UNIVERSITY OF BIRMINGHAM

C++ for Finance
Assessment 1
Spell Checker

1222781

Date: Nov 2016

School of Social Sciences
University of Birmingham
Birmingham, B15 2TT

Contents

1 Briefing

In this assessment I have created a prototype for a spell checker. The program is written in ISO C++ 11 using DevC++ 5.11 compiler. The project contains four files :

- main.cpp
- small_dictionary.h
- small_dictionary.cpp
- utility.h

The program contains a dictionary of 200 common English words. The program use this dictionary to check if a word is correct or to provide suggestions for the words that do not exist in the dictionary. You can find the dictionary in the source code number 3.

2 Functionality

The program starts by asking the user to input a word. If the word was found in the dictionary, it would inform the user and then asks you to to either continue or quite the program. If you chose to continue, the process starts again, and inf you chose to exit the program, the program ends. This is demonstrated in the Figure 1

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Please enter your word :  follow
The word is found in the dictionary
Enter:
      -q :to quit the program
      -c :to continue entering words
your choice :  c
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Please enter your word :  be
The word is found in the dictionary
Enter:
      -q :to quit the program
      -c :to continue entering words
your choice :  q
```

Figure 1: Screen-shot of the console. The console closes after entering the q

However, if you enter a word that does not exist in the dictionary, the program will try to give you suggestions from the dictionary. It will then asks the user to chose one of the suggestions or proceed. To chose a suggestion one should enter the suggestion number or to proceed should enter an integer outside the suggestion range. If the user enters a suggestion number, the program will inform the user of the decision, and ask the user to either quit or continue with the next word. The program is designed to ignore inputs that are out of range. This processes can be seen in Figure 2

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Please enter your word :  follow
The word is found in the dictionary
Enter:
      -q :to quit the program
      -c :to continue entering words
your choice :  c
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Please enter your word :  ge
** The entered word could not be found in the dictionary **
The suggested word(s) are/is:
                             1-be
                             2-he
                             3-we
                             4-go
                             5-get
If you meant any of the suggestions enter the suggestion number
 otherwise press any integer to proceed

Enter your choice:  h

Enter your choice:  f

Enter your choice:  1
The correct word is now : <be>
Enter:
      -q :to quit the program
      -c :to continue entering words
your choice :  q
```

Figure 2: Screen-shot of the console. The console closes after entering the q

However if the user decides not to go for any of the suggested words and enter a number outside of the range, the program will ask the user to either add the new word to the dictionary or ignore the word. If the user likes to add the word to the dictionary, the user will be notified, but in either case the user will be asked to chose to either continue with the next word or to quit the program. This process is illustrated in Figure 3

In the same way as the previous part, the user input is checked for validity. If the user inputs a string that is not recognised by the program, the user will be asked to input another response. This also applies to the input at quit or continue level.

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Please enter your word :  gollow
** The entered word could not be found in the dictionary **
The suggested word(s) are/is:
                              1-follow
If you meant any of the suggestions enter the suggestion number
 otherwise press any integer to proceed

Enter your choice:  2
Enter:
      -add :to add the word to the dictionary
      -ignore :to ignore the mistake
your choice :  ignore
Enter:
      -q :to quit the program
      -c :to continue entering words
your choice :  c
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Please enter your word :  ge
** The entered word could not be found in the dictionary **
The suggested word(s) are/is:
                              1-be
                              2-he
                              3-we
                              4-go
                              5-get
If you meant any of the suggestions enter the suggestion number
 otherwise press any integer to proceed

Enter your choice:  7
Enter:
      -add :to add the word to the dictionary
      -ignore :to ignore the mistake
your choice :  add
The Word has been added to the dictionary
Enter:
      -q :to quit the program
      -c :to continue entering words
your choice :  c
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Please enter your word :  ge
The word is found in the dictionary
Enter:
      -q :to quit the program
      -c :to continue entering words
your choice :  q
```

Figure 3: Screen-shot of the console. The console closes after entering the q

3 suggestion creation

Creating a list of suggestions from the dictionary is the most sophisticated part of this program. In order to find suggestions, the program calculates the Damerau-Levenshtein distance between every word in the dictionary and the user's input. It will then supply the words in the dictionary that have the distance of at most 51% of the size of the input word as

suggestions.
The maximum distance has a linear positive relationship withe size of the input word. On the other hand the bigger the word is the more mistakes could happen and therefore more distance should be allowed. If the factor is much bigger than 51% it will give suggestions are are too far away from the input word. However if the factor is anything less than 51% the program will not suggest well for very small words. For example for two letter words, the maximum distance would be computed to be 0 and no word will be suggested. But with 51% the program suggest similar words to two digit words.

4 Source Codes

```
1  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
2  // main.cpp
3  // ID1222781
4  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
5  #include <iostream>
6  #include <string>
7  #include <algorithm>
8  #include "small_dictionary.h"
9  #include "utility.h"
10
11 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
12 //  main
13 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
14
15 int main ()
16 {
17    std::vector<std::string> Dict = dct::Dictionary200 ();
18    // creating the dictionary vector.
19    bool run = true;  // while run is true the following while loop, loops.
20
21    while(run)
22    {
23       ut::Printline(); //print a line to distinguish new word's journey
24       std::string input = ut::GetString("Please enter your word ");
25       //Ask the user for the input
26
27       if (dct::existance( Dict , input )) std::cout <<
28         "The word is found in the dictionary" << std::endl;
29
30       else
31       {
32          std::cout << "** The entered word could not be found in the dictionary ** \n";
33
34          std::vector<std::string> suggestions = dct::suggestions ( Dict , input );
35          //creating suggestions from the dictionary.
36          long SugSize = suggestions.size();
37          //find the size to see if there is any suggestions at all
38
39          bool correction = false; // true when the user has chosen a correction
40
41          if (SugSize > 0) // if there was any suggestion
42          {
43             std::cout << "The suggested word(s) are/is: \n";
44
45             for (long i = 0 ; i != SugSize ; ++i)
46             std::cout << "                              "<<
47             i+1 << "-" << suggestions[i] << std::endl;
48             //print the suggestions
49
50             std::cout << "If you meant any of the suggestions enter the suggestion number"
51             << std::endl << " otherwise press any integer to proceed " << std::endl;
52
53             long choice = ut::GetLong ("Enter your choice");
54
```

```
55             if (choice <= SugSize)
56             {
57                correction = true;
58                std::cout << "The correct word is now : <" << suggestions[choice-1] << ">\n";
59             }
60
61             else correction = false;
62          }
63
64          while (correction != true) //if the user did not use the sugestions loop starts.
65                                     // This loops until the user enters a valid input.
66          {
67             std::string add = "add" , ignore = "ignore";
68             std::cout << "Enter: \n" ;
69             std::cout << "      -add :to add the word to the dictionary \n" ;
70             std::cout << "      -ignore :to ignore the mistake" ;
71             std::string choice = ut::GetString("your choice ");
72
73             if (choice == add)
74             {
75                Dict.push_back(input);
76                std::cout << "The Word has been added to the dictionary \n" ;
77                break;
78             }
79             if (choice == ignore) break;
80          }
81
82       }
83
84       while (run) //this loops until a valid respond is captured
85       {
86          std::string quit = "q" , fcontinue = "c";
87          std::cout << "Enter: \n" ;
88          std::cout << "      -q :to quit the program \n" ;
89          std::cout << "      -c :to continue entering words" ;
90          std::string choice = ut::GetString("your choice ");
91
92          if (choice == quit) run = false;
93          if (choice == fcontinue) break;
94       }
95
96    }
97
98     return 0;
99 }
100
101 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
102 // end
103 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Source Code 1: main.cpp.

```
1  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
2  //  small_dictionary.h
3  // ID1222781
4  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
5
6  #include <iostream>
7  #include <string>
8  #include <vector>
9
10 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
11 //  namespace Dictionary
12 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
13 namespace Dictionary
14 {
15    std::vector<std::string> Dictionary200 ();
```

```
16     bool existance (std::vector<std::string> & Dict , std::string & input);
17     std::vector<std::string> suggestions (std::vector<std::string> & Dict , std::string & input);
18 }
19
20 namespace dct = Dictionary;     //alias
21
22 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
23 //   End
24 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Source Code 2: small_dictionary.h.

```
1  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
2  // small_dictionary.cpp
3  // ID1222781
4  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
5
6  #include "small_dictionary.h"
7  #include "utility.h"
8
9  #include <string>
10 #include <vector>
11 #include <algorithm>
12
13 long DLdistance (std::string A , std::string B);
14
15 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
16 // dict::Dictionary200 ()
17 //  returns a vector of first initial 200 words
18 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
19
20 std::vector<std::string> dct::Dictionary200 ()
21 {
22    std::vector<std::string> Dict
23    ({
24        "the", "be", "of", "and", "a", "to", "in", "he", "have", "it", "that",
25        "for", "they", "I", "with", "as", "not", "on", "she", "at", "by", "this",
26        "we", "you", "do", "but", "from", "or", "which", "one", "would", "all",
27        "will", "there", "say", "who", "make", "when", "can", "more", "if",
28        "no", "man", "out", "other", "so", "what", "time", "up", "go", "about",
29        "than", "into", "could", "state", "only", "new", "year", "some", "take",
30        "come", "these", "know", "see", "use", "get", "like", "then", "first",
31        "any", "work", "now", "may", "such", "give", "over", "think", "most",
32        "even", "find", "day", "also", "after", "way", "many", "must", "look",
33        "before", "great", "back", "through", "long", "where", "much", "should",
34        "well", "people", "down", "own", "just", "because", "good", "each",
35        "those", "feel", "seem", "how", "high", "too", "place", "little",
36        "world", "very", "still", "nation", "hand", "old", "life", "tell",
37        "write", "become", "here", "show", "house", "both", "between", "need",
38        "mean", "call", "develop", "under", "last", "right", "move", "thing",
39        "general", "school", "never", "same", "another", "begin", "while",
40        "number", "part", "turn", "real", "leave", "might", "want", "point",
41        "form", "off", "child", "few", "small", "since", "against", "ask",
42        "late", "home", "interest", "large", "person", "end", "open", "public",
43        "follow", "during", "present", "without", "again", "hold", "govern",
44        "around", "possible", "head", "consider", "word", "programme",
45        "problem", "however", "lead", "system", "set", "order", "eye", "plan",
46        "run", "keep", "face", "fact", "group", "play", "stand", "increase",
47        "early", "course", "change", "help", "line"
48    });
49    //The vector comes from the assessment given file.
50
51    return Dict;
52 }
53
54
55 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
// dict::existance ()
//  true if the input exists in the dictionary and false if it doesn't
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

bool dct::existance (std::vector<std::string>  & Dict , std::string & input)
{
   std::vector<std::string>::iterator it;
   it = find (Dict.begin(), Dict.end(), input); //find the location of the
                                     //input in the dictionary
   if (it != Dict.end()) return true; // if the pointer points at the end of
                              // the vector, input was not found
   else return false;
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
// disct::suggestions
//  give suggestive words from the dictionary
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

std::vector<std::string> dct::suggestions (std::vector<std::string> & Dict , std::string & input)
{
   std::vector<std::string> suggestions;

   long DicSize = Dict.size();
   for (long i = 0 ; i != DicSize ; ++i)
   {
      long Distance = DLdistance (input , Dict[i] );
      if (Distance < input.size()*0.51) suggestions.push_back(Dict[i]);
      // The suggestion holds the words with distance smaller than 51% of the input size.
      // The logic is: the bigger the word, the more mistakes can be made, the bigger the
      // distance is to the real word. 51% was chosen to cover two letter words.
      // anything less than 51% does not cover two letter words
   }
   return suggestions;
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
// DLdistance()
//  Find the distance between two strings
//  local function to the script
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
long DLdistance (std::string A , std::string B)
{
   long lal = A.size();        //number of characters in A
    long lbl = B.size();       //number of characters in B


    A.insert(0," ");
    B.insert(0," ");
    //inserting null characters. It helps to point at the right character when
    //pointing at the character number.

    std::vector< std::vector<long> > d_j_i; //the distance matrix


    std::vector<long> temp; //temporary vector defined local to the function

    for (long i = 0; i != lal+1 ; ++i)
    {
      temp.push_back(i);
   }
   d_j_i.push_back(temp); //filling the first row the matrix
   temp.clear();        //releasing the memory for temp



    for (long j = 1; j != lbl+1 ; ++j ) //filling the matrix row by row
    {
```

```
124
125       temp.push_back(j); // The first colume of the matrix is the 1,2,3,...
126
127       for (long i = 1; i != lal+1; ++i)
128       {
129          long option1 = temp[i-1]+1;
130          long option2 = d_j_i[j-1][i]+1;
131          long option3 = d_j_i[j-1][i-1];
132          if (A[i] != B[j]) ++option3;
133
134          long the_minimum; // The minimum will be allocated to the minimum option
135
136          if (i > 1 and j > 1 and A[i] == B[j-1] and A[i-1] == B[j])
137          {
138             long option4 = d_j_i[j-2][i-2]+1;  // Under this condition, and onother option exists
139             if (option1 < option2 and option1 < option3 and option1 < option4) the_minimum = ↩
                    option1;
140             if (option2 < option1 and option2 < option3 and option2 < option4) the_minimum = ↩
                    option2;
141             if (option3 < option1 and option3 < option2 and option3 < option4) the_minimum = ↩
                    option3;
142             else the_minimum = option4;
143          }
144
145          if (i == 1 or j == 1 or A[i] != B[j-1] or A[i-1] != B[j])
146          {
147             if (option1 < option2 and option1 < option3) the_minimum = option1;
148             if (option2 < option1 and option2 < option3) the_minimum = option2;
149             else the_minimum = option3;
150          }
151
152          temp.push_back(the_minimum); // The minimum is found and the next element of row is added↩
                 to the temp
153       }
154
155       d_j_i.push_back(temp); //The newly created row saved in the temp, is now pushed back to the ↩
              matrix.
156
157       temp.clear(); //Temporary file is cleared and memory is released.
158
159    }
160     return d_j_i[lbl][lal]; // The distance is defined as the final element of the matrix.
161 }
162 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
163 // end
164 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Source Code 3: small_dictionary.cpp.

```
1  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
2  //   utility.h
3  // ID1222781
4  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
5
6  #include <iostream>
7  #include <string>
8
9  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
10 //   namespace UtilityFunctions
11 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
12
13 namespace UtilityFunctions
14 {
15
16    std::string GetString(const std::string &);
17
18    long GetLong(const std::string &);
```

```
19
20      void Printline();
21  }
22
23  namespace ut = UtilityFunctions;     // alias
24
25  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
26  //   End
27  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Source Code 4: utility.h.

```
1   //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
2   //  utility.cpp
3   //  ID1222781
4   //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
5
6   #include "utility.h"
7
8   #include <string>
9   #include <iostream>
10  #include <stdexcept>
11  #include <iomanip>
12  #include <sstream>
13  #include <stdexcept>
14
15  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
16  // declaration of internal function
17  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
18
19  long StringToLong(const std::string & s, bool & success);
20
21  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
22  //  GetString()
23  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
24
25  std::string UtilityFunctions::GetString(const std::string & mess)
26  {
27      std::cout << std::endl << mess + ":  ";
28
29     std::string text;
30     std::cin >> text;
31      return text;
32  }
33
34  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
35  //  GetLong()
36  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
37
38  long UtilityFunctions::GetLong(const std::string & prompt)
39  {
40     bool valid_input = false;
41     long value;
42
43     while (valid_input == false)
44     {
45        std::string input = GetString(prompt);
46        value = StringToLong(input, valid_input);
47     }
48
49     return value;
50  }
51
52
53  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
54  //  Printline()
55  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
56
57 void UtilityFunctions::Printline()
58 {
59    std::cout <<
60    "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
61    << std::endl;
62 }
63
64 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
65 // StringToLong()
66 // Validation function for get long used to make sure user inputs long
67 // Function is taken from the course materials.
68 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
69
70 long StringToLong(const std::string & s, bool & success)
71 {
72     success = false;
73    std::istringstream i(s);
74    double x;
75    if (!(i >> x)) return long(x);
76
77    long y = long(x);
78    if (y == x) success = true;
79    return y;
80 }
81
82 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
83 // end
84 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Source Code 5: utility.cpp.