

```

1  /*
2   * Programming Challenge 10
3   */
4  #include <cassert>
5  #include <iomanip>
6  #include <iostream>
7  using namespace std;
8
9  /*
10   * Class Converter.
11   * A class that performs a variety of conversions.
12   */
13  class Converter
14  {
15      public:
16
17          /*
18           * Convert a temperature from metric to imperial units or from
19           * imperial to metric units.
20           * @param temp the temperature to be converted -- in degrees Fahrenheit
21           *               or degrees Celsius
22           * @param targetUnits a char representing the system to convert the
23           *                   temperature to -- 'M' to convert to metric, 'I' to convert
24           *                   to imperial
25           * @return a float containing the converted temperature
26           */
27          float convertTemperature (float temp, char targetUnits);
28
29          /*
30           * Convert a distance from metric to imperial units or from
31           * imperial to metric units.
32           * @param distance the distance to be converted -- in feet or meters
33           * @param targetUnits a char representing the system to convert the
34           *                   temperature to -- 'M' to convert to metric, 'I' to convert
35           *                   to imperial
36           * @return a float containing the converted distance
37           */
38          float convertDistance (float distance, char targetUnits);
39
40      private:
41
42          /* Called by convertTemperature if target units is 'M' */
43          float fahrenheitToCelsius (float temp);
44
45          /* Called by convertTemperature if target units is 'I' */
46          float celsiusToFahrenheit (float temp);
47
48          /* Called by convertDistance if target units is 'M' */
49          float feetToMeters (float distance);
50

```

```

51     /* Called by convertDistance if target units is 'M' */
52     float metersToFeet (float distance);
53 };
54
55 /* for unit testing -- do not alter */
56 template <typename X, typename A>
57 void btassert(A assertion);
58 void unittest ();
59
60 int main (int argc, char* argv[])
61 {
62     unittest();
63
64     return 0;
65 }
66
67 // CODE HERE -- FUNCTION DEFINITIONS
68
69 float Converter::convertTemperature (float temp, char targetUnits)
70 {
71     if( targetUnits == 'M' )
72         return fahrenheitToCelsius( temp );
73     else if( targetUnits == 'I' )
74         return celsiusToFahrenheit( temp );
75 }
76
77 float Converter::convertDistance (float distance, char targetUnits)
78 {
79     if( targetUnits == 'M' )
80         return feetToMeters( distance );
81     else if( targetUnits == 'I' )
82         return metersToFeet( distance );
83 }
84
85 float Converter::fahrenheitToCelsius (float temp)
86 {
87     return (5.0/9.0 * (temp - 32));
88 }
89
90 float Converter::celsiusToFahrenheit (float temp)
91 {
92     return ((9.0/5.0 * temp) + 32);
93 }
94
95 float Converter::feetToMeters (float distance)
96 {
97     return (0.3048 * distance );
98 }
99
100 float Converter::metersToFeet (float distance)

```

```

101 {
102     return ((1/0.3048) * distance);
103 }
104
105 /*
106  * Unit testing functions. Do not alter.
107  */
108
109 void unittest ()
110 {
111     cout << "\nSTARTING UNIT TEST\n\n";
112
113     Converter c;
114
115     try {
116         btassert<bool>(static_cast<int>(c.convertTemperature(32, 'M')) == 0);
117         cout << "Passed TEST 1: convertTemperature(32, 'M')\n";
118     } catch (bool b) {
119         cout << "# FAILED TEST 1 convertTemperature(32, 'M') #\n";
120     }
121
122     try {
123         btassert<bool>(static_cast<int>(c.convertTemperature(212, 'M')) == 100);
124         cout << "Passed TEST 2: convertTemperature(212, 'M')\n";
125     } catch (bool b) {
126         cout << "# FAILED TEST 2 convertTemperature(212, 'M') #\n";
127     }
128
129     try {
130         btassert<bool>(static_cast<int>(c.convertTemperature(98.6, 'M')) == 37);
131         cout << "Passed TEST 3: convertTemperature(98.6, 'M')\n";
132     } catch (bool b) {
133         cout << "# FAILED TEST 3 convertTemperature(98.6, 'M') #\n";
134     }
135
136     try {
137         btassert<bool>(static_cast<int>(c.convertTemperature(0, 'I')) == 32);
138         cout << "Passed TEST 4: convertTemperature(0, 'I')\n";
139     } catch (bool b) {
140         cout << "# FAILED TEST 4 convertTemperature(0, 'I') #\n";
141     }
142
143     try {
144         btassert<bool>(static_cast<int>(c.convertTemperature(100, 'I')) == 212);
145         cout << "Passed TEST 5: convertTemperature(100, 'I')\n";
146     } catch (bool b) {
147         cout << "# FAILED TEST 5 convertTemperature(100, 'I') #\n";
148     }
149
150     try {

```

```

151     btassert<bool>(c.convertDistance(300, 'M') > 91.43f && c.convertDistance(300, 'M') < 91.45f);
152     cout << "Passed TEST 6: convertDistance(300, 'M')\n";
153 } catch (bool b) {
154     cout << "# FAILED TEST 6 convertDistance(300, 'M') #\n";
155 }
156
157 try {
158     btassert<bool>(c.convertDistance(5280, 'M') > 1609.3f && c.convertDistance(5280, 'M') < 1609.4f);
159     cout << "Passed TEST 7: convertDistance(5280, 'M')\n";
160 } catch (bool b) {
161     cout << "# FAILED TEST 7 convertDistance(5280, 'M') #\n";
162 }
163
164 try {
165     btassert<bool>(c.convertDistance(1, 'M') > .30f && c.convertDistance(1, 'M') < .31f);
166     cout << "Passed TEST 8: convertDistance(1, 'M')\n";
167 } catch (bool b) {
168     cout << "# FAILED TEST 8 convertDistance(1, 'M') #\n";
169 }
170
171 try {
172     btassert<bool>(c.convertDistance(1, 'I') > 3.28f && c.convertDistance(1, 'M') < 3.29f);
173     cout << "Passed TEST 9: convertDistance(1, 'I')\n";
174 } catch (bool b) {
175     cout << "# FAILED TEST 9 convertDistance(1, 'I') #\n";
176 }
177
178 try {
179     btassert<bool>(c.convertDistance(.5f, 'I') > 1.6f && c.convertDistance(.5f, 'M') < 1.7f);
180     cout << "Passed TEST 10: convertDistance(.5f, 'I')\n";
181 } catch (bool b) {
182     cout << "# FAILED TEST 10 convertDistance(.5f, 'I') #\n";
183 }
184
185 cout << "\nUNIT TEST COMPLETE\n\n";
186 }
187
188 template <typename X, typename A>
189 void btassert (A assertion)
190 {
191     if (!assertion)
192         throw X();
193 }

```