<bool>(goldilocks("chair", 3) == "This chair is just right");
261         cout << "Passed TEST 2: goldilocks (chair, 3)\n";
262     } catch (bool b) {
263         cout << "# FAILED TEST 2 goldilocks (chair, 3) #\n";
264     }
265
266     try {
267         btassert<bool>(goldilocks("bed", 1) == "This bed is too hard");
268         cout << "Passed TEST 3: goldilocks (bed, 1)\n";
269     } catch (bool b) {
270         cout << "# FAILED TEST 3 goldilocks (bed, 1) #\n";
271     }
272
273     try {
274         btassert<bool>(rockScissorPaper('r', 'S') == 1);
275         cout << "Passed TEST 4: rockScissorPaper (r, S)\n";
276     } catch (bool b) {
277         cout << "# FAILED TEST 4 rockScissorPaper (r, S) #\n";
278     }
279
280     try {
281         btassert<bool>(rockScissorPaper('R', 'p') == 2);
282         cout << "Passed TEST 5: rockScissorPaper (R, p)\n";
283     } catch (bool b) {
284         cout << "# FAILED TEST 5 rockScissorPaper (R, p) #\n";
285     }
286
287     try {
288         btassert<bool>(rockScissorPaper('S', 'P') == 1);
289         cout << "Passed TEST 6: rockScissorPaper (S, P)\n";
290     } catch (bool b) {
291         cout << "# FAILED TEST 6 rockScissorPaper (S, P) #\n";
292     }
293
294     try {
295         btassert<bool>(rockScissorPaper('r', 'r') == 3);
296         cout << "Passed TEST 7: rockScissorPaper (r, r)\n";
297     } catch (bool b) {
298         cout << "# FAILED TEST 7 rockScissorPaper (r, r) #\n";
299     }
300
```

```

301     try {
302         btassert<bool>(charWithAsciiValueAsString('Z') == "Z 90");
303         cout << "Passed TEST 8: charWithAsciiValueAsString (Z)\n";
304     } catch (bool b) {
305         cout << "# FAILED TEST 8 charWithAsciiValueAsString (Z) #\n";
306     }
307
308     try {
309         btassert<bool>(charWithAsciiValueAsString('a') == "a 97");
310         cout << "Passed TEST 9: charWithAsciiValueAsString (a)\n";
311     } catch (bool b) {
312         cout << "# FAILED TEST 9 charWithAsciiValueAsString (a) #\n";
313     }
314
315     try {
316         btassert<bool>(toLower("HELLO") == "hello");
317         cout << "Passed TEST 10: toLower (HELLO)\n";
318     } catch (bool b) {
319         cout << "# FAILED TEST 10 toLower (HELLO) #\n";
320     }
321
322     try {
323         btassert<bool>(toLower("g00dbYe") == "goodbye");
324         cout << "Passed TEST 11: toLower (g00dbYe)\n";
325     } catch (bool b) {
326         cout << "# FAILED TEST 11 toLower (g00dbYe) #\n";
327     }
328
329     try {
330         btassert<bool>(toUpper("hello") == "HELLO");
331         cout << "Passed TEST 12: toUpper (hello)\n";
332     } catch (bool b) {
333         cout << "# FAILED TEST 12 toUpper (hello) #\n";
334     }
335
336     try {
337         btassert<bool>(toUpper("g00dbYe") == "GOODBYE");
338         cout << "Passed TEST 13: toUpper (g00dbYe)\n";
339     } catch (bool b) {
340         cout << "# FAILED TEST 13 toUpper (g00dbYe) #\n";
341     }
342
343     try {
344         btassert<bool>(getCharacter("amazing", 3) == 'z');
345         cout << "Passed TEST 14: getCharacter (amazing, 3)\n";
346     } catch (bool b) {
347         cout << "# FAILED TEST 14 #\n";
348     }
349
350     try {

```

```
351         btassert<bool>(getCharacter("hooray!", 6) == '!');
352         cout << "Passed TEST 15: getCharacter (hooray, 6)\n";
353     } catch (bool b) {
354         cout << "# FAILED TEST 15 getCharacter (hooray, 6) #\n";
355     }
356
357     cout << "\nUNIT TEST COMPLETE\n\n";
358 }
359
360 template <typename X, typename A>
361 void btassert (A assertion)
362 {
363     if (!assertion)
364         throw X();
365 }
366
```