Lab 8: Vectors

1. Collecting Phrases

1. Create a new file for all your work in Lab 8 named: `lab8.cpp`
2. Write a function with the following prototype:

```
/*
 * Creates a vector of strings by asking the user to enter phrases.
 * Stops when user enters a phrase of length 0 characters.
 *   Returns: vector of phrases (strings)
 */
vector<string> createPhraseVector();
```

   - In the function, first create an empty vector of strings.
     Hint: `#include` both `vector` and `string`
   - Repeatedly read in a phrase from the user (using a loop) until the user enters a phrase of zero length (just hits enter).
     - Phrases may have multiple words. Use the following code to read in a full line of text into a string variable named `phrase`:

```
string phrase;
print("Enter a phrase: ");
getline(cin, phrase);
```

     - *Hint: Given a string such as:*
       `string message =  "Hello";`
       You can find the length of the string using:
       `int length = message.size();`

       Or, use the `empty()` member function to check if the string is zero length:
       `if (message.empty()) {...}`
   - Add each phrase into your vector of strings.
   - When your function finishes (i.e., when the user enters an empty phrase), *return* the vector of strings which you created.
3. Write a function named `displayPhrases()` which accepts as a parameter the vector of strings (phrases), and displays them. For example, for the following calling (client) code:

```
vector<string> messages {"Hello world", "Hi!", "Good bye"};
displayPhrases(messages);
```

   Print the following output:

```
Phrases:
          "Hello world"
          "Hi!"
          "Good bye"
```

   The sample code here is good for testing. However, your client code (the code that calls this function, i.e. `main()`) will not hard-code the messages vector; it will instead call `createPhraseVector()` to create it.

4. Have `main()` call `createPhraseVector()` to get phrases from the user, store the `vector` in a local variable in `main()`, and then pass it to `displayPhrases()`.
   - Test your program with one phrase, two phrases, zero phrases, multi-word phrases, single word phrases, and single character phrases.
   - Sample test run:

```
Enter a phrase: It's a beautiful day!
Enter a phrase: Hello world.
Enter a phrase: I code; therefore I am.
Enter a phrase:
Phrases:
          "It's a beautiful day!"
          "Hello world."
          "I code; therefore I am."
```

5. **Understanding:**
   - Briefly state how a function (like `createPhraseVector()`) can create a vector and then "give" it to the calling code (`main()`).
   - Briefly explain how a vector in `main()` can be given to a function like `displayPhrases()` to use.

2. Finding Phrases with a Vector

Continue the phrases program from above.

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 have `main()` 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. You must not have any `print` statements in `findShortestIdx()`; the `print()` call must be in `main()`.*
   - *Hint: To find the length of the first string in the vector named `myPhrases` use:*
     `int daFirstLength = myPhrases.at(0).size();`
     So, use a loop and check out the length of each string, looking for the shortest.
   - *Hint: To find the shortest string, first create a variable that tracks the index of the shortest string so far. Initialize it to the first string's index. Then as you go through all the strings compare the shortest so far to the new string. If the new string is shorter, save its index in your variable.*
2. Write a function which returns the index of the longest phrase:
   `int findLongestIdx(vector<string> data);`
   - Have `main()` call this function, and have `main()` 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? *Hint: It's not pass-by-value.*
- Use some of the functions you have defined above to simplify your task.
- Have `main()` call this function.

4. Add code to `main()` to print the phrases again.
   Hint: you already have this function! Just call it!
5. Test your program.
   - Test with two elements: they should swap.
   - Test with three elements: the correct two should swap.
   - Test it with one element: it should behave OK.
   - Getting it working with zero elements is the next section!
   - Sample output (input shown in bold-underlined):

```
Enter a phrase: It's a beautiful day!
Enter a phrase: Hello world.
Enter a phrase: I code; therefore I am.
Enter a phrase:
Phrases:
          "It's a beautiful day!"
          "Hello world."
          "I code; therefore I am."
Shortest idx = 1 = Hello world.
Longest idx  = 2 = I code; therefore I am.
Phrases:
          "It's a beautiful day!"
          "I code; therefore I am."
          "Hello world."
```

6. **Understanding:**
   - What is the difference between an element in a vector, and the element's index? What is the data type of the index? What is the data type of the element (for this lab)?
   - How can we pass a vector to a function so it can be changed by the function? If you did this incorrectly (and the function could not change the actual vector), how would this bug be visible while testing the program?

3. Challenges (optional)

1. Make the phrases program work when the user enters no phrases (zero length vector).
2. Make the phrase program trim blank spaces from the front and end of each phrase (if any).
3. For any non-empty phrase, require the user to enter at least two words.
4. ***Harder***: Can you use your swap function to sort all the phrases by their length?

4. Lab credit

- Complete above steps in one file. OK to skip any “optional” sections.
- Add brief answers to your .cpp file for each of the understanding questions.
- Comment your code correctly: comment for every ~2-5 lines of code.
- Submit your work to CourSys.