

```

1  /*
2   * Programming Challenge 2
3   */
4  #include <cassert>
5  #include <cmath>
6  #include <iostream>
7  using namespace std;
8
9  bool isFactor (int numerator, int denominator);
10 void makeChange (int initialValue, int& quarters, int& dimes, int& nickels, int& pennies);
11 double launchHumanCannonball (double initialVelocity, double launchAngle);
12
13 /* for unit testing -- do not alter */
14 template <typename X, typename A>
15 void btassert(A assertion);
16 void unittest ();
17
18 int main (int argc, char* argv[])
19 {
20     unittest();
21
22     return 0;
23 }
24
25 /*
26  * Determine if one integer is a factor (evenly divisible into)
27  * another integer.
28  * @param numerator an int representing the numerator (the value being tested against)
29  *      in the equation
30  * @param denominator an int representing the denominator (the potential factor)
31  *      in the equation
32  * @return true if the numerator is evenly divisible by the denominator (making the denominator
33  *      a factor, else return false
34  */
35 bool isFactor (int numerator, int denominator)
36 {
37     if( !(numerator % denominator) )    {
38
39         return true;
40
41     } else {
42
43         return false;
44
45     }
46 }
47
48 /*
49  * Given an initial integer value (representing change to be given, such as in a financial
50  * transaction), break the value down into the number of quarters, dimes, nickels, and pennies

```

```

51  * that would be given as change.
52  * @param initialValue an int representing the amount of change to be broken down
53  * @param quarters the number of quarters that come out of the initial value
54  * @param dimes the number of dimes that come out of the initial value, after quarters have
55  *     been taken out
56  * @param nickels the number of nickels that come out of the initial value, after quarters
57  *     and dimes have been taken out
58  * @param pennies the number of pennies that come out of the initial value, after quarters,
59  *     dimes, and nickels have been taken out
60  */
61  void makeChange (int initialValue, int& quarters, int& dimes, int& nickels, int& pennies)
62  {
63      int remainder;
64
65      quarters    = initialValue / 25;
66      remainder   = initialValue % 25;
67
68      dimes       = remainder / 10;
69      remainder   = remainder % 10;
70
71      nickels     = remainder / 5;
72      remainder   = remainder % 5;
73
74      pennies     = remainder;
75  }
76
77  /*
78  * Computes the horizontal distance traveled by a human cannonball given an initial
79  * velocity and launch angle. Simplified -- does not account for many factors that
80  * affect projectile motion.
81  * @param initialVelocity a double representing the initial velocity of the human
82  *     cannonball in meters/second
83  * @param launchAngle a double representing the launch angle of the human cannonball
84  *     in degrees
85  * @return a double representing the horizontal distance the human cannonball will
86  *     travel
87  */
88  double launchHumanCannonball (double initialVelocity, double launchAngle)
89  {
90      double radangle    = launchAngle * (M_PI/180);
91      double xveloc      = initialVelocity * cos( radangle );
92      double yveloc      = initialVelocity * sin( radangle ) * -1;
93      double flighttime  = yveloc * 2 / -9.8;
94
95      return xveloc * flighttime;
96  }
97
98  /*
99  * Unit testing functions. Do not alter.
100 */

```

```

101 void unittest ()
102 {
103     cout << "\nSTARTING UNIT TEST\n\n";
104
105     try {
106         btassert<bool>(isFactor(100, 25) == true);
107         cout << "Passed TEST 1: isFactor 100/25\n";
108     } catch (bool b) {
109         cout << "# FAILED TEST 1 isFactor 100/25 #\n";
110     }
111
112     try {
113         btassert<bool>(isFactor(100, 26) == false);
114         cout << "Passed TEST 2: isFactor 100/26\n";
115     } catch (bool b) {
116         cout << "# FAILED TEST 2 isFactor 100/26 #\n";
117     }
118
119     try {
120         btassert<bool>(isFactor(100, 26) == false);
121         cout << "Passed TEST 3: isFactor 100/100\n";
122     } catch (bool b) {
123         cout << "# FAILED TEST 3 isFactor 100/100 #\n";
124     }
125
126     int q = 0, n = 0, d = 0, p = 0;
127
128     try {
129         makeChange(0, q, d, n, p);
130         btassert<bool> ((q == 0) && (d == 0) && (n == 0) && (p == 0));
131         cout << "Passed TEST 4: makeChange $0.00\n";
132     } catch (bool b) {
133         cout << "# FAILED TEST 4 makeChange $0.00 #\n";
134     }
135
136     try {
137         makeChange(41, q, d, n, p);
138         btassert<bool> ((q == 1) && (d == 1) && (n == 1) && (p == 1));
139         cout << "Passed TEST 5: makeChange $0.41\n";
140     } catch (bool b) {
141         cout << "# FAILED TEST 5 makeChange $0.41 #\n";
142     }
143
144     try {
145         makeChange(99, q, d, n, p);
146         btassert<bool> ((q == 3) && (d == 2) && (n == 0) && (p == 4));
147         cout << "Passed TEST 6: makeChange $0.99\n";
148     } catch (bool b) {
149         cout << "# FAILED TEST 6 makeChange $0.99 #\n";
150     }

```

```

151
152     double value = launchHumanCannonball(25.0, 45.0);
153     double scale = 0.01; // round to nearest one-hundreth
154
155     try {
156         value = (int)(value / scale) * scale;
157         btassert<bool>((value <= 63.9) && (value >= 63.5));
158         cout << "Passed TEST 7: launchHumanCannonball(25, 45)\n";
159     } catch (bool b) {
160         cout << "# FAILED TEST 7 launchHumanCannonball(25, 45) #\n";
161     }
162
163     try {
164         value = launchHumanCannonball(40.0, 60.0);
165         value = (int)(value / scale) * scale;
166         btassert<bool>((value <= 141.5) && (value >= 141.1));
167         cout << "Passed TEST 8: launchHumanCannonball(40, 60)\n";
168     } catch (bool b) {
169         cout << "# FAILED TEST 8 launchHumanCannonball(40, 60) #\n";
170     }
171
172     try {
173         value = launchHumanCannonball(10.0, 30.0);
174         value = (int)(value / scale) * scale;
175         btassert<bool>((value <= 9.0) && (value >= 8.6));
176         cout << "Passed TEST 9: launchHumanCannonball(10, 30)\n";
177     } catch (bool b) {
178         cout << "# FAILED TEST 9 launchHumanCannonball(10, 30) #\n";
179     }
180
181     try {
182         value = launchHumanCannonball(10.0, 90.0);
183         value = (int)(value / scale) * scale;
184         btassert<bool>((value <= 0.2) && (value >= -0.2));
185         cout << "Passed TEST 10: launchHumanCannonball(10, 90)\n";
186     } catch (bool b) {
187         cout << "# FAILED TEST 10 launchHumanCannonball(10, 90) #\n";
188     }
189
190     cout << "\nUNIT TEST COMPLETE\n\n";
191 }
192
193 template <typename X, typename A>
194 void btassert (A assertion)
195 {
196     if (!assertion)
197         throw X();
198 }
199

```