Introduction to Programming
CS1336

Assignment #9 -- Arrays

Assignment #9 -- Arrays

Introduction

Your ninth programming assignment will consist of two C++programs. Your programs should compile correctly and produce the specified output.

Please note that your programs should comply with the commenting and formatting rules we discussed in class. Please see the descriptive file on eLearning for details

Program 1 – Linear Search Algorithm

In Computer Science, it is often very important to be able to locate a specific data item inside a list or collection of data. Algorithms that perform this function are called *searching algorithms*, and there are many such algorithms in Computer Science.

Although it is inefficient, one of the most common searching algorithms is called Linear Search. In Linear Search we have a set of data that serves as the standard, usually stored within an array, and a separate *value* that we are searching for within that data set. We'd like to know whether the value is within the data set, so we scan through the data set looking for it, one element at a time, starting at the beginning of the array and proceeding, if necessary, to the very last element. If the value is found within the standard array, we return a number indicating its index position within the array and terminate further processing. (Note that it is important that we terminate the loop if the value is found. Otherwise the algorithm would return the last instance of the value instead of the first instance.) If the value is not found, we return an error indicator, oftentimes a -1, that indicates the value was not in the data set.

For this problem, please implement a linear search algorithm that performs this function. You will be given two input files, "`LSStandard.txt`" and "`LSTest.txt`". The `LSStandard.txt` file contains integer values against which we are searching. (There will be no more than 100 of these.) The `LSTest.txt` file contains a set of numbers that we are trying to locate within the standard data set. (There will be no more than 50 of these.) Read both of these into separate arrays and then determine which of the numbers in the LSTest file are included in the LSStandard data set by using a Linear Search algorithm. Have your program print out a report (to the console only is sufficient) that indicates whether the number was found or not.

Your output should look something like:

```
Number 1 ( 79) was located at index 44.
Number 2 ( 74) was not in the file.
```

```
Number 3 ( 56) was not in the file.
Number 4 (103) was located at index 75.
etc.
```

Note that the number for which we searched is indicated in parenthesis in the report. The "index" number refers to the index of the element within the LSStandard data.

Your function header for the Linear Search function should look like:

```
int searchList(int stdList [], int numElems, int value)
```

You'll notice that this function accepts an array as input parameter. That array, called "stdList" in the parameter list, will be the array that contains the standard data set. The parameter "numElems" is the number of elements in that array, and the parameter "value" is the element that we are searching for.

Your function should search for "value" within the "stdList" array and return one of two answers: (a) a -1 if "value" is not in "stdList", or (b) the index position of "value" within "stdList" if "value" is in "stdList". (This should be a number between 0 and (numElems-1).) Your program should then use that result to determine what should be printed in the report.

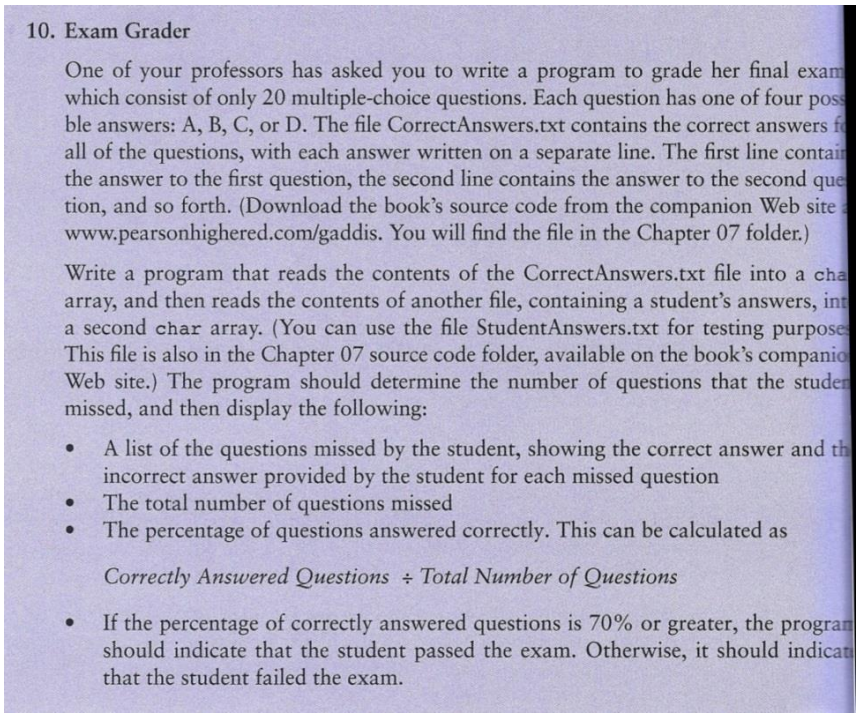
Note that some of the numbers in LSTest appear more than once in LSStandard. The Linear Search algorithm as we discussed in class will return the first found instance, and that's perfectly okay for the basic assignment. However, you can get 5 bonus points if your program can find and report the indices of all instances of a number that were found in LSStandard. In this case, your output could look something like:

```
Number 1 ( 79) was located at index(es) 44, 47.
Number 2 ( 74) was not in the file.
Number 3 ( 56) was not in the file.
Number 4 (103) was located at index 75, 82, 93.
etc.
```

If you have to modify the input parameters of `searchList` to solve this bonus problem that would be okay. However, as a hint, one could use a static local variable to solve the problem without changing the prototype as given.

Program 2 – Exam Grader

With a couple of changes, please implement Problem #11 on page 457 of Gaddis, 9th Edition (Problem #11, p. 451 of 8E). Note that the problem statement from the book instructs you to download the input files from the Pearson Web site. Instead, you will be provided with the two input files: "`CorrectAnswers.txt`" and "`StudentAnswers.txt`," on eLearning. Please use those files instead of the ones from Pearson. In addition, note that the scan of the problem below is from the 7th, and not the 9th, edition of the book. We left it this way because the 9th edition splits the problem between two pages. The problem is the same in both editions.



10. Exam Grader

One of your professors has asked you to write a program to grade her final exam which consist of only 20 multiple-choice questions. Each question has one of four possible answers: A, B, C, or D. The file CorrectAnswers.txt contains the correct answers for all of the questions, with each answer written on a separate line. The first line contains the answer to the first question, the second line contains the answer to the second question, and so forth. (Download the book's source code from the companion Web site at www.pearsonhighered.com/gaddis. You will find the file in the Chapter 07 folder.)

Write a program that reads the contents of the CorrectAnswers.txt file into a `char` array, and then reads the contents of another file, containing a student's answers, into a second `char` array. (You can use the file StudentAnswers.txt for testing purposes. This file is also in the Chapter 07 source code folder, available on the book's companion Web site.) The program should determine the number of questions that the student missed, and then display the following:

- A list of the questions missed by the student, showing the correct answer and the incorrect answer provided by the student for each missed question
- The total number of questions missed
- The percentage of questions answered correctly. This can be calculated as

  Correctly Answered Questions ÷ Total Number of Questions

- If the percentage of correctly answered questions is 70% or greater, the program should indicate that the student passed the exam. Otherwise, it should indicate that the student failed the exam.

The `CorrectAnswers` file contains exactly 20 answers that represent the correct answers for each question. (That is, the first entry in the file is the correct answer for Problem 1, the second entry is the correct answer for Problem 2, etc.) This file contains, in effect, the answer key for the exam, one answer per line.

The `StudentAnswers` file contains the answers the students actually entered on the exam, again, one answer per line. Each set of 20 lines in that file represents the answers for one student. We won't specify how many students are represented in the `StudentAnswers` file (your program, therefore, needs to be able to detect this), but we will stipulate that there will not be any partial sets in that file.

Your program should first read the answers in `CorrectAnswers.txt` into a character array. This array will serve as the answer key for the exam. Your program should then read the answers from the `StudentAnswers` file into another character array. You may safely use 300 as the size declarator for that array. Your program should then close both of the input files. All subsequent processing will be done on the arrays only.

After all the input data have been read into the arrays, your program can begin to process the student answers array. (As we know, every set of 20 entries represents the answers for one student.) As your program examines the next answer, have it determine whether the answer was correct by comparing it against the proper entry in the answer key. Use accumulators to track the number of questions missed for each student so bullet points 2, 3, and 4 above can be answered correctly. Then print out your report for that student, reset the accumulators and counters, and proceed to the next student.

Note that two questions have been added to the assignment in addition to the ones specified in the Gaddis problem. After all the student grades have been processed, have your program print a brief report indicating which student had the best grade and what that grade was, and which student had the worst grade and what that grade was.

Your output should look similar to the following:

```
Report for Student 1:
---------------------
Missed 3 out of 20 questions for 85% correct.
Questions missed:
   2(A/B), 5(C/A), 15(D/A)
This student passed the exam!


Report for Student 2:
---------------------
Missed 8 out of 20 questions for 60% correct.
Questions missed:
   3(A/D), 5(B/C), 8(C/D), 9(C/B), 12(A/B), 14(A/C), 18(B/C), 19(D/A)
This student failed the exam.

Student 2 had the best grade with 100%.
Student 5 had the worst grade with 57%.
```

Note that the first bullet point in the problem contains a requirement that the program print out not only which questions were missed but also what the right and wrong answers were. Let's handle that by using the same output format that was used in the midterm exam report: "2(A/C)" indicates that the student missed question #2, that "A" was the (incorrect) student response, and "C" was the correct response for that question.

Note that a difficult problem in any programming language is the conversion of numbers to strings and vice versa. Fortunately, the C++11 standard includes a very convenient conversion function called:

```
string to_string (<number>);
```

If you send in a number as an argument, `to_string` will return the C++ string equivalent of that number. This should help in creating the display of missed questions.

Although it is not indicated in the problem, please print your output both to the screen and to an output file named "`ExamAnalysis.txt`". Please submit the output file along with your program.