```cpp
/*
 * Programming Challenge 12
 */
#include <cassert>
#include <iomanip>
#include <iostream>
using namespace std;

/*
 * Class Prize. A class representing a simple prize, such as for a gameshow.
 */
class Prize
{
    public:

        // CODE HERE -- DECLARE FUNCTIONS

        /*
         * Constructor.
         * @param newName string containing a name for this Prize; default argument is "no name";
         *        if newValue is > 100, converts name to all capital letters and concatenates a '!'
         *        at the end of name
         * @param newValue unsigned int containing a value for this Prize; default argument is 0
         */
        Prize( string newName = "no name!", unsigned int newValue = 0 );

        /*
         * Get this Prize's name
         * @return a string containing this Prize's name
         */
        string getName();

        /*
         * Get this Prize's value
         * @return an unsigned int containing this Prize's value
         */
        unsigned int getValue();

    private:

        string name;
        unsigned int value;
};

/*
 * Class SecretDoor. A class representing a "secret door" such as might be used on a game show.
 * Contains a Prize.
 */
class SecretDoor
{
```

```
51      public:
52
53          // CODE HERE -- DECLARE FUNCTIONS
54
55          /*
56           * Constructor.
57           * @param newNumber unsigned int containing a value for this SecretDoor's number; default argument is 1
58           * @param newPrize Prize containing a Prize that is "hidden" behind this secret door; default argument
59           *        is Prize()
60           */
61          SecretDoor( unsigned int newNumber = 1, Prize newPrize = Prize() );
62
63          /*
64           * Get this SecretDoor's number.
65           * @return an unsigned int containing this SecretDoor's number
66           */
67          unsigned int getNumber();
68
69          /*
70           * Get this SecretDoor's Prize.
71           * @return the Prize, by reference, "hidden behind" this SecretDoor
72           */
73          Prize getPrize();
74
75      private:
76
77          unsigned int number;
78          Prize prize;
79  };
80
81  /* for unit testing -- do not alter */
82  template <typename X, typename A>
83  void btassert(A assertion);
84  void unittest ();
85
86  int main (int argc, char* argv[])
87  {
88      unittest();
89
90      return 0;
91  }
92
93  // CODE HERE -- FUNCTION DEFINITIONS FOR PRIZE; USE INITIALIZER SECTION FOR CONSTRUCTOR
94
95  Prize::Prize( string newName, unsigned int newValue )
96  {
97      if( newValue > 100 )
98      {
99          for( int count = 0; count < newName.length(); count++ )
100         {
```

```
            newName[count] = toupper( newName[count] );
        }

        name = newName + "!";
    }
    else
    {
        name = newName;
    }

    value = newValue;
}

string Prize::getName()
{
    return name;
}

unsigned int Prize::getValue()
{
    return value;
}


// CODE HERE -- FUNCTION DEFINITIONS FOR SECRETDOOR; USE INITIALIZER SECTION FOR CONSTRUCTOR

SecretDoor::SecretDoor( unsigned int newNumber, Prize newPrize )
{
    number = newNumber;
    prize = newPrize;
}

unsigned int SecretDoor::getNumber()
{
    return number;
}

/*
 * Get this SecretDoor's Prize.
 * @return the Prize, by reference, "hidden behind" this SecretDoor
 */
Prize SecretDoor::getPrize()
{
    return prize;
}

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

```

```
151 void unittest ()
152 {
153     cout << "\nSTARTING UNIT TEST\n\n";
154     Prize * prizePointer;
155     SecretDoor * doorPointer;
156     try {
157         prizePointer = new Prize();
158         btassert<bool>(prizePointer->getName() == "no name!" && prizePointer->getValue() == 0);
159         cout << "Passed TEST 1: CREATING A PRIZE () \n";
160     } catch (bool b) {
161         cout << "# FAILED TEST 1: CREATING A PRIZE () #\n";
162     }
163     delete prizePointer;
164     try {
165         prizePointer = new Prize("A Brand New Car");
166         btassert<bool>(prizePointer->getName() == "A Brand New Car" && prizePointer->getValue() == 0);
167         cout << "Passed TEST 2: CREATING A PRIZE (NAME) \n";
168     } catch (bool b) {
169         cout << "# FAILED TEST 2: CREATING A PRIZE (NAME) #\n";
170     }
171     delete prizePointer;
172     try {
173         prizePointer = new Prize("A Couch",1000);
174         btassert<bool>(prizePointer->getName() == "A COUCH!" && prizePointer->getValue() == 1000);
175         cout << "Passed TEST 3: CREATING A PRIZE (NAME,VALUE) \n";
176     } catch (bool b) {
177         cout << "# FAILED TEST 3: CREATING A PRIZE (NAME,VALUE) #\n";
178     }
179
180     try {
181         doorPointer = new SecretDoor();
182         btassert<bool>(doorPointer->getPrize().getName() == "no name!" && doorPointer->getNumber() == 1);
183         cout << "Passed TEST 4: CREATING A SECRET DOOR () \n";
184     } catch (bool b) {
185         cout << "# FAILED TEST 4: CREATING A SECRET DOOR () #\n";
186     }
187     delete doorPointer;
188     try {
189         doorPointer = new SecretDoor(5);
190         btassert<bool>(doorPointer->getPrize().getName() == "no name!" && doorPointer->getNumber() == 5);
191         cout << "Passed TEST 5: CREATING A SECRET DOOR (NUMBER) \n";
192     } catch (bool b) {
193         cout << "# FAILED TEST 5: CREATING A SECRET DOOR (NUMBER) #\n";
194     }
195     delete doorPointer;
196     try {
197         doorPointer = new SecretDoor(5,*prizePointer);
198         btassert<bool>(doorPointer->getPrize().getName() == "A COUCH!" && doorPointer->getNumber() == 5);
199         cout << "Passed TEST 6: CREATING A SECRET DOOR (NUMBER,PRIZE) \n";
200     } catch (bool b) {
```

```
        cout << "# FAILED TEST 6: CREATING A SECRET DOOR (NUMBER,PRIZE) #\n";
    }
    delete doorPointer;
    delete prizePointer;

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

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