Week 3

Thursday, January 30, 2020 11:31 AM

Week 3 - Spiral 2 Submission

Edits from Spiral 1 to Spiral 2 have been noted in red. These are solutions to better my answer for this Creativity Challenge after reviewing Professor Morrison's solution.

Student Name: Nolan Downey
Email: ndowney@nd.edu

Problem Statement: Modified from *Cracking the Coding Interview:* Write code which creates a std::string of unsorted characters, and write a function which remove duplicates from the string, and then prints the final string to the user. The order of the string must be maintained.

a) Identify Objectives: The stated problem provides a "real-world" or advanced concept situation. In this section, you will break the problem down from an abstract description to a computing problem.
   a. Write a function that takes in a string and that removes duplicates from a string
   b. Compare each character of the string to the other characters
   c. If a duplicate is found, the duplicate needs to be removed without altering the rest of the string
   d. If no duplicates are found, the output string should be the same as the original string
   e. The string must stay in order
   f. The final output needs to be printed to the user

b) Identify Risks and Alternatives: Describe tradeoffs for memory (spatial and temporal) and runtime complexity. If you know an STL library that may solve your problem, state that you will use it here. You must identify both the name of the data structure and the name in the C++ STL.
   a. Risk: Create a new string which takes in every character unless it is a duplicate
   b. Alternative: Takes up more memory and space than needed for this problem.
   c. Risk: We need to know whether the characters of the string are in Unicode or ASCII.
   d. Alternative: We can assume ASCII because the codes are run on the Notre Dame machines.
   e. Risk: We could index or iterate through the whole string every time we get to a certain character
   f. Alternative: This might compare the same value to itself, causing it to erase. There is also no need to compare characters with characters earlier in the string because they would already have been compared.
   g. Risk: The C++ STL "string" comes with a method called "erase" that can easily help with removing characters from an array.
   h. Alternative: I need to ensure that I am using the erase method correctly, only affecting the one character in mind.
   i. Risk: Sort the array and then remove similar characters.
   j. Alternative: The problem does not allow sorting
   k. Risk: The string could be passed by value and then returned
   l. Alternative: This takes up double the memory as passing by reference (de-referencing required)
   m. Alternative: A bool array (length 128) could be used in which all values are false then then duplicates could be erased while values could be set to true.

n. Risk: This approach has a bit of extra memory, but the problem does not restrict memory so this seems like the best approach.

c) Product Development and Testing: Describe your proposed solution, and show a test case indicating how it works.
   a. First, the strings in question will be defined in the main function
   b. Then these strings will be passed to a function that removes any duplicate characters from the string.
   c. The best way to test whether this function is acting correctly is to test it by passing one string with duplicates and one string without duplicates.
   d. My proposed solution for this function is one nested for-loop that contains an if statement and the string.erase() method mentioned earlier. Logically, this for loop will simply compare the current character with all the following characters in the string, erasing any duplicate characters.
   e. The following shows the logic for the function removeDupes, found in stringRem.cpp
      i. Take the word banana
         1) The first for loop starts at the letter b (variable i)
         2) The nested for loop starts at the letter a and iterates to the rest of the string (variable j)
            a) if (s.at(i) == s.at(j)
               i) s.erase(s.begin()+j);
         3) By this logic, after the letter b is checked, the letter a is checked, and the nested for-loop starts at the first letter n. Once it moves to the second letter a, the computing device knows to delete the character that is "j" values away from the beginning of the device
   f. Professor Morrisson's solution uses a while loop that only iterates when a character is not erased. This avoids many of the problems that arise when using a for loop, and avoids the issue of using two for loops.
   g. while itr < string.size
      i. Increment if erase is not called

d) Planning the Next Phase
   a. I believe that I correctly coded the function to remove duplicates in stringRem.cpp, and after testing it works like a charm. So I do not believe there is anything to be written in this step. However, I will comment that during my first attempt at writing the function, I was not fully aware of how the erase method of the C++ STL library "string" worked, and needed to use cplusplus.com to reference what needed to be passed to the method.
   b. Professor Morrison's solution avoids the use of iterating through the loop more than one time, but I believe my solution uses less memory than his. Although his solution runs in O(n) time and mine runs in O(n squared) time, I believe my solution is an acceptable choice.
   c. However, one important note was that I forgot to make my function void and not return. I passed by reference, but still returned the string (on accident) which is not what I needed to do. This was an important realization of a minor error on my part, and something I need to pay more attention to in the future.

```
ndowney@student12:~/DSInClass/Week3$ make
g++ -m64 -std=c++11 -Weffc++ -O0 -g -Wall -Wextra -Wconversion -Wshadow -pedanti
c -Werror -lm   -c -o /escnfs/home/ndowney/DSInClass/Week3/stringRem.o /escnfs/h
ome/ndowney/DSInClass/Week3/stringRem.cpp
g++ -m64 -std=c++11 -Weffc++ -O0 -g -Wall -Wextra -Wconversion -Wshadow -pedanti
c -Werror -lm -o ~/DSInClass/Week3/stringRem ~/DSInClass/Week3/stringRem.o
ndowney@student12:~/DSInClass/Week3$ ./stringRem
Before Removing Duplicates
adagahljASDj67SAl7
abcdABCD1234

After Removing Duplicates:
adghljASD67
abcdABCD1234
ndowney@student12:~/DSInClass/Week3$
```