Project One: Palindrome Analysis Toolkit

Out: Feb. 24, 2024; Due: Mar. 6, 2024

Introduction

This project will give you experience in writing, compiling, and debugging a C++ program. You will gain experience with header files and multiple-file compilation. We expect the program itself to be straightforward for any student who has successfully completed the prerequisites for this course.

Palindrome

Palindromes are strings that are read the same either forwards or backwards. Here are examples of palindromes:

12321, 2157512, abcba, en89e98ne

The following stringss are not palindromes: 123, 4565, asdfg.

Given a string, part of it can be a palindrome. E.g. the longest palindrome in 4565 is 565.

Programming Assignment

You will first implement a function that determine whether a given string is a palindrome or not, and then another function that will find the longest palindrome within the given string. Note that when there is more than one longest palindrome within the given string, the first one should be returned.

Integration within the overall program will be handled by our driver program provided upon submission to JOJ, so your focus will be solely on these two functions' implementations.

The declarations of the two functions are already written for you in `p1.h`, given by:

****

```
bool isPalindrome(const std::string& str);
// EFFECTS: check if a given string is a palindrome.
// If the given string is a palindrome, returns true.
// Otherwise, returns false.

std::string findLongestSubPalindrome(const std::string& str);
// EFFECTS: return the longest sub-palindrome within a given string.
// If multiple longest sub-palindromes are found, return the first
one.
```

Files

There are several files located in the Project-One-Related-Files.zip:

`p1.h`	The header file for the two functions you are to write. This is included (via `#include "p1.h"`) in the driver program.
`simpledriver.cpp`	A sample driver program for you to test two functions. It contains a sample main function.
`pretest_cases`	A directory containing four sample test cases for you.

You should copy the above files into your working directory. **DO NOT modify p1.h!**

You should put **all** of the functions you write in a single file, called `p1.cpp`. You may only `#include <string>` in your `p1.cpp`. No other system header files may be included, and you may not make any calls to any function in any other libraries.

Compiling and Testing

To facilitate local testing, we have provided a simple driver program. You are encouraged to utilize this for preliminary tests and feel free to write and use additional driver programs to further validate your functions. Compile your code in a Linux environment using the command:

```
g++ -Wall simpledriver.cpp p1.cpp -o p1
```

Start by testing your program with one provided simple test case (`1.in` and `1.out`). This gives you a baseline assurance that your program can handle basic scenarios. To run this test:

```
./p1 < pretest_cases/1.in > test.out
diff test.out pretest_cases/1.out
```

This sequence of commands executes your program, directing input from `1.in` and output to `test.out`, followed by using `diff` to compare your output against the expected `1.out`. If `diff` produces no output, it indicates that your program's output matches the expected output for the provided test case.

The test cases we have given you are not sufficient to catch all bugs, so testing with a diverse set of inputs is crucial, as the online judging system will evaluate your submission using a series of hidden test cases after the due date. These hidden test cases are designed to thoroughly assess the functionality and reliability of your palindrome checker under a wide array of conditions.

Submitting and Due Date

You should submit your source code via JOJ. Your submission should only include your source code file `p1.cpp`. The due date is 11:59 pm on Mar. 6th, 2024.

Grading

Your program will be graded along two criteria:

1) Functional Correctness

2) General Style

An example of Functional Correctness is whether or not you produce the correct output. General Style speaks to the cleanliness and readability of your code. We don't need you to follow any particular style, as long as your style is consistent and clear. Some typical style requirements include: 1) appropriate use of indenting and white space, 2) program appropriately split into subroutines, 3) variable and function names that reflect their use, and 4) informative comments at the head of each function.