

This file was updated on Monday, 2015-11-23 at 12:44 PM

```
=====
main.cpp
=====
```

```
101  /*
102  * File:   main.cpp
103  * Author: Jesse Heines, heines@cs.uml.edu
104  *
105  * Created on November 23, 2015, 10:23 AM
106  */
107
108  #include <iostream> // for cout and friends
109  #include <sstream>   // for string streams
110  #include <string>    // for the STL string class
111  #include <regex>     // for regular expressions
112
113  #include "jmhUtilities.h"
114
115  using namespace std;
116
117  // forward references
118  void ParseCommandString( string strUserEntry );
119  void ProcessAddCommand( string strUserEntry );
120  void ProcessAddElementCommand( string strUserEntry );
121  void ProcessAddAttributeCommand( string strUserEntry );
122  void ProcessPrintCommand( string strUserEntry );
123
124
125  /**
126   * The standard C++ main function.
127   * @param argc number of command line arguments
128   * @param argv array of command line arguments
129   * @return
130   */
131  int main(int argc, char** argv) {
132
133      // display program title
134      cout << "===== " << endl ;
135      cout << "Assignment No. 7 Starter Code" << endl ;
136      cout << "===== " << endl ;
137
138      // string entered by the user in response to the command prompt
139      string strUserEntry = "" ;
140
141      // // prompt the user to enter a command string
142      // // version 1 without using a regular expression
143      // while ( ! jmhUtilities::caseInsensitiveCompare( jmhUtilities::trim( strUserEntry ), "quit" ) ) {
144      //     cout << "\nYour command: " ;
145      //     getline( cin, strUserEntry );
146      // }
147
148      // prompt the user to enter a command string
149      // version 2 using a regular expression
150      regex reQuit( "\\s*quit\\s*", regex::icase );
151      while( ! regex_match( strUserEntry, reQuit ) ) {
152          cout << "\nYour command: " ;
153          getline( cin, strUserEntry );
154
155          // if the user didn't enter 'quit', go parse the command string
156          if ( ! regex_match( strUserEntry, reQuit ) ) {
157              ParseCommandString( strUserEntry );
158          }
159      }
160
161      return EXIT_SUCCESS ;
162  }
163
164
```

```

165 /**
166  * Check for a valid basic command.
167  * @param strUserEntry command string entered by the user
168  */
169 void ParseCommandString( string strUserEntry ) {
170     // regular expressions for basic commands
171     regex reBasicAddCommand( "\\s*add.*", regex::icase );
172     regex reBasicPrintCommand( "\\s*print.*", regex::icase );
173     regex reBasicHelpCommand( "\\s*help.*", regex::icase );
174
175     // test for each basic command in turn
176     if ( regex_match( strUserEntry, reBasicAddCommand ) ) {
177         ProcessAddCommand( strUserEntry );
178     } else if ( regex_match( strUserEntry, reBasicPrintCommand ) ) {
179         ProcessPrintCommand( strUserEntry );
180     } else if ( regex_match( strUserEntry, reBasicHelpCommand ) ) {
181         cout << " Commands are:" << endl;
182         cout << "     add element parent_name element_name" << endl;
183         cout << "     add attribute parent_name attribute_name attribute_value" << endl;
184         cout << "     print" << endl;
185         cout << "     help (this command)" << endl;
186         cout << "     quit" << endl;
187     } else {
188         cout << " Invalid command. Acceptable commands are 'add', 'print', 'help', and 'quit'." << endl;
189     }
190 }
191
192
193 /**
194  * Handle an add command entered by the user
195  * @param strUserEntry command string entered by the user
196  */
197 void ProcessAddCommand( string strUserEntry ) {
198     // regular expressions for the second parameter in the add command
199     regex reAddElementCommand( "\\s*add\\s+element.*", regex::icase );
200     regex reAddAttributeCommand( "\\s*add\\s+attribute.*", regex::icase );
201
202     // test for each possible second parameter in turn
203     if ( regex_match( strUserEntry, reAddElementCommand ) ) {
204         ProcessAddElementCommand( strUserEntry );
205     } else if ( regex_match( strUserEntry, reAddAttributeCommand ) ) {
206         ProcessAddAttributeCommand( strUserEntry );
207     } else {
208         cout << " Invalid add command: 2nd parameter must be 'element' or 'attribute'." << endl;
209     }
210 }
211
212
213 /**
214  * Handle an add element command entered by the user
215  * @param strUserEntry command string entered by the user
216  */
217 void ProcessAddElementCommand( string strUserEntry ) {
218     // the what variable is actually an array that will be populated by the regex_match function
219     // when matched groups are found
220     cmatch what;
221     // what[0] contains the entire matched string
222     // what[1] contains the first matched group
223     // what[2] contains the second matched group
224     // what[3] etc.
225
226     // regular expression to pick out the name of the parent to which the new element is to be added
227     // and the name of the new element itself
228     regex reAddElementCmd( "^\\s*add\\s+element\\s(\\w+)\\s(\\w+)(.*)$", regex::icase );
229
230     // note that the following variant of the regex_match command requires a C string, not an STL string
231     if ( regex_match( strUserEntry.c_str(), what, reAddElementCmd ) ) {
232         cout << " You have specified that you want to add a new element named '" << what[2]
233         << "' to parent element '" << what[1] << "'." << endl;
234     } else {
235         cout << " Invalid 'add element' command." << endl;

```

```

236     cout << "      'add element' must be followed by two more parameters:" << endl ;
237     cout << "          (1) the name of the parent to which the new element is to be added, and" << endl ;
238     cout << "          (2) the name of the new element itself." << endl ;
239 }
240 }
241
242
243 /**
244  * Handle an add attribute command entered by the user
245  * @param strUserEntry command string entered by the user
246  */
247 void ProcessAddAttributeCommand( string strUserEntry ) {
248     // the what variable is actually an array that will be populated by the regex_match function
249     // when matched groups are found
250     cmatch what;
251     // what[0] contains the entire matched string
252     // what[1] contains the first matched group
253     // what[2] contains the second matched group
254     // what[3] etc.
255
256     // regular expression to pick out the name of the element to which the new attribute is to be added,
257     // the name of the new attribute, and the value of that attribute
258     regex reAddAttributeCmd( "^\\s*add\\s*attribute\\s(\\w+)\\s(\\w+)\\s(\\w+)(.*)$", regex::icase ) ;
259
260     // note that the following variant of the regex_match command requires a C string, not an STL string
261     if ( regex_match( strUserEntry.c_str(), what, reAddAttributeCmd ) ) {
262         cout << " You have specified that you want to add a new attribute named '" << what[2]
263             << "' with a value of '" << what[3] << "' to element '" << what[1] << "'." << endl ;
264     } else {
265         cout << " Invalid 'add attribute' command." << endl ;
266         cout << "      'add attribute' must be followed by three more parameters:" << endl ;
267         cout << "          (1) the name of the element to which the new attribute to be added," << endl ;
268         cout << "          (2) the name of the new attribute to be added, and " << endl ;
269         cout << "          (3) the value of the new attribute to be added." << endl ;
270     }
271 }
272
273
274 /**
275  * Handle a print command entered by the user
276  * @param strUserEntry command string entered by the user
277  */
278 void ProcessPrintCommand( string strUserEntry ) {
279     cout << " ... add your code to handle a print command here ..." << endl ;
280 }

```

```

=====
jmhUtilities.h
=====

```

```

301 /*
302  * File:   jmhUtilities.h
303  * Author: Jesse Heines, heines@cs.uml.edu
304  *
305  * Created on November 23, 2015, 10:46 AM
306  */
307
308 #ifndef JMHUTILITIES_H
309 #define JMHUTILITIES_H
310
311 #include <string>
312
313 using namespace std ;
314
315 class jmhUtilities {
316 public:
317     /**
318      * NetBeans-supplied default constructor.
319      */
320     jmhUtilities();

```

```

321
322  /**
323   * NetBeans-supplied copy constructor.
324   */
325   jmhUtilities(const jmhUtilities& orig);
326
327  /**
328   * NetBeans-supplied destructor.
329   */
330   virtual ~jmhUtilities();
331
332  /**
333   * Trim leading and trailing white space (spaces, tabs, and newlines) from the
334   *   string passed as an argument and return the trimmed string.
335   * This version is more sophisticated than the one above. It uses iterators
336   *   and an improved search technique.
337   * @param str string to trim (a copy, so that the original is not destroyed)
338   * @return a copy of the original string with leading and trailing white space removed
339   */
340   static string trim( string str ) ;
341
342  /**
343   * Compare two characters in a case-insensitive manner by converting them both to
344   *   uppercase and then testing if they are equal.
345   * @param a the first character to compare
346   * @param b the second character to compare
347   * @return true if the two uppercase characters are equal, false otherwise
348   * @see https://www.safaribooksonline.com/library/view/c-cookbook/0596007612/ch04s14.html
349   */
350   static inline bool caseInsCharCompareN(char a, char b) ;
351  /**
352   * Compare two strings in a case-insensitive manner using the companion case-insensitive
353   *   character comparison helper function.
354   * @param s1 the first string to compare
355   * @param s2 the second string to compare
356   * @return true if the two uppercase strings are equal, false otherwise
357   * @see https://www.safaribooksonline.com/library/view/c-cookbook/0596007612/ch04s14.html
358   */
359   static bool caseInsCompare(const string& s1, const string& s2) ;
360
361   private:
362
363 };
364
365 #endif /* JMHUTILITIES_H */

```

```

=====
jmhUtilities.cpp
=====

```

```

401  /**
402   * File:   jmhUtilities.cpp
403   * Author: Jesse Heines, heines@cs.uml.edu
404   *
405   * Created on November 23, 2015, 10:46 AM
406   */
407
408  #include "jmhUtilities.h"
409  #include <string>
410  using namespace std ;
411
412  /**
413   * NetBeans-supplied default constructor.
414   */
415  jmhUtilities::jmhUtilities() {
416  }
417

```

```

418 /**
419  * NetBeans-supplied copy constructor.
420  */
421 jmhUtilities::jmhUtilities(const jmhUtilities& orig) {
422 }
423
424 /**
425  * NetBeans-supplied destructor.
426  */
427 jmhUtilities::~jmhUtilities() {
428 }
429
430
431 /**
432  * Trim leading and trailing white space (spaces, tabs, and newlines) from the
433  * string passed as an argument and return the trimmed string.
434  * This version is more sophisticated than the one above. It uses iterators
435  * and an improved search technique.
436  * @param str string to trim (a copy, so that the original is not destroyed)
437  * @return a copy of the original string with leading and trailing white space removed
438  */
439 string jmhUtilities::trim( string str ) {
440     // define the iterator to be used in both loops
441     string::iterator it ;
442
443     // search for leading white space characters
444     it = str.begin() ;
445     while ( *it == ' ' || *it == '\t' || *it == '\n' ) {
446         str.erase( it ) ; // erase the found whitespace character
447         it = str.begin() ; // reposition the iterator to the first character in the string
448     }
449
450     // search for trailing white space characters
451     it = str.end() - 1 ;
452     while ( *it == ' ' || *it == '\t' || *it == '\n' ) {
453         str.erase( it ) ; // erase the found whitespace character
454         it = str.end() - 1 ; // reposition the iterator to the last character in the string
455     }
456
457     // return the result
458     return str ;
459 }
460
461
462 /**
463  * Compare two characters in a case-insensitive manner by converting them both to
464  * uppercase and then testing if they are equal.
465  * @param a the first character to compare
466  * @param b the second character to compare
467  * @return true if the two uppercase characters are equal, false otherwise
468  * @see https://www.safaribooksonline.com/library/view/c-cookbook/0596007612/ch04s14.html
469  */
470 inline bool jmhUtilities::caseInsCharCompareN(char a, char b) {
471     return ( toupper( a ) == toupper( b ) ) ;
472 }
473
474 /**
475  * Compare two strings in a case-insensitive manner using the companion case-insensitive
476  * character comparison helper function.
477  * @param s1 the first string to compare
478  * @param s2 the second string to compare
479  * @return true if the two uppercase strings are equal, false otherwise
480  * @see https://www.safaribooksonline.com/library/view/c-cookbook/0596007612/ch04s14.html
481  */
482 bool jmhUtilities::caseInsCompare( const string& s1, const string& s2 ) {
483     return ( ( s1.size() == s2.size() ) &&
484             equal( s1.begin(), s1.end(), s2.begin(), caseInsCharCompareN ) ) ;
485 }

```

=====