

**Students:**

This content is controlled by your instructor, and is not zyBooks content. Direct questions or concerns about this content to your instructor. If you have any technical issues with the zyLab submission system, use the **Trouble with lab** button at the bottom of the lab.

3.25 Lab 3: WordLadder - Stack, Queue, List, STL (Summer B 21)

Due Thursday August 5, 2021 at 11:59 PM - No Lab Demo, Grade based on Tests in zyBooks

Collaboration Policy Reminder

Students will work in pairs. But each student must submit their own solution for testing on zyBooks. You may not use code from any source (another student, a book, online, etc.) within your solution to this PROGRAM. In fact, you may not even look at another student's solution or partial solution to this PROGRAM. You also may not allow another student to look at any part of your solution to this exercise. You should get help on this lab from your partner or the teaching staff for the course. Make use of office hours by coming to the instructors' office hours as a pair, individually. Posting questions on Slack is encouraged (do not post your code publicly on Slack.)

Problem Definition

A word ladder is a word game. The player is given a start word and an end word, of the same length, and must find a "ladder" of words between the start and end words such that each word in the ladder is no more than one letter different than the word before it and the word after it. This off-by-one letter comparison is a simplified version of the edit distance between two strings. In computing, an edit distance is used to measure the similarity between two strings. The "distance" refers to the amount of work, or cost of the operations needed to transform one sequence of characters into the other. Common operations can include: replacement, insertion or removal of a character, insertion of one or more blank spaces.

Many computational problems use some variation of edit distance to gauge the similarity/dissimilarity of two or more sequences. For example, spell checking, plagiarism detection, text auto-complete, and DNA sequence alignment.

Example word ladders of the length 5:

For this lab, you will write a program to find the shortest word ladder given a start word and an end word. We will test on words of various lengths. So your solution should generalize. You can download the dictionary files for testing in the Student Materials Google Drive folder. There are dictionaries with words length 4, 5, 6, 7, and 8. Longer word ladders of words with higher length typically take longer to return a result using the algorithm above. Keep this in mind for initial testing. You may want to get your program working on short word ladders between words of size 4 and 5, before moving on to more difficult word ladders to find. The dictionary files are here: [Google Drive: dictionaries](#)

Algorithm: Find Word Ladder

```
Create a stack of strings.
Push the start word on this stack.
Create a queue of stacks.
Enqueue this stack.

while the queue is not empty
    for each word in the dictionary
        if a word is exactly 1 letter different than the top
string of the front stack
            then
                if this word is the end word
                    then
                        word ladder found, it is the front stack plus the
end word
                else
                    Make a copy of the front stack.
                    Push the found word onto the copy.
                    Enqueue the copy.
    Dequeue front stack.
end while loop
```

HINTS

1. The above pseudo-code, above, does not indicate where in the algorithm you should output either: a word ladder found, or "No Ladder found!" This is to allow you to consider this, and solve it yourself.
2. Also crucial will be the elimination of words that are already present in potential word ladders. Otherwise your program could toggle back and forth between two words with

For the test cases (100 pts), implement an STL-based solution only. Use an **STL list** (see below), **STL stack**, and **STL queue**. Pass all test cases. You must also implement a WordLadder class with the following public interface:

- WordLadder(const string &listFile) - Constructor that passes in the name of the dictionary file.
- void outputLadder(const string &start, const string &end) - passes in the start and end words and outputs to standard output the word ladder.

Dictionary Use and Storage

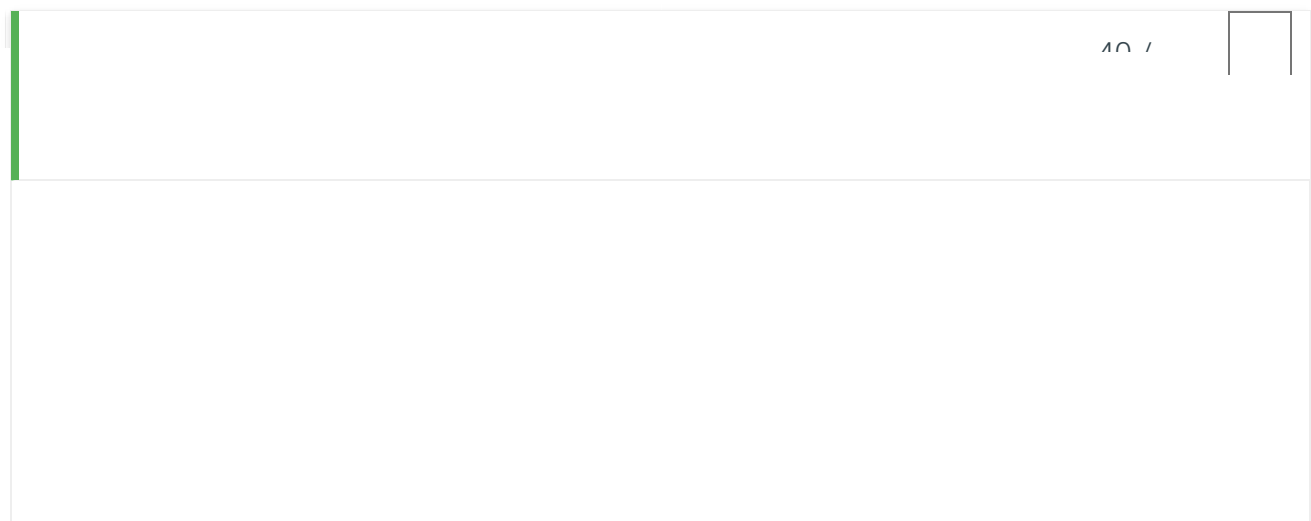
The dictionary is in a text file. Your program will read the dictionary from that file once and store it in a list. Since we will be reading each item in the dictionary a "list" data structure will be a suitable container to use. Once you begin to understand the algorithm, you will find that you can make this algorithm run much faster if you take the word out of the dictionary stored in memory, or at least ignore, any word you've added to a potential word ladder. Why is a linked list a good choice here? After initial insert of dictionary into a list object, what type of removals will be needed? How is the dictionary processed in the Word Ladder algorithm?

You are required to use **STL list** or **STL slist** (whichever is the most efficient possible given your algorithm) to store the dictionary. Do not make your own linked list. We have implemented linked lists extensively in the course so far. Now, I want you to gain practice using a production level library list implementation, i.e., the STL list along with its iterator.

It is highly suggested you get the WordLadder portion of the program working first using STL data structure classes. Then gain some success passing some of the 4 and 5 word test cases. Your WordLadder class must be able to be called via our test bench. Follow all naming conventions therein.

Turn-In: to receive credit, students must submit their source code in a zip file to the Canvas lab 3 link.

330538.1661232.qx3zqy7



```
98     cout << "A valid dictionary file was not provided." << endl;  
99     cout << "Valid files: *.txt, e.g., words4.txt, words5.txt, etc."  
100    exit(1);  
101    }  
102  
103    return 0;  
104 }  
105  
106  
107
```

Develop mode**Submit mode**

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

Run command

 Run program

Input (from above)

**main.cpp**
(Your program)

Program output displayed here

