Lab 8 - Vectors, and Debugging.

Directions

- The labs are marked based on attendance and effort. It is your responsibility to ensure the TA records your progress by the end of the lab.
- While completing these labs, you are encouraged to help your classmates and receive as much help as you like. Assignments, however, are individual work.
  You may not work on assignments during your lab time.

1. Collecting Phrase

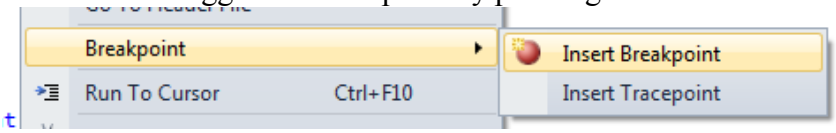
1. Write a program which repeatedly reads in phrases (strings) from the user until the user enters a 0-length string.
   - Phrases may have multiple words. What statement do you have to use to make it work?
   - Store each phrase in a vector.
   - Hint: You can find the length of a string using:
     ```
     string message =  "Hello";
     int daLength1 = message.size();
     ```
2. Write a function which accepts the vector of phrases as a parameter and returns the shortest phrase:
   ```
   string findShortest(vector<string> data);
   ```
   - Hint: You can find the length of the first element of the vector using:
     ```
     int daLength2 = data[0].size();
     ```
   - Have `main()` call `findShortestPhrase()` and output the phrase.
3. Sample output:

   ```
   Enter a phrase: Hello world!
   Enter a phrase: To be, or not to be?
   Enter a phrase: if (b || !b) ...
   Enter a phrase:
   findShortest() returned: Hello world!
   ```
4. Test your program.
   - Test with two elements
   - Test with three or more elements
5. **Understanding:**
   - How to create, write to, and read from vectors.
   - How to pass a vector into a function.

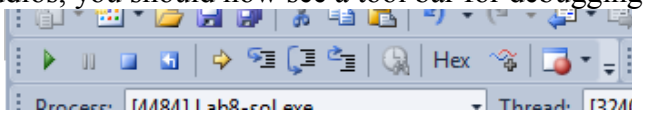
2. Debugger

Visual C++ has a built in debugger. This tool is very *very* **very** helpful when tracking down why your program is doing something wrong.

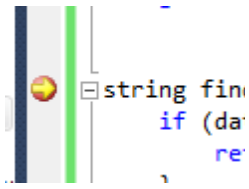
1. Using your phrase-reading program from above, go to the `findShortest()` function.
2. Right click on the first line of the function and select **Breakpoint --> Insert Breakpoint**.
   - You will see a red dot appear in the left margin of the code.
   - You can also toggle the breakpoint by pressing **F9**



3. Run your program by selecting **Debug --> Start debugging** (or press **F5**).
   - Enter in a few (about three) phrases and have the program find the shortest phrase.
   - Your program will seem to pause; the debugger has taken over and interrupted your program so you can debug it.
4. In Visual Studios, you should now see a tool bar for debugging (near top of screen).



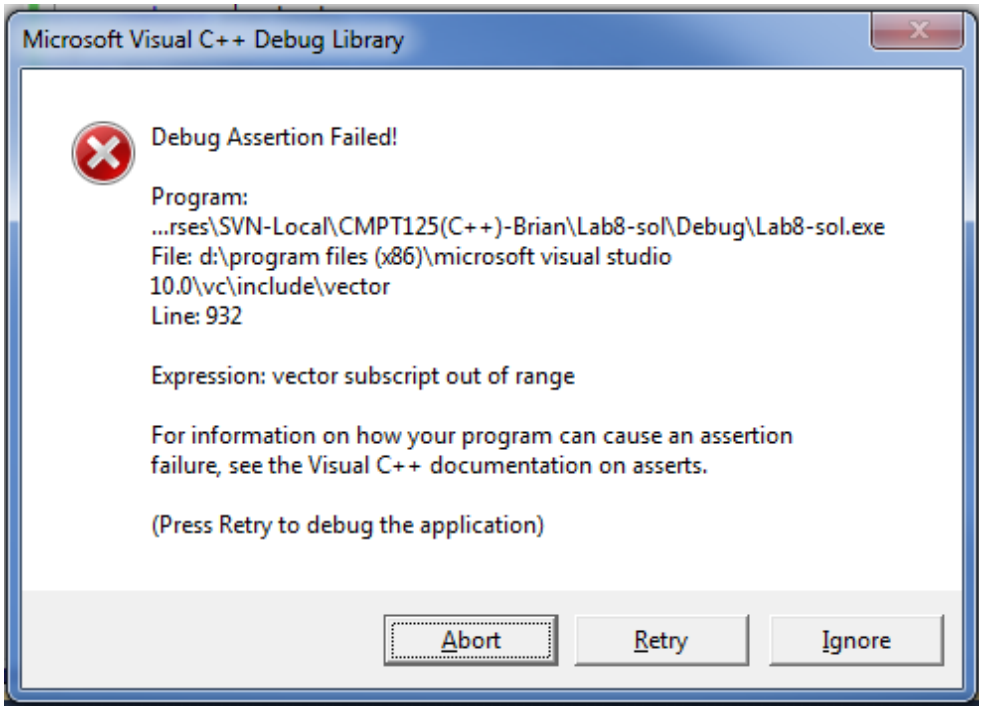
   - On the tool bar there is a green arrow for Continue (F5). If you click this, it will let your program run until it hits another breakpoint. This is useful if you want to let your program keep running.
   - The blue stop button completely stops your application and returns you to the normal coding setup within Visual Studios. This is useful if you have found the error or want to change something.
   - The Step Over button executes the current line of code you are on and then pauses again on the next line: useful for slowly single-stepping through your code.
     - Yes, this does actually execute the current line; it does not just skip it.
   - The Step Into button is like Step Over, except it will trace into any (and all) functions. Often it is better to use Step Over so you don't inadvertently end up debugging some function in vectors or strings.
5. Click the **Step Over** button a couple times.
   - You should see the yellow arrow on the left move to the next line of code each time you click Step Over (**F10**).



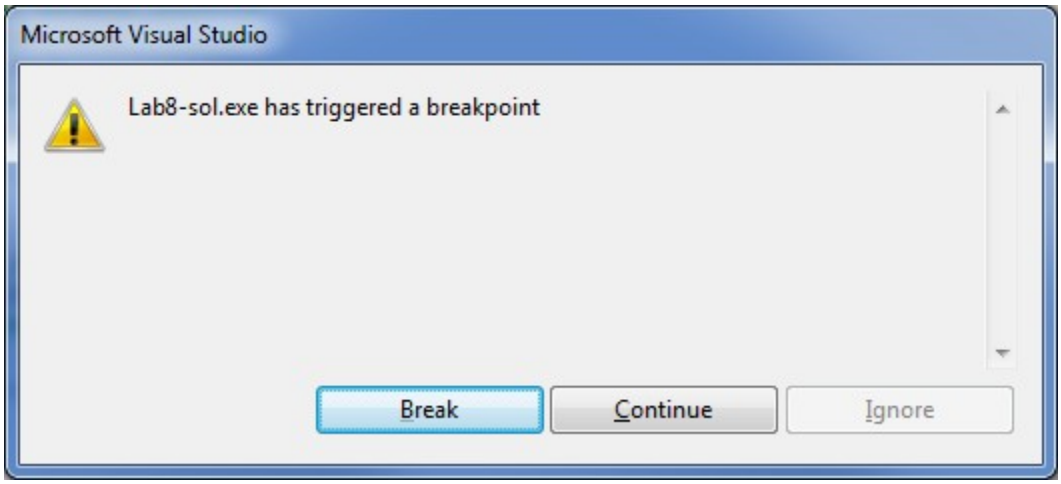
6. **Mouse over a variable** in your function.
   - You should see a little pop-up box that shows you the value of the variable your mouse is hovering over. Note that if your program has not yet created the variable, it may not have a valid value.

```
string shortest = data[0];
for (unsigned shortest "Hello world!" {
    if (data[i].size() < shortest.size()) {
```

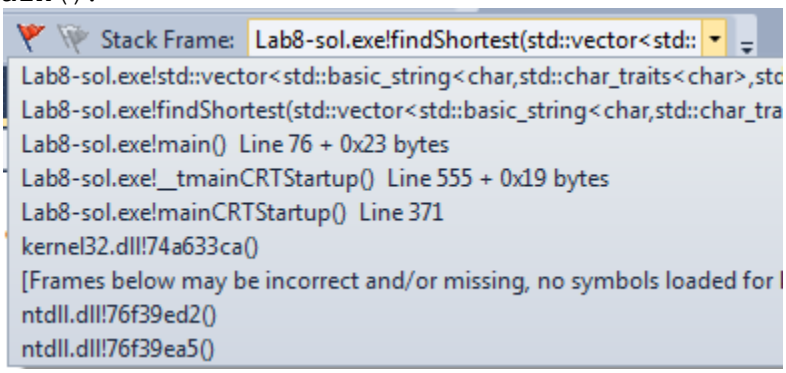
7. Notice the **Autos** view at the bottom shows some variables used by your code.
   - Use this for watching what is happening. For example, you can trace calculations, or watch your loop iteration variable to see what element of a vector it will access.
8. Use **Step Over** to keep stepping through the function until it exits. Notice you can follow it back into `main()`.
9. **Stop** the debugger with the blue stop button.
10. **Introduce a bug** into your `findShortest()` function. Instead of looping up until your counter is < the size of the vector, loop until <= size of vector.
    - Remove the breakpoint in `findShortest()`.
    - Re-run the program and create two phrases, then let it continue to find the shortest.
    - Your program should generate a run-time exception looking like the following. Select **Retry** to begin debugging the application.



- You should then see the following window. Select **Break.**



- You will now likely be looking at some complex code you did not write! This is some code inside the vectors. You need to change the stack frame.
- On the tool bar at the top, look for a drop-down named **"Stack Frame:"**. Try selecting a couple different options. Find one to point to `findShortest()`, and another to `main()`.



- The stack frame is which level of the function calls you are debugging. See which line it is pointing to in `findShortest()`; it should be the line where you tried to access one too many elements.

```
    // initialize answer to first element.
    string shortest = data[0];
    for (unsigned int i=1; i<=data.size(); i++) {
        if (data[i].size() < shortest.size()) {
            shortes i 3 [i];
```

- Stop the debugger, correct the code and re-run the program.

11. Understanding:

- Feel comfortable using the debugger to track down hard to understand bugs. If something is not working, trace it through and see what's going wrong.

3. Finding Phrases with a Vector

Continue the phrases program from above. **Use the debugger when you encounter any errors!**

1. Write a function which is passed the vector of phrases and returns the index of the shortest phrase:
   `int findShortestIdx(vector<string> data);`
   - Have `main()` call this function and output both the index of the element (an `int`), and the actual element (the phrase) itself.
   - Hint: You get the index from the return value of the function call, and using that index you can look-up the phrase in the vector.
2. Write a function which returns the index of the longest phrase:
   `int findLongestIdx(vector<string> data);`
   - Have `main()` call this function and output both the index of the element (an `int`), and the actual element (the phrase) itself.
3. Write a function which accepts a vector of phrases (`strings`) and swaps the longest and shortest phrases in the vector.
   - The function should change the actual vector you are passing in. How can you pass the vector in so that it can be changed?
   - Use some of the functions you have defined above to simplify your task.
   - Have `main()` call this function, and then repeat the calls to `findShortestIdx()` and `findLongestIdx()` and output the indices and elements as before.
4. Test your program.
   - Test with two elements: they should swap.
   - Test with three elements: the correct two should swap.
   - Sample output:

```
Enter a phrase: Hello world!
Enter a phrase: To be, or not to be?
Enter a phrase: if (b || !b) ...
Enter a phrase:
findShortest() returned: Hello world!
Shortest idx = 0 = Hello world!
Longest idx = 1 = To be, or not to be?
Shortest idx = 1 = Hello world!
Longest idx = 0 = To be, or not to be?
```

5. **Understanding:**
   - How to pass a vector to a function.
   - What is the difference between returning the element in a vector (in this case a string), and returning the index of an element (an int)?

4. Challenges

- Make the phrases program to work when the user enters no phrases.
- Investigate the use of step into, step over, and continue with your program.
- Experiment with the vector version of the phrase program. How do vectors appear in the debugger?
- When you have the debugger "broken-into" a function call, experiment with the "Stack Frame" drop-down box on the tool-bar.
  - It will allow you to select which function to look at in the debugger. It will show you the current function (current stack frame), or the calling function (previous stack-frame), and so on.
  - This can be very helpful when your function was passed in a bad value (such as an invalid parameter value).

5. Skills and Understanding

You should now be able to answer all the "understanding" questions in the previous sections. Complete the following to get credit for the lab:

- Show the TA the following:
  - Your operational programs which complete all of the above tasks.
  - The TA may ask you to explain any section of the lab, or answer any of the "Understanding" questions.
- **Nothing** is to be submitted electronically or in hard-copy for this lab.

