Project Name: Project 3

Name: Marc Luzuriaga

UID: 205916585

**Notable Obstacles**

The biggest obstacle that I overcame in this project was checking the integers between the character values. In Particular, I had a difficult time with the index position and undefined behavior. Whenever I would call my check\_integers function that took in parameters of the string, starting index position, and termination index position, the loop within the check\_integers function would go out of bounds of the string. In order to solve this problem, I had to implement multiple if conditions to check if the str.size() didn’t go out of bounds in order to execute the check\_integers function.

The second biggest obstacle that I overcame was checking whether or not there were no leading zeros. Again, I found the index position and undefined behavior difficult in implementing this check because of the for loop . I was able to solve this by implementing if conditions to check if one of the character’s index positions went over the str.size().

Finally, the third biggest obstacle that I overcame was looping through multiple strings. At first, I thought that I could delete the string that my program checked in one iteration. However, I noticed that this approach to the problem was much more complex. Therefore, I decided to use the substring that would take the index positions of Q through a single loop and the distance between the first iteration of ‘Q’ and the next ‘Q.’ This gave me a substring that I was able to check for one iteration; and, for the next iteration, I would get the next string in the sequence of the characters to check.

**Description of the Program**

isValidQC

There were 5 requirements that were needed to pass a valid string in this program. The following are the 5 requirements followed by the steps the program needed to take in order to declare a string valid.

1. Requirement 1: Check if the index position of ‘Q’ is at 0
2. Requirement 2: Check if the number of ‘Q’s, ‘p’s, ‘d’s match
3. Requirement 3: Check if the integers between the characters are valid
4. Requirement 4: Check if ‘Q’ p’ ‘d’ have no leading zeros
5. Requirement 5: Check if the number of passes and defects matches the total tests

Step 1 (Preparing to test the requirements): The first step to determining whether the string was valid was counting the occurrence of ‘Q’, ‘p’, and ‘d’. The following was the structure for the count function that created used to count the characters:

Parameters passed in: 1) String to loop through 2) The character that needs to be found

For each index position in the string

If the corresponding character matches the character that needs to be found

q++

Return q

Step 2 (Preparing to test the requirements): Because a sub-string always starts with ‘Q’, I set the loop iterations to go from 1 to the number\_of\_times ‘Q’ is present in the string

Step 3 (Preparing to test the requirements): Next, I would find the index position of two ‘Q’ index positions. The following was the structure for the IndexPositionOfNthOccurrence that I used to find the Nth Occurrence of a character

Parameters passed in: 1) String to loop through 2) The character that needs to be found 3) The Nth occurrence

For each index position in the string

If the character equals the character that needs to be found

Counter += 1 in order to skip or find the N occurrence

If the counter equals the Nth Occurrence

Return the character

Return -1 in order to move back one position if not found

Step 4 (Preparing to test the requirements): Take the substring between the two index positions. This substring would enable the program to loop through multiple iterations of the same string to check if it is valid

Step 5 (Preparing to test the requirements): Find the index positions of the substring. The purpose of finding the index positions of the substrings is to use it for control flow and iterations at specific places in the sub-string.

Step 6 (Requirement 1): The first requirement was to check if the index position of ‘Q’ was in fact at 0. In order to do this, I simply placed an if statement with a condition that the ‘Q’ index position must equal 0; and, inside the block of this if statement, a boolean valid\_Q would be set to true.

Step 7 (Requirement 2): The second requirement in this program is to check the number of ‘Q’s, number of ‘p’s, and ‘d's that were present in this program. Again, I used an if statement checking the conditions, and, within the if statement block, a boolean named validcharacters would be set to true.

Step 8 (Requirement 3): The third requirement in this program is to check if the digits are valid between the letter characters in the string. In order to do this, I would call the function check\_integers. The following is a representation of how check\_integers works:

Parameters passed in: 1) String to loop through 2) Starting\_index\_position 3) Termination\_condition\_index

Difference\_in\_index = termination\_condition\_index - starting\_index\_position

Bool valid\_integers = false;

If the starting\_index\_position is less than or equal to size of string, move to next check

If difference in the index is less than or equal to one

Valid\_integers = false;

Else

For numbers between the termination condition to the starting index

If the character in the index is not a digit

Valid\_integers = false;

If the loop reaches the digit after the starting\_index\_position

Valid\_integers = true;

Else

Valid\_integers = false;

Return valid\_integers

Regarding the index position of Q, there were two possibilities, the character ‘p’ came after Q or ‘d’ came second. In order to avoid out of bounds errors, I only ran the check\_integers between ‘Q’ and ‘p’ if the difference in the index position of p and Q was greater than 1, or the index position of d and q was greater than 1.

Regarding the index position of p and d, there are two possibilities, ‘p’ goes first then ‘d’ or ‘d’ goes first then ‘d’. In order to orderly check the integers to the respective positions to the end of the string, I separated these two possibilities into their own two if statements.

Step 8 (Requirement 4): The next check in this program is checking if there were no leading zeros. I created an outside function called no\_leading\_zeroes that passed parameters of the string and the index position to start checking whether there were no leading zeros. The following is a description of how the following function works:

Parameters passed in: string s, size\_t index position

If the next index position == ‘0’

If there is no character after the first occurrence of 0, return true

Return true

Else if the index position after the first occurrence of 0 is a digit, return false

Return false

Return true

Return true

Step 9 (Requirement 5): The final requirement in this function is to check if the sum of passed tests and defect tests matches the total tests. In order to do this, I was able to use a for loop to concatenate the characters between the characters into a string. From there, I was about to use the stoi function to convert it into digits. Finally, after it was converted, I used an if statement to set a boolean either to true or false.

Step 10: Finally if all requirements passed, I would set a final boolean equal to true. Otherwise, it would be set to false. If the boolean was set to false, I would break from the loop.

passQC

The first step associated with passQC is to call the function isValidQC and check if the string passed in was actually valid. If it was valid, I would use the same technique from isValidQC to loop through the number of Qs present and set it to a substring. Finally, I would use the same technique from isValidQC to append the digits after ‘p’ to a new string and use the stoi function to return the total\_passed\_tests.

defectQC

The first step associated with defectQC is to call the function isValidQC and check if the string passed in was actually valid. If it was valid, I would use the same technique from isValidQC to loop through the number of Qs present and set it to a substring. Finally, I would use the same technique from isValidQC to append the digits after ‘d’ to a new string and use the stoi function to return the total defect tests.

totalQC

The first step associated with totalQC is to call the function isValidQC and check if the string passed in was actually valid. If it was valid, I would use the same technique from isValidQC to loop through the number of Qs present and set it to a substring. Finally, I would use the same technique from isValidQC to append the digits after ‘q’ to a new string and use the stoi function to return the total tests.

Batches

The first step in this function is to check if the string passed in was valid by calling the isValidQC function. If the string was valid, the function would count the number of ‘Q’ occurrences. Otherwise, the function would return -1

**Test Data**

1. assert(passQC("Q200p200d0Q201p200d1Q150d130p20Q150d130p20") == 440);
   1. I used this assert to test my program because I wanted to ensure that the function recognized the difference between switching the position of p and d. I also wanted to ensure that the program could handle long strings.
2. assert(defectQC("Q2p1d1") == 1);
   1. I used this assert in order to test if my program could handle simple one batch strings
3. assert(batches("Q200p200d0Q201p200d1Q150d130p20") == 3);
   1. I used this assert in order to ensure that batches could handle long strings of data
4. assert(batches("QQQQQQQQQQQ") == -1);
   1. I used this assert in order to ensure that batches marked a series of Qs false because I depended on counting the number of Qs to count the number of matches.
5. assert(defectQC("Q2p1d1 1") == -1);
   1. I used this assert to check that the program was taking into account spaces at the end of the string.
6. assert(batches("Q2p1d1 Q2p1d1 Q2p1d1 Q2p1d1 Q2p1d1") == -1);
   1. I wanted to ensure that isValidQC was checking spaces in between sets of batches.
7. assert(passQC("Q0p0d0") == 0);
   1. I used this assert to check whether ot not my program was not flagging 0 as invalid.