```
/*
 * Programming Challenge 4
 */
#include <cassert>
#include <cstdlib>
#include <iostream>
#include <sstream>
using namespace std;

string makeString (string label, double value, char separator);
char stringToChar (string value);
int stringToInt (string value);
double stringToDouble (string value);
bool stringToBool (string value);

/* helper functions -- do not alter */
void clearScreen ();

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

int main (int argc, char* argv[])
{
    string input;
    cout << "[1] Do interactive test, [2] Skip to unit test: ";
    getline(cin, input);
    if (stringToInt(input) == 1)
        interactiveTest();

    unittest();

    return 0;
}


/*
 * Return a string comprised of a label, followed by a space,
 * followed by a separator character, followed by a space, followed
 * by a floating-point value. For example, label="Temperature", value=41.7,
 * separator=':' will return "Temperature : 41.7". Uses stringstream.
 * @param label the label for the value
 * @param value a double containing the value associated with the label
 * @param separator the character that will separate the label and the value
 * @return a string comprised of a label, followed by a space,
 *         followed by a separator character, followed by a space, followed
 *         by a floating-point value
 */
```

```
string makeString (string label, double value, char separator)
{
    stringstream ss;

    ss << label << " " << separator << " " << value;

    return ss.str();
}

/*
 * Useful when accepting input from stdin using the getline function.
 * Return the first character of a length 1 string. If the value is of
 * length 0 or of length > 1, return the null character ('\0').
 * @param value a string containing an expected single character
 * @return the first character of the string or null character ('\0')
 *         when value is length 0 or value is length > 1
 */
char stringToChar (string value)
{

    if( value.length() == 0 || value.length() > 1 ) {   return '\0';    }

    return value.at(0);

}

/*
 * Useful when accepting input from stdin using the getline function.
 * Convert a string containing an expected integer value (such
 * as a string captured from stdin) into an integer. If value is
 * not valid as an integer, return 0.
 * @param value a string containing an expected integer value
 * @return an integer representing the value, or 0 on failure
 */
int stringToInt (string value)
{
    // THIS FUNCTION PROVIDED AS AN EXAMPLE
    int ivalue = 0;
    stringstream converter(value);
    converter.exceptions(ios_base::failbit);

    try
    {
        converter >> ivalue;
    }
    catch (ios_base::failure f) {}

    return ivalue;
}
```

```
/*
 * Useful when accepting input from stdin using the getline function.
 * Convert a string containing an expected floating-point value (such
 * as a string captured from stdin) into a double. If value is
 * not valid as an double, return 0.
 * @param value a string containing an expected floating-point value
 * @return an double representing the value, or 0 on failure
 */
double stringToDouble (string value)
{

    double dblConvertedValue = 0.0;

    stringstream converter( value );
    converter.exceptions( ios_base::failbit );

    try {

        converter >> dblConvertedValue;

    } catch( ios_base::failure f )  {   }

    return dblConvertedValue;

}

/*
 * Useful when accepting input from stdin using the getline function.
 * Convert a string containing an boolean value (such
 * as a string captured from stdin) into a bool. Return true if the first
 * character is 'T' (case-insensitive), false if the first character is 'F'
 * (case-insensitive), and false on anything else.
 * @param value a string expected to start with either 'T' or 'F'
 * @return an bool if the first character is 'T' (case-insensitive), false
 *         if the first character is 'F' (case-insensitive), and false on
 *         anything else.
 */
bool stringToBool (string value)
{

    if( value.length() == 0 )   {   return false;   } else {

        char charDecision   = value.at(0);

        if( charDecision == 't' || charDecision == 'T' )    {   return true;    } else {    return false;   }

    }

}

```

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

void interactiveTest ()
{
    cout << "\nSTARTING INTERACTIVE TEST\n\n";

    bool quit = false;
    char c = 'z';
    string input;

    while (!quit)
    {
        cout << "! TRY EVERYTHING YOU CAN TO BREAK THESE ON BAD INPUT !\n\n";

        cout << "Enter a char ('z' to stop interactive test): ";
        getline(cin, input);
        c = stringToChar(input);
        if (c == '\0')
            cout << input << " not a valid char\n";
        else if (c == 'z')
            break;
        else
            cout << "Char input: " << c << endl;

        cout << "\nEnter an integer: ";
        getline(cin, input);
        cout << "Integer input: " << stringToInt(input) << endl;

        cout << "\nEnter a double: ";
        getline(cin, input);
        cout << "Double input: " << stringToDouble(input) << endl;

        cout << "\nEnter TRUE or FALSE: ";
        getline(cin, input);
        cout << "Boolean input: " << boolalpha << stringToBool(input) << endl;

        cout << "\nHIT ENTER TO CONTINUE";
        getline(cin, input);
        clearScreen();
    }

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

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

```

```
    try {
        btassert<bool>(makeString("Temperature", 42.6, ':') == "Temperature : 42.6");
        cout << "Passed TEST 1: makeString(Temperature, 42.6, ':')\n";
    } catch (bool b) {
        cout << "# FAILED TEST 1 makeString(Temperature, 42.6, ':') #\n";
    }

    try {
        btassert<bool>(makeString("", 75, ',') == " , 75");
        cout << "Passed TEST 2: makeString(\"\", 75, ',')\n";
    } catch (bool b) {
        cout << "# FAILED TEST 2 makeString(\"\", 75, ',') #\n";
    }

    try {
        btassert<bool>(makeString("Total", 100.05, '=') == "Total = 100.05");
        cout << "Passed TEST 3: makeString(Total, 100.05, '=')\n";
    } catch (bool b) {
        cout << "# FAILED TEST 3 makeString(Total, 100.05, '=') #\n";
    }

    try {
        btassert<bool>(stringToChar("") == '\0');
        cout << "Passed TEST 4: stringToChar(\"\")\n";
    } catch (bool b) {
        cout << "# FAILED TEST 4 stringToChar(\"\") #\n";
    }

    try {
        btassert<bool>(stringToChar("yn") == '\0');
        cout << "Passed TEST 5: stringToChar(yn)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 5 stringToChar(yn) #\n";
    }

    try {
        btassert<bool>(stringToChar("X") == 'X');
        cout << "Passed TEST 6: stringToChar(X)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 6 stringToChar(X) #\n";
    }

    try {
        btassert<bool>(stringToInt("42") == 42);
        cout << "Passed TEST 7: stringToInt(42)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 7 stringToInt(42) #\n";
    }

    try {
```

```
        btassert<bool>(stringToInt("hello") == 0);
        cout << "Passed TEST 8: stringToInt(hello)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 8 stringToInt(hello) #\n";
    }

    try {
        btassert<bool>(stringToInt("") == 0);
        cout << "Passed TEST 9: stringToInt(\"\")\n";
    } catch (bool b) {
        cout << "# FAILED TEST 9 stringToInt(\"\") #\n";
    }

    try {
        btassert<bool>(stringToDouble("") == 0);
        cout << "Passed TEST 10: stringToDouble(\"\")\n";
    } catch (bool b) {
        cout << "# FAILED TEST 10 stringToDouble(\"\") #\n";
    }

    try {
        btassert<bool>(stringToDouble("3.14") == 3.14);
        cout << "Passed TEST 11: stringToDouble(3.14)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 11 stringToDouble(3.14) #\n";
    }

    try {
        btassert<bool>(stringToDouble("hello") == 0);
        cout << "Passed TEST 12: stringToDouble(hello)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 12 stringToDouble(hello) #\n";
    }

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

    try {
        btassert<bool>(stringToBool("TrUe") == true);
        cout << "Passed TEST 14: stringToBool(TrUe)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 14 stringToBool(TrUe) #\n";
    }

    try {
        btassert<bool>(stringToBool("FALSE") == false);
```

```
        cout << "Passed TEST 15: stringToBool(FALSE)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 15 stringToBool(FALSE) #\n";
    }

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

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

void clearScreen ()
{
#ifdef WIN32
    system("cls");
#else
    system("clear");
#endif
}
```