Programming Assignment – Programming in C - Pointers

1. Create a C program file named ***pointers.cc*** that contains the methods from the steps below.
   - **DO NOT USE ANY CONSTANTS FROM MAIN IN YOUR FUNCTIONS.**
   - **YOU MUST USE POINTERS FOR THESE FUNCTIONS. YOU CANNOT USE ARRAYS OR [] ANYWHERE IN YOUR PROGRAM.**
   - **YOU MAY ONLY CREATE THE FUNCTIONS LISTED BELOW. NO HELPER FUNCTIONS ALLOWED.**
   - **YOU ARE ONLY ALLOWED TO USE STDIO.H.**
   - **DO NOT CALCULATE A LENGTH OF STRINGS ANYWHERE IN THESE FUNCTIONS. THE POINT IS FOR YOU TO LEARN HOW TO USE POINTERS.**
   - **RUNTIME ERRORS:** If your code has a runtime error with any of my tests you will receive a zero for the attempt. Make sure to thoroughly test your code before submitting. If you receive a zero, EVEN IF IT LOOKS LIKE YOU PASSED SOME TESTS, it is because a test after the ones you passed caused a runtime error and caused the whole testing system to crash. Since the program crashed, it means no output is generated which means the system cannot determine your grade and you receive a zero.
   - **FAILURE TO FOLLOW THESE RULES WILL RESULT IN FAILURE ON WEB-CAT. YOU HAVE BEEN WARNED.**

2. Create a function named ***countCharRange*** which has a string and two characters as parameters. The function should search the string and return the number of characters from the string are in the given range. The test will use the standard printable ASCII values. You SHOULD include c1 and c2 in the range. Note that c1 and c2 can be in any order .

```
int countCharRange (char *str, char c1, char c2) {
        int count = 0;
        while (*str) {
                //Do your work here.
                //Make sure to point str at the next character
        }
        return count;
}
```

   For example, if you call the function this way from main

   char str[] = "eabbmcddeeff";

   int y = countCharRange(str, 'e', 'z');   //How many characters in str and are >= 'e' and <='z'

   y should be 6 because the underlined characters are in the given range "eabbmcddeeff"

   note that countCharRange(str, 'z', 'e') should also return 6.

3. Create a function named ***findStringEnd*** which has a string (char pointer) as a parameter and returns a char pointer to the null terminating zero in the string.

   Here is what the function header should look like. The parameter names do not matter.

   ```
   char *findStringEnd(char *str)
   {
   }
   ```

   Note that this function will be used in place of a string length to terminate our loops in the following functions. YOU DON'T NEED LENGTH. YOU ONLY NEED THIS POINTER. DO NOT CALCULATE LENGTH.

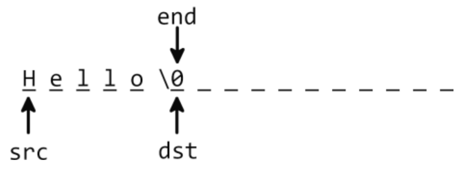
4. Create a function named ***doubleString*** which has a string as a parameter. The function MUST MODIFY the char array that was passed as an argument. You should concatenate the string passed onto itself.

   For example, if the string "Hello" were passed into the function, on return the string should contain "HelloHello".

   ```
   char str[20]="Hello";
   doubleString(str);
   printf("%s\n", str);  //Should print HelloHello and nothing else.
   ```

   Some hints:
   - You can assume the user will always provide a big enough array for the function to work.
   - You can use findStrEnd() to get a pointer to the end of the original string.
   - You can compare pointers like this:

     while (src < end) {copy character from src to dst then increment src and dst}



5. Create a method named ***palindrome*** which has a string as a parameter. The function MUST MODIFY the char array that was passed as an argument. You should concatenate the string passed onto itself with characters in reverse order to make a palindrome.

   For example, if the string "Hello" were passed into the function, on return the string should contain "HelloolleH".

   ```
   char str[20]="Hello";

   pallindrome(str);

   printf("%s\n", str);  //Should print HelloolleH and nothing else.
   ```

   Note that this program is almost identical to doubleString() with the only difference is the starting point and direction of the movement of the src pointer.

6. Use a main.cc file with a main method for testing your program before uploading. There is a main.cc listed below.

   MAKE YOUR OWN TESTS WITH DIFFERENT ARRAYS!!! Make sure to test with negative elements and zeros.

   DO NOT UPLOAD MAIN.CC OR ANY FILE WITH A MAIN METHOD IN IT!

   DO NOT PUT A MAIN FUNCTION IN pointers.cc (NOT EVEN FOR TESTING)

   Do it the correct way. If you don't understand **ASK**!!!

   The command for compiling is the same as last week but with pointers.cc.

   ```
   g++ -Werror -o pointers main.cc pointers.cc
   ```

   Run using ***pointers*** or ***./pointers***

   You will get an error message on test fail in main. If nothing is printed you passed all tests.

7. DO NOT PUT A MAIN METHOD INSIDE OF pointers.cc. A main method should ONLY be in main.cc. If you cannot figure out how to make your program compile without putting a main method in pointers.cc, come see me.

   YOUR TESTS WILL FAIL WHEN SUBMITTED TO WEB-CAT IF YOU PUT A MAIN FUNCTION INSIDE OF pointers.cc.

8. Submit your ***pointers.cc*** file to Web-CAT for grading.

   Do not submit a file with a main function.
   DO NOT submit main.cc.
   File cannot be submitted more than 2 days late.
   Late submissions will suffer a 10-point penalty per day.
   You will lose 1 point for each submission over 5.

THERE IS A MAIN.CC on AsULearn. I was going to add it to this document, but it is long, so I decided to put it on AsULearn instead.

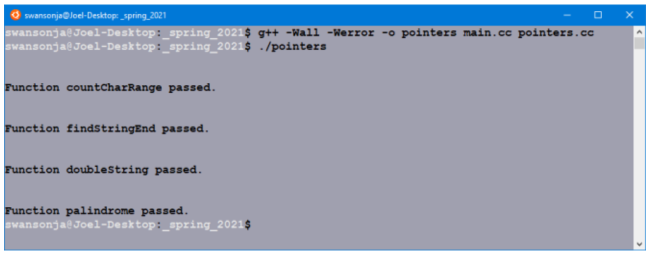
The following is the output of that main if you run it and have all your functions correct. You should teste with more than just the tests listed in that main.

Just passing these tests does not guarantee that you have everything done correctly. What happens on an empty string? Ask yourself what should happen on an empty string. You may want to answer the question "What is an empty string?" first though.

You should copy the tests that are in main and make new tests instead of just modifying the tests that are there.

If you have an error, then you change the test to check that error and then you "fix" that second problem, sometimes you break the first test when you fix the issue for the second test. The only way to be sure your changes didn't mess up something else is to keep running the old tests AND make new tests.

That is how real testing on real systems is conducted. You don't change one test over and over. You copy and paste the tests to make new tests then run ALL the tests every time you make a change.



```
swansonja@Joel-Desktop:_spring_2021$ g++ -Wall -Werror -o pointers main.cc pointers.cc
swansonja@Joel-Desktop:_spring_2021$ ./pointers


Function countCharRange passed.


Function findStringEnd passed.


Function doubleString passed.


Function palindrome passed.
swansonja@Joel-Desktop:_spring_2021$
```