```
 1  This file was updated on Saturday, 2012-11-24 at 9:32 AM
 2
 3
 4  ======================================================================
 5  RegexTests.cpp
 6  ======================================================================
 7
 8
 9  /**
10   *  File:  /~heines/91.204/91.204-2012-13f/204-lecs/code/BoostRegexTests/RegExTests.cpp
11   *  Jesse M. Heines, UMass Lowell Computer Science, heines@cs.uml.edu
12   *  Copyright (c) 2012 by Jesse M. Heines.  All rights reserved.  May be freely
13   *    copied or excerpted for educational purposes with credit to the author.
14   *  updated by JMH on November 19, 2012 at 8:12 PM
15   *  updated by JMH on November 23, 2012 at 9:58 PM
16   */
17
18  #include <iostream>  // for cout and friends
19  #include <sstream>   // for string streams
20  #include <string>    // for the STL string class
21
22  #include <boost/regex.hpp>              // for regex_match
23  #include <boost/algorithm/string.hpp>   // for iequals (case-insensitive match)
24    // the Boost string library contains many string manipulation functions not found in
25    //    the STL library that you may be familiar with from Java or JavaScript, such as
26    //    case-insensitive comparisons and trimming
27    // see http://stackoverflow.com/questions/11635/case-insensitive-string-comparison-in-c
28    // see http://www.boost.org/doc/libs/1_52_0/doc/html/string_algo/usage.html
29    // see http://www.boost.org/doc/libs/1_52_0/doc/html/string_algo/reference.html
30
31
32  using namespace std ;     // to eliminate the need for std::
33  using namespace boost ;   // to eliminate the need for boost::
34
35
36  // this function is modeled after code found in credit_card_example.cpp
37  void test1_BasicAndCaseInsensitiveMatches() {
38    // set up the strings to be tested
39    string str[ 10 ] ;
40    str[0] = "quit" ;
41    str[1] = "exit" ;
42    str[2] = "Quit" ;
43    str[3] = "quite" ;
44    str[4] = "unrequited" ;
45    str[5] = "unreQUITed" ;
46    int nStrings = 6 ;
47
48    // define a regular expression to test for "quit"
49    const boost::regex reQuit( "quit" ) ;
50    cout << "Results of case-sensitive searches:" << endl ;
51    // test each string against the regular expression
52    for ( int k = 0 ; k < nStrings ; k++ ) {
53      cout << "attempting to match \"" << str[k] << "\" to \"" << reQuit << "\" returned "
54           << boost::regex_match( str[k], reQuit ) << endl ;
55    }
56
57    // define a regular expression to test for "quit" anywhere in a string
58    const boost::regex reQuitA( ".*quit.*" ) ;
59    cout << "\nResults of anywhere searches:" << endl ;
60    // test each string against the regular expression
61    for ( int k = 0 ; k < nStrings ; k++ ) {
62      cout << "attempting to match \"" << str[k] << "\" to \"" << reQuit << "\" returned "
63           << boost::regex_match( str[k], reQuitA ) << endl ;
64    }
65
66    // define a case-insensitive regular expression to test for "quit" anywhere in a string
67    const boost::regex reQuitI( ".*quit.*", boost::regex::icase ) ;
68    cout << "\nResults of case-insensitive anywhere searches:" << endl ;
```

```
 69    // test each string against the regular expression
 70    for ( int k = 0 ; k < nStrings ; k++ ) {
 71      cout << "attempting to match \"" << str[k] << "\" to \"" << reQuitI << "\" returned "
 72          << boost::regex_match( str[k], reQuitI ) << endl ;
 73    }
 74  }
 75
 76
 77  // this function is modeled after code found in regex_match_example.cpp
 78  // note that this code demonstrates just one way to address the issue of parsing
 79  //    a command line using regular expressions, other approaches are not only
 80  //    possible, but perhaps even better
 81  void test2_BasicCommandParsing_v1() {
 82
 83    string strCmd[10] ;
 84    strCmd[0] = " add element root first one" ;
 85    strCmd[1] = "  add element root second" ;
 86    strCmd[2] = "  add  attribute  first   attr1 attr1value" ;
 87    strCmd[3] = "  add  attribute  second  attr2" ;
 88    strCmd[4] = "print" ;
 89    strCmd[5] = "quit" ;
 90    strCmd[6] = "another command" ;
 91    int nCmds = 7 ;
 92
 93    cmatch what;
 94    // what[0] contains the entire matched string
 95    // what[1] contains the first matched group
 96    // what[2] contains the second matched group
 97    // what[3] etc.
 98
 99    regex reBasicCmd( "^\\s*(add|print|quit).*", boost::regex::icase ) ;
100    regex reAddCmd( "^\\s*add\\s*(element|attribute)\\s(.+)\\s(.+)\\s*(.*)$", boost::regex::icase ) ;
101    regex reAddElementCmd( "^\\s*add\\s*element\\s(.+)\\s(.+)\\s*(.*)$", boost::regex::icase ) ;
102    regex reAddAttributeCmd( "^\\s*add\\s*attribute\\s(.+)\\s(.+)\\s*(.*)$", boost::regex::icase ) ;
103    regex reQuitCmd( "^\\s*quit", boost::regex::icase ) ;
104
105    // loop through all hard-coded command strings for testing purposes
106    for ( int n = 0 ; n < nCmds ; n++ ) {
107
108      // user entry point
109      cout << "\nYour command: " ;
110      // cin >> strCmd ;
111      cout << strCmd[n] << endl ;
112
113      // string version of a matched group
114      // for building a bridge between the cmatch type and an STL sting so that we can
115      //    process matches with STL string functions
116      string strWhat ;
117
118      // test for a match of a basic command
119      if ( regex_match( strCmd[n].c_str(), what, reBasicCmd ) ) {
120        cout << "  what.size() = " << what.size() << endl ;
121        for ( int k = 0 ; k < what.size() ; k++ ) {
122          strWhat = what[k] ;
123          cout << "    what[" << k << "] = " << what[k] << " (" << strWhat.size()  << " chars)" << endl ;
124        }
125
126        // handle an ADD command
127        if ( iequals( strWhat, "add" ) ) {
128          cout << "  Command is ADD" << endl ;
129
130          // test for a match on the second word in the command
131          if ( regex_match( strCmd[n].c_str(), what, reAddCmd ) ) {
132            for ( int k = 0 ; k < what.size() ; k++ ) {
133              strWhat = what[k] ;
134              cout << "    what[" << k << "] = " << what[k] << " (" << strWhat.size()  << " chars)" << endl ;
135            }
136            strWhat = what[1] ;
137
```

```
138           // handle an ADD ELEMENT command
139           if ( iequals( strWhat, "element" ) ) {
140             cout << "  Command is ADD ELEMENT" << endl ;
141             cout << "  Continue with adding an element here." << endl ;
142           }
143           // handle an ADD ATTRIBUTE command
144           else if ( iequals( strWhat, "attribute" ) ) {
145             cout << "  Command is ADD ATTRIBUTE" << endl ;
146             cout << "  Continue with adding an attribute here." << endl ;
147           }
148           // parsing error: ADD is followed by an invalid keyword
149           else {
150             cout << "    Invalid ADD command: 2nd word must be 'element' or 'attribute'." << endl ;
151           }
152         }
153         // parsing error: ADD command syntax does not match the regular expression
154         else {
155           cout << "    Invalid ADD command syntax." << endl ;
156         }
157       }
158
159       // handle a PRINT command
160       else if ( iequals( strWhat, "print" ) ) {
161         cout << "  Command is PRINT" << endl ;
162         cout << "  Call your print function here." << endl ;
163       }
164
165       // handle a QUIT command
166       else if ( iequals( strWhat, "quit" ) ) {
167         cout << "  Command is QUIT" << endl ;
168         cout << "  Goodbye." << endl ;
169         return ;
170       }
171
172       // parsing error: the first keyword is not ADD, PRINT, or QUIT
173       else {
174         cout << "  Invalid command: 1st word must be 'add', 'print', or 'quit'." << endl ;
175       }
176     }
177   }
178 }
179
180
181
182
183 // this function is modeled after code found in regex_match_example.cpp
184 // note that this code demonstrates just one way to address the issue of parsing
185 //    a command line using regular expressions, other approaches are not only
186 //    possible, but perhaps even better
187 void test2_BasicCommandParsing_v2() {
188
189   string strCmd[10] ;
190   strCmd[0] = " add element root first one" ;
191   strCmd[1] = "  add element root second" ;
192   strCmd[2] = "  add  attribute  first   attr1 attr1value" ;
193   strCmd[3] = "  add  attribute  second  attr2" ;
194   strCmd[4] = "print" ;
195   strCmd[5] = "a" ;
196   strCmd[6] = "ad" ;
197   strCmd[7] = "add" ;
198   strCmd[8] = "quit" ;
199   strCmd[9] = "another command" ;
200   int nCmds = 10 ;
201
202   cmatch what;
203   // what[0] contains the entire matched string
204   // what[1] contains the first matched group
205   // what[2] contains the second matched group
206   // what[3] etc.
```

```
207
208    regex reAddCmd( "^\\s*a(d|dd)?.*", boost::regex::icase ) ;
209    regex rePrintCmd( "^\\s*p(r|ri|rin|rint)?.*", boost::regex::icase ) ;
210    regex reQuitCmd( "^\\s*q(u|ui|uit)?.*", boost::regex::icase ) ;
211
212    // loop through all hard-coded command strings for testing purposes
213    for ( int n = 0 ; n < nCmds ; n++ ) {
214
215      // user entry point
216      cout << "\nYour command: " ;
217      // cin >> strCmd ;
218      cout << strCmd[n] << endl ;
219
220      // string version of a matched group
221      // for building a bridge between the cmatch type and an STL sting so that we can
222      //    process matches with STL string functions
223      string strWhat ;
224
225      // test for a match of an ADD command
226      if ( regex_match( strCmd[n].c_str(), what, reAddCmd ) ) {
227        cout << "  Command is ADD" << endl ;
228        cout << "  Call a function to do your add command processing here." << endl ;
229      }
230
231      // test for a match of a PRINT command
232      else if ( regex_match( strCmd[n].c_str(), what, rePrintCmd ) ) {
233        cout << "  Command is PRINT" << endl ;
234        cout << "  Call your print function here." << endl ;
235      }
236
237      // handle a QUIT command
238      else if ( regex_match( strCmd[n].c_str(), what, reQuitCmd ) ) {
239        cout << "  Command is QUIT" << endl ;
240        cout << "  Goodbye." << endl ;
241        return ;
242      }
243
244      // parsing error: the first keyword is not ADD, PRINT, or QUIT
245      else {
246        cout << "  Invalid command: 1st word must be 'add', 'print', or 'quit'." << endl ;
247      }
248    }
249  }
250
251
252  // standard C++ main function
253  int main( int argc, char* argv[] ) {
254    // test1_BasicAndCaseInsensitiveMatches() ;
255    // test2_BasicCommandParsing_v1() ;
256    test2_BasicCommandParsing_v2() ;
257
258    return 0 ;
259  }
260
261
262  ========================================================================
```