

C-Style Strings, Structs, Sorting

This programming assignment uses many of the ideas presented in sections 8 through 11 of the course notes, so you are advised to read those notes carefully as well as the following program specification.

The Program Specification:

Palindromes

A palindrome is a word or phrase that is spelled the same backwards as it is forwards (ignoring spacing, punctuation, and capitalization). One of the most common examples is the word "radar". For this assignment, you will read in strings from an input file, determine if they are palindromes, and print out the results. However, unlike previous programs, the output will not be in the same order as the input - instead, the strings will be printed out in alphabetical order.

In the previous programs, a value was read in, calculations done, and the results printed out. Since the calculations were complete, the variables that were used could then be reused for the next line of input, overwriting the previous values. However, in this assignment, we need to read all the input into the program before we can sort it - we need some way to store our data.

The solution to this problem is to use an array - but what kind? Your initial thought might be an array of character strings; that is, after all, the data that we are using. However, there is an additional problem. There needs to be two arrays for each input string - one to hold the string, and one to hold a formatted copy of the string, with whitespace, punctuation, and capitalization removed. So, instead, we will create a struct to hold each input string. At a **minimum**, each struct will contain the following:

- A character array to hold the input string
- An integer for the length of the input string
- A second character array to hold the formatted string
- Another integer for the length of the formatted string
- A boolean to indicate if the string is a palindrome

This is the minimum number of fields that need to be in the struct - if you find use for others, feel free to add them (this problem can actually be solved with less, but this is an exercise in using structs). Now that we have a struct, we can create an array of them to store all the input.

Input file description and samples:

This assignment has perhaps the simplest input file of all. Each string is on a single line by itself. There are no headers or other information that must be ignored. The strings are in an input file named, "palindrome.dat" - use of any other name will result in a score of 0.

A sample input file would look like:

```
Hello World!!!  
Radar  
A man, a plan, a canal, Panama!!!  
She sells sea shells by the sea shore
```

The maximum length for any given string will be 256 characters, and the maximum number of strings will be 50.

Processing

OK, now that we have read the string in, we need to get it into a format so that we can test it to see if it's a palindrome. With a palindrome, we ignore whitespace, punctuation, and capitalization. So, what we need to do is copy the string to a second array, but remove any non-alphabetic character (whitespace and punctuation). Additionally, those alphabetic characters that we do copy over need to be forced to be the same case - either upper or lower. Luckily, C++ provides us with several functions in the `<cctype>` header file (yes, that's two c's) to do just that:

- `isalpha(char)` - takes a single character and returns a boolean to indicate if it's an alphabetic character
- `tolower (char)` - takes a single character and returns the lowercase version of it
- `toupper (char)` - takes a single character and returns the uppercase version of it

Note: you only need to worry about one case, it's your decision which to use. More on these functions can be found on pages 516-518 of the textbook.

Now that we have a formatted version of the string, how do we determine if it's a palindrome? There are several ways (some more elegant than others), and we will go over some ideas in class.

Sorting

You will use the bubble sort algorithm to sort the string in alphabetical order. We will discuss the bubble sort in class, and additionally, the code is in the course notes (Chapter 11, slide 10).

It is important to realize that you can't just blindly copy the code and expect it to work. The code example uses an array of integers, but this assignment uses an array of structs, which are being sorted based on strings. Strings can't be compared like integers, using the `>`, `<` and `==` operators. Instead, use the appropriate string function(s). However, the changes should be **very minor**, as long as you **think** through the problem and don't just mindlessly follow the example.

If you do use an modified version of the code in the course notes, make sure you properly cite it in your function comments.

Output description and sample:

For the given input file, the output file, "palindrome.out", should look like:

```
Programmer:  Chris Knestrick
Palindromes

Palindrome      String
-----
      Y          A man, a plan, a canal, Panama!!!
      N          Hello World!!!
      Y          Radar
      N          She sells sea shells by the sea shore
-----
```

The first column of output should contain either a "Y" or an "N" (uppercase), to indicate if a string is a palindrome, followed by a copy of that string (unformatted, please :-). While you do not need to use **exact** horizontal spacing, your output should line up neatly in columns.

Programming Standards:

You'll be expected to observe good programming/documentation standards. All the discussions in class about formatting, structure, and commenting your code should be followed.

Documentation:

- You must include the honor pledge in your program header comment.
- Your header comment must describe what your program does.
- You must include a comment explaining the purpose of every variable or named constant you use in your program.
- You must use meaningful identifier names that suggest the meaning or purpose of the constant, variable, function, etc.
- Precede every major block of your code with a comment explaining its purpose.
- Precede every function you write with a header comment. This should explain in one sentence what the function does, then describe the logical purpose of each parameter (if any), describe the return value (if any), and any side effects.
- You must use indentation and blank lines to make control structures like loops and if-else statements more readable.

Coding:

- Use named constants instead of variables where appropriate.
- You must make good use of user-defined functions in your design and implementation. To encourage this, the body of `main()` must contain no more than 20 executable statements and the bodies of the other functions you write must each contain no more than 40 executable statements. An executable statement is any statement **other than** a constant or variable declaration, function prototype or comment. Blank lines do not count.
- You must use a struct, with the fields described at the beginning of the spec.
- You must use bubble sort to sort the array.
- **You must write at least seven functions, besides `main`.** The return type of each function is up to you.
- The definition of `main()` must be the first function definition in your source file. You may use file-scoped function prototypes and you may use file-scoped constants.
- You may not use file-scoped variables of any kind.
- Function parameters should be passed appropriately. Use pass-by-reference only when the called function needs to modify the parameter. Pass array parameters by constant reference (using `const`) when pass-by-reference is not needed.

Your submission that receives the highest score will be graded for adherence to these requirements, whether it is your last submission or not. If two or more of your submissions are tied for highest, the earliest of those will be graded. Therefore: implement and comment your C++ source code with these requirements in mind from the beginning rather than planning to clean up and add comments later.

This assignment will be worth 12% of your final grade. Of that, 80% will be based on your output (graded by the EAGS), and 20% will be based on your adherence to the programming standards (graded by hand).

This assignment will be due by midnight on Thursday, August 10. The drop dead date (resulting in a loss of 20 points) will be midnight on Friday, August 11 (the exam day).

Testing:

Obviously, you should be certain that your program produces the output given above when you use the given input file. However, verifying that your program produces correct results on a single test case does not constitute a satisfactory testing regimen.

At minimum, you should test your program on **all** the posted input/output examples given along with this specification. The same program that will be used to test your solution generated those input/output examples. You could make up and try additional input files as well; of course, you'll have to determine by hand what the correct output would be.

Pledge:

Each of your submissions to the EAGS must be pledged to conform to the Honor Code requirements for this course. Specifically, you **must** include the following pledge statement in the header comment for your program:

```
// On my honor:  
//  
// - I have not discussed the C++ language code in my program with  
// anyone other than my instructor or the teaching assistants  
// assigned to this course.  
//  
// - I have not used C++ language code obtained from another student,  
// or any other unauthorized source, either modified or unmodified.  
//  
// - If any C++ language code or documentation used in my program  
// was obtained from another source, such as a text book or course  
// notes, that has been clearly noted with a proper citation in  
// the comments of my program.  
//  
// - I have not designed this program in such a way as to defeat or  
// interfere with the normal operation of the Automated Grader.
```

Failure to include this pledge in a submission is a violation of the Honor Code.