```cpp
/*
 * Programming Challenge 3
 */
#include <cassert>
#include <cmath>
#include <iostream>
#include <sstream>
#include <string>
using namespace std;

string goldilocks (string item, int number);
int rockScissorPaper (char playerOne, char playerTwo);
string charWithAsciiValueAsString (char c);
string toLower (string input);
string toUpper (string input);
char getCharacter (string input, int charIndex);

/* for unit testing -- do not alter */
template <typename X, typename A>
void btassert(A assertion);
void unittest ();

int main (int argc, char* argv[])
{
    unittest();

    return 0;
}

/*
 * Tell the story of Goldilocks. For example,
 * if item is "bed" and number is 1, the story
 * will say "This bed is too soft". "item" parameter must be passed
 * in all lowercase characters -- the function is case-sensitive.
 * <ul>
 * <li>item: "porridge", number: 1, return "This porridge is too hot"</li>
 * <li>item: "porridge", number: 2, return "This porridge is too cold"</li>
 * <li>item: "porridge", number: 3, return "This porridge is just right"</li>
 * <li>item: "chair", number: 1, return "This chair is too big"</li>
 * <li>item: "chair", number: 2, return "This chair is too big"</li>
 * <li>item: "chair", number: 3, return "This chair is just right"</li>
 * <li>item: "bed", number: 1, return "This bed is too hard"</li>
 * <li>item: "bed", number: 2, return "This bed is too soft"</li>
 * <li>item: "bed", number: 3, return "This bed is just right"</li>
 * </ul>
 * @param item a string representing the item in the story ("porridge", "chair", and
 *        "bed" are the only legal values -- will default to "bed" on invalid argument)
 * @param number which item (1, 2, and 3 are the only legal values -- will default to 3
 *        on invalid argument)
 * @return the output specified in the documentation above
```

```
 */
string goldilocks (string item, int number)
{

    if( item == "porridge" )    {

        switch( number )    {

            case 1:
                return "This porridge is too hot";

            case 2:
                return "This porridge is too cold";

            case 3:
                return "this porridge is just right";

        }

    }

    if( item == "chair" )   {

        switch( number )    {

            case 1:
                return "This chair is too big";

            case 2:
                return "This chair is too big";

            case 3:
                return "This chair is just right";

        }

    }

    if( item == "bed" ) {

        switch( number )    {

            case 1:
                return "This bed is too hard";

            case 2:
                return "This bed is too soft";

            case 3:
                return "This bed is just right";
```

```
        }

    }

}

/*
 * Compute the outcome of a round of a rock-scissor-paper game. Lowercase or uppercase
 * values for playerOne and playerTwo arguments are acceptable.
 * Possible inputs: 'R' rock, 'S' scissor, 'P' paper
 * <ul>
 * <li>rocks beats scissors</li>
 * <li>scissors beats paper</li>
 * <li>paper beats rock</li>
 * </ul>
 * @param playerOne a char representing player one's "play" ('R', 'S', or 'P')
 * @param playerTwo a char representing player two's "play" ('R', 'S', or 'P')
 * @return 1 if player one wins, 2 if player two wins, 3 on a draw
 */
int rockScissorPaper (char playerOne, char playerTwo)
{
    switch( playerOne ) {

        case 'r':
        case 'R':
            if( playerTwo == 's' || playerTwo == 'S' )  {

                return 1;

            } else if( playerTwo == 'p' || playerTwo == 'P' )   {

                return 2;

            } else if( playerTwo == 'r' || playerTwo == 'R' )   {

                return 3;

            }

        case 's':
        case 'S':
            if( playerTwo == 'p' || playerTwo == 'P' )  {

                return 1;

            } else if( playerTwo == 'r' || playerTwo == 'R' )   {

                return 2;

```

```
            } else if( playerTwo == 's' || playerTwo == 'S' )   {

                return 3;

            }

        case 'p':
        case 'P':
            if( playerTwo == 'r' || playerTwo == 'R' )  {

                return 1;

            } else if( playerTwo == 's' || playerTwo == 'S' )   {

                return 2;

            } else if( playerTwo == 'p' || playerTwo == 'P' )   {

                return 3;

            }

    }

}

/*
 * Return a string that contains a character (taken from the parameter
 * c) and its ASCII integer value. For example, If the char passed in is 'A',
 * the function will return '"A 65"
 * @param c the char from which an ASCII value will be taken
 * @return a string containing the original character argument, followed by a space,
 *         followed by the ASCII integer value of the char
 */
string charWithAsciiValueAsString (char c)
{

    int intCharacterValue   = static_cast<int>( c );
    stringstream ss;

    ss << c << ' ' << intCharacterValue;

    return ss.str();

}

/*
 * Return the input string with all characters converted to lowercase.
 * @param input the string that will be converted to all lowercase characters.
 * @return a string containing the input string, converted to all lowercase characters.
```

```
 */
string toLower (string input)
{

    for( int count = 0; count < input.length(); count++ )   {

        input.at(count) = tolower( input.at(count) );

    }

    return input;

}

/*
 * Return the input string with all characters converted to uppercase.
 * @param input the string that will be converted to all uppercase characters.
 * @return a string containing the input string, converted to all uppercase characters.
 */
string toUpper (string input)
{

    for( int count = 0; count < input.length(); count++ )   {

        input.at(count) = toupper( input.at(count) );

    }

    return input;

}

/*
 * Return the character from the input string at index charIndex.
 * @param input the string from which the character will be taken
 * @param charIndex an integer containing the zero-indexed position of the character
 *        to return; charIndex must be >= 0 and < length of input string
 * @return a char containing the character from the input string at charIndex position
 */
char getCharacter (string input, int charIndex)
{
    return input.at( charIndex );
}

/*
 * Unit testing functions. Do not alter.
 */
void unittest ()
{
    cout << "\nSTARTING UNIT TEST\n\n";
```

```
    try {
        btassert<bool>(goldilocks("porridge", 2) == "This porridge is too cold");
        cout << "Passed TEST 1: goldilocks (porridge, 2)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 1 goldilocks (porridge, 2) #\n";
    }

    try {
        btassert<bool>(goldilocks("chair", 3) == "This chair is just right");
        cout << "Passed TEST 2: goldilocks (chair, 3)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 2 goldilocks (chair, 3) #\n";
    }

    try {
        btassert<bool>(goldilocks("bed", 1) == "This bed is too hard");
        cout << "Passed TEST 3: goldilocks (bed, 1)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 3 goldilocks (bed, 1) #\n";
    }

    try {
        btassert<bool>(rockScissorPaper('r', 'S') == 1);
        cout << "Passed TEST 4: rockScissorPaper (r, S)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 4 rockScissorPaper (r, S) #\n";
    }

    try {
        btassert<bool>(rockScissorPaper('R', 'p') == 2);
        cout << "Passed TEST 5: rockScissorPaper (R, p)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 5 rockScissorPaper (R, p) #\n";
    }

    try {
        btassert<bool>(rockScissorPaper('S', 'P') == 1);
        cout << "Passed TEST 6: rockScissorPaper (S, P)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 6 rockScissorPaper (S, P) #\n";
    }

    try {
        btassert<bool>(rockScissorPaper('r', 'r') == 3);
        cout << "Passed TEST 7: rockScissorPaper (r, r)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 7 rockScissorPaper (r, r) #\n";
    }
```

```
    try {
        btassert<bool>(charWithAsciiValueAsString('Z') == "Z 90");
        cout << "Passed TEST 8: charWithAsciiValueAsString (Z)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 8 charWithAsciiValueAsString (Z) #\n";
    }

    try {
        btassert<bool>(charWithAsciiValueAsString('a') == "a 97");
        cout << "Passed TEST 9: charWithAsciiValueAsString (a)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 9 charWithAsciiValueAsString (a) #\n";
    }

    try {
        btassert<bool>(toLower("HELLO") == "hello");
        cout << "Passed TEST 10: toLower (HELLO)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 10 toLower (HELLO) #\n";
    }

    try {
        btassert<bool>(toLower("gOOdbYe") == "goodbye");
        cout << "Passed TEST 11: toLower (gOOdbYe)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 11 toLower (gOOdbYe) #\n";
    }

    try {
        btassert<bool>(toUpper("hello") == "HELLO");
        cout << "Passed TEST 12: toUpper (hello)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 12 toUpper (hello) #\n";
    }

    try {
        btassert<bool>(toUpper("gOOdbYe") == "GOODBYE");
        cout << "Passed TEST 13: toUpper (gOOdbYe)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 13 toUpper (gOOdbYe) #\n";
    }

    try {
        btassert<bool>(getCharacter("amazing", 3) == 'z');
        cout << "Passed TEST 14: getCharacter (amazing, 3)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 14 #\n";
    }

    try {
```

```
        btassert<bool>(getCharacter("hooray!", 6) == '!');
        cout << "Passed TEST 15: getCharacter (hooray, 6)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 15 getCharacter (hooray, 6) #\n";
    }

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

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

```