Listing 1: Coordinate.hpp

```
1  #pragma once
2
3  #include <cmath>
4  #include <doctest.h>
5
6  /*--------------------------------------------------------------------------
7   * class definition
8   *-------------------------------------------------------------------------*/
9
10 /**
11  * @brief CMP 246 Module 1 class representing an (x, y) coordinate.
12  *
13  * This class represents an (x, y) coordinate. The class is templated, and
14  * can support float, double, or long double for the type of each element of
15  * the coordinate.
16  */
17 template <class T> class Coordinate {
18 public:
19     /**
20      * @brief Default constructor.
21      *
22      * Create a new coordinate with each element set to zero.
23      */
24     Coordinate() : x(), y() { }
25
26     /**
27      * @brief Initializing constructor.
28      *
29      * Create a new coordinate with the specified values.
30      *
31      * @param inX x-value for this coordinate.
32      * @param inY y-value for this coordinate.
33      */
34     Coordinate(T inX, T inY) : x(inX), y(inY) { }
35
36     /**
37      * @brief Mutator for the x-value.
38      *
39      * @param inX New x-value for this coordinate.
40      */
41     void setX(T inX) { x = inX; }
42
43     /**
44      * @brief Mutator for the y-value.
45      *
46      * @param inY New y-value for this coordinate.
47      */
48     void setY(T inY) { y = inY; }
49
50     /**
51      * @brief Accessor for the x-value.
52      *
53      * @return This coordinate's x-value.
54      */
55     T getX() const { return x; }
56
57     /**
58      * @brief Accessor for the y-value.
59      *
```

```
60       * @return This coordinate's y-value.
61       */
62      T getY() const { return y; }
63
64      /**
65       * @brief Euclidean distance method.
66       *
67       * Calculate the Euclidean distance between this coordinate and another.
68       *
69       * @param other Reference to the other coordinate to use.
70       * @return Euclidean distance between this coordinate and the other.
71       */
72      T distanceTo(const Coordinate<T> &other) const;
73  private:
74      /**
75       * X-value of this coordinate.
76       */
77      T x;
78
79      /**
80       * Y-value of this coordinate.
81       */
82      T y;
83  };
84
85  /*--------------------------------------------------------------------------
86   * method implementations
87   *-------------------------------------------------------------------------*/
88
89  /*
90   * Distance method.
91   */
92  template <class T>
93  T Coordinate<T>::distanceTo(const Coordinate<T> &other) const {
94      T dx = x - other.x;
95      dx = dx * dx;
96      T dy = y - other.y;
97      dy = dy * dy;
98      return sqrt(dx + dy);
99  }
100
101 // Doctest unit tests for distanceTo
102 TEST_CASE("testing Coordinate<T>::distanceTo") {
103     Coordinate<float> f1, f2;
104
105     // distance should be 0
106     CHECK(f1.distanceTo(f2) == 0.0f);
107
108     f1.setX(11.0f); f1.setY(11.0f);
109     f2.setX(11.0f); f2.setY(11.0f);
110     // distance should still be 0
111     CHECK(f1.distanceTo(f2) == 0.0f);
112
113     Coordinate<double> f3(0.7071068, 0.0d);
114     Coordinate<double> f4(0.0d, 0.7071068);
115     // distance should be 1 -- within floating point tolerance errors
116     CHECK(f3.distanceTo(f4) == doctest::Approx(1.0));
117
118     Coordinate<long double> f5(2.0L, 3.0L);
119     Coordinate<long double> f6(-3.0L, -2.0L);
```

```
120     // distance should be approx 7.071
121     CHECK(f5.distanceTo(f6) == doctest::Approx(7.071));
122 }
```

Listing 2: MontePi.cpp

```
1  #include <cstdlib>
2  #include <ctime>
3  #include <iostream>
4  #include <random>
5  #include "Coordinate.hpp"
6
7  /**
8   * @brief CMP 246 Module 1 main program to exercise the Coordinate class.
9   *
10  * This program uses the Monte Carlo technique to create an estimate of the
11  * number pi, using float and long double coordinates.
12  */
13 int main() {
14     // Mersenne Twister high-quality pseudo-random number generator
15     std::mt19937_64 prng(time(0));
16
17     // distributions to produce numbers in [0, 1] for floats and long doubles
18     std::uniform_real_distribution<float> distribF(0.0f, 1.0f);
19     std::uniform_real_distribution<long double> distribLD(0.0L, 1.0L);
20
21     // prompt for number of coordinates
22     unsigned n;
23     std::cout << "Enter␣number␣of␣darts␣to␣throw:␣";
24     std::cin >> n;
25
26     // perform the estimate using float coordinates
27     Coordinate<float> zeroF, dartF;
28     float numInF = 0.0f;
29     for(unsigned i = 0u; i < n; i++) {
30         dartF.setX(distribF(prng));
31         dartF.setY(distribF(prng));
32         if(zeroF.distanceTo(dartF) < 1.0f) {
33             numInF++;
34         }
35     }
36     float estPiF = numInF / n * 4.0f;
37
38     std::cout << "Float␣estimate␣of␣pi:␣" << estPiF << std::endl;
39
40     // perform the estimate using long double coordinates
41     Coordinate<long double> zeroLD, dartLD;
42     long double numInLD = 0.0L;
43     for(unsigned i = 0u; i < n; i++) {
44         dartLD.setX(distribLD(prng));
45         dartLD.setY(distribLD(prng));
46         if(zeroLD.distanceTo(dartLD) < 1.0L) {
47             numInLD++;
48         }
49     }
50     long double estPiLD = numInLD / n * 4.0L;
51
52     std::cout << "Long␣double␣estimate␣of␣pi:␣" << estPiLD << std::endl;
53
54     return EXIT_SUCCESS;
```

```
55  }
```