```
/*
 * Programming Challenge 9
 */
#include <cassert>
#include <iomanip>
#include <iostream>
#include <sstream>
#include <string>
using namespace std;


/*
 * Class Spaceship.
 * A class to model a simple spaceship for a science fiction
 * game or story.
 */
class Spaceship
{
    public:

        /*
         * Set the name of this Spaceship.
         * @param newName a string containing a name for this Spaceship
         */
        void setName (string newName);

        /*
         * Set the top speed of this Spaceship.
         * @param newTopSpeed a float containing a top speed for this Spaceship
         *        in warp
         */
        void setTopSpeed (float newTopSpeed);

        /*
         * Set the fuel source of this Spaceship.
         * @param newFuelSource a string containing a fuel source for this Spaceship
         */
        void setFuelSource (string newFuelSource);

        /*
         * Set the crew capacity of this Spaceship.
         * @param newCrewCapacity an int containing a crew capacity for this Spaceship
         */
        void setCrewCapacity (int newCrewCapacity);

        /*
         * Get the name of this Spaceship.
         * @return a string containing the name of this Spaceship
         */
        string getName () const;
```

```
51
52          /*
53           * Get the top speed of this Spaceship.
54           * @return a float containing the top speed of this Spaceship
55           */
56          float getTopSpeed () const;
57
58          /*
59           * Get the fuel source of this Spaceship.
60           * @return a string containing the fuel source of this Spaceship
61           */
62          string getFuelSource () const;
63
64          /*
65           * Get the crew capacity of this Spaceship.
66           * @return an int containing the crew capacity of this Spaceship
67           */
68          int getCrewCapacity () const;
69
70          /*
71           * Get a string representation of this Spaceship's specifications.
72           * The string will be formatted as
73           * "NAME, top speed: warp TOP_SPEED, fuel source: FUEL_SOURCE, crew capacity: CREW_CAPACITY"
74           * where NAME, TOP_SPEED (to two decimal places), FUEL_SOURCE, and CREW_CAPACITY
75           * contain the values of the associated member variables.
76           * @return a string representation of this Spaceship's specifications
77           */
78          string toString () const;
79
80      private:
81
82          string name;
83          float topSpeed;
84          string fuelSource;
85          int crewCapacity;
86  };
87
88  /* for unit testing -- do not alter */
89  template <typename X, typename A>
90  void btassert(A assertion);
91  void unittest ();
92
93  int main (int argc, char* argv[])
94  {
95      unittest();
96
97      return 0;
98  }
99
100 // CODE HERE -- FUNCTION DEFINITIONS
```

```
101
102 void Spaceship::setName (string newName)
103 {
104     name = newName;
105 }
106
107 void Spaceship::setTopSpeed (float newTopSpeed)
108 {
109     topSpeed = newTopSpeed;
110 }
111
112 void Spaceship::setFuelSource (string newFuelSource)
113 {
114     fuelSource = newFuelSource;
115 }
116
117 void Spaceship::setCrewCapacity (int newCrewCapacity)
118 {
119     crewCapacity = newCrewCapacity;
120 }
121
122 string Spaceship::getName () const
123 {
124     return name;
125 }
126
127 float Spaceship::getTopSpeed () const
128 {
129     return topSpeed;
130 }
131
132 string Spaceship::getFuelSource () const
133 {
134     return fuelSource;
135 }
136
137 int Spaceship::getCrewCapacity () const
138 {
139     return crewCapacity;
140 }
141
142 string Spaceship::toString () const
143 {
144     stringstream ssSpecifications;
145
146     ssSpecifications.setf( ios::fixed );
147     ssSpecifications.setf( ios::showpoint );
148     ssSpecifications.precision( 2 );
149
150     ssSpecifications << name << ", top speed: warp " << topSpeed << ", fuel source: " << fuelSource << ", crew capacity: " <<
```

```
    crewCapacity;

    return ssSpecifications.str();
}

/*
 * Unit testing functions. Do not alter.
 */

void unittest ()
{
    cout << "\nSTARTING UNIT TEST\n\n";

    Spaceship enterprise;

    enterprise.setName("USS Enterprise");
    try {
        btassert<bool>(enterprise.getName() == "USS Enterprise");
        cout << "Passed TEST 1: setName/getName\n";
    } catch (bool b) {
        cout << "# FAILED TEST 1 setName/getName #\n";
    }

    enterprise.setTopSpeed(9.6);
    try {
        btassert<bool>(enterprise.getTopSpeed() >= 9.59 && enterprise.getTopSpeed() <= 9.61);
        cout << "Passed TEST 2: setTopSpeed/getTopSpeed\n";
    } catch (bool b) {
        cout << "# FAILED TEST 2 setTopSpeed/getTopSpeed #\n";
    }

    enterprise.setFuelSource("plasma");
    try {
        btassert<bool>(enterprise.getFuelSource() == "plasma");
        cout << "Passed TEST 3: setFuelSource/getFuelSource\n";
    } catch (bool b) {
        cout << "# FAILED TEST 3 setFuelSource/getFuelSource #\n";
    }

    enterprise.setCrewCapacity(5000);
    try {
        btassert<bool>(enterprise.getCrewCapacity() == 5000);
        cout << "Passed TEST 4: setCrewCapacity/getCrewCapacity\n";
    } catch (bool b) {
        cout << "# FAILED TEST 4 setCrewCapacity/getCrewCapacity #\n";
    }

    try {
        btassert<bool>(enterprise.toString() ==
                            "USS Enterprise, top speed: warp 9.60, fuel source: plasma, crew capacity: 5000");
```

```
        cout << "Passed TEST 5: toString\n";
    } catch (bool b) {
        cout << "# FAILED TEST 5 toString #\n";
    }

    cout << "\nUNIT TEST COMPLETE\n\n";
}

template <typename X, typename A>
void btassert (A assertion)
{
    if (!assertion)
        throw X();
}
```