

UID: 705844713

**1.OBSTACLES FACED:**

A majority of the obstacles which I faced came from the isValidQC function, but after I figured out how to code that function, the other obstacles were much easier to overcome.

The first major obstacle I faced when tackling this project came about coding isValidQC to take multi-digit numbers ( for example Q as 147) instead of single-digit numbers. After trial an error with using another else if statement, I decided to use a for loop instead, which would define the integer, digitsInQ, which I would add on to whatever index of results[i]. This was then the same process I used for multi-digit p and d values.

Another obstacle I had was when I had to code the simplification of the Q value equaling p + d, mainly for multi digit numbers. Much of this project gave me deja-vu to coding in python, so my first instinct to tackle this obstacle was to use string slicing. By using string.substr(), and the stoi function, I could get the digit values for p and d for even multi digit values. ( Also, I used the extractNumbers function earlier in the code, which worked fine, but when I tried to use it here it didn’t work [yet another obstacle], but the stoi function was able to work instead.

The last large obstacle, and probably the biggest obstacle I had for this function, was when coding the loop portion of isValidQC. Originally, I tried to incorporate a for loop into my code, but after much time wasted, nothing worked. A while loop was attempted as well, but no progress was made there either. My last resort was to simply slice the code from the first batch, and recall the function onto a new substring which doesn’t include the first batch. After some tweaking and lots of tests, the isValidQC function finally was complete.

The only other function which caused me major difficulty was the passQC function. I first tried coding this using the same code that I used for the isValidQC function, which defined two variables of digitsInFirst( digits in either p or d, whichever came first) and digitsInSecond(p or d, whichever was second. This however brought up the problem if I did a test case where p came before d in batch 1, but after d in batch 2 ( Q2p1d1Q3d1p2). Instead, I had to use if else statements, containing the for statements which would define new variables, digitsInP and digitsInD. It's difficult to explain the rest of the code, but after figuring out how to do this, I was then able to finish the passQC function, which is almost identical to the defectQC function.

**2.Program Design**

1. First section to initialize and declare my function Header
   1. isValidQC,passQC, defectQC,totalQC,batches, extractNumber
2. The int main() contains assert test cases, separated by each function
3. FUNCTION 1: bool isValidQC ( string results)
   1. if (string empty) ~ return false
   2. else if (string doesn’t begin w/ Q) ~ return false
   3. else if(# following Q is <= 0) ~ return false
   4. for( int i; results[i] == # 0 - 9; ++i)
      1. digitsInQ += 1 ( calculates the digits of a multi digit number, besides first, in Q)
   5. if( p or d don’t follow the test digits) ~ return false
   6. else if( the number following p or d < 0) ~ return false
   7. for(# following p or d is <= 0)
      1. digitsInFirst += 1(calculates the digits of a multi digit number, besides first, in the first p or d to occur)
   8. if(p or d don’t follow first digits) ~ return false
   9. else if ( there are leading 0’s in Q) ~ return false
   10. Define numQ, num1, and num2 ( represents # of batches, pass, and defects)
   11. If (Total tests isn’t equal to pass + defects) ~ return false
   12. if(test string followed by random characters other than new batch) ~ return false
   13. else if( string has only pass values) ~ return false
   14. else if( string has only defect values) ~ return false
   15. else if(there is another batch)~ return isValidQC(string of new batches)
   16. else ~ return true
4. FUNCTION 2: int passQC(string results)
   1. if (isValidQC(results) is true)
      1. Use for loops to redefine digitsInQ, digitsInFirst, and digitsInSecond, same as in ValidQC
      2. If ( p comes before d)
         1. for(i=4; results[i] is = 1-9; i++) ~ digitsInP += 1 (New variable defining the digits in P (-1 digit)
      3. else
         1. for(i=4; results[i] is = 1-9; i++) ~ digitsInD += 1 (New Variable defining the digits in D (-1 digit)
      4. If ( p comes after d)
         1. for(i=6; results[i] is = 1-9; i++)~ digitsInP += 1
      5. else
         1. for(i=6; results[i] is = 1-9; i++)~ digitsInD +=1
      6. howManyP = stoi(results.subst()) (Total Pass if one batch)
      7. if(there is another batch) ~ return howManyP + passQC(string of new batches)
      8. return howManyP
   2. else ~ return -1
5. FUNCTION 3: int defectQC(string results)
   1. if (isValidQC(results) is true)
      1. Use for loops to redefine digitsInQ, digitsInFirst, and digitsInSecond, same as in ValidQC
      2. If ( d comes before p)
         1. for(i=4; results[i] is = 1-9; i++) ~ digitsInD += 1 (New variable defining the digits in D (-1 digit)
      3. else
         1. for(i=4; results[i] is = 1-9; i++) ~ digitsInP += 1 (New Variable defining the digits in P (-1 digit)
      4. If ( d comes after p)
         1. for(i=6; results[i] is = 1-9; i++) ~ digitsInD += 1
      5. else
         1. for(i=6; results[i] is = 1-9; i++) ~digitsInP +=1
      6. howManyD = stoi((results.subst()) (Total Defect if one batch))
      7. if(there is another batch) - return howManyD + passQC(string of new batches)
      8. return howManyD
   2. else ~ return -1
6. FUNCTION 4: int totalQC(string results)
   1. if(isValidQC(results) is true)
      1. Use for loops to redefine digitsInQ, digitsInFirst, and digitsInSecond, same as in ValidQC
      2. howManyQ = stoi(results.substr()) ( Total tests if only 1 batch)
      3. If (there is another batch) ~ return howManyQ + totalQC(string of new batches)
      4. return howManyQ
   2. else ~ return -1
7. FUNCTION 5: int batches(string results)
   1. if(isValidQC(results) is true)
      1. for( i =0; i < string length; ++i)
         1. if(iterating over ‘Q’) ~ batchNumber += 1
      2. return batchNumber
   2. else ~ return -1

**TEST DATA:**

| **assert(isValidQC("") == false);**   * Testing the empty case   **assert(isValidQC(" ") == false);**   * Testing white space   **assert(isValidQC("Q0") == false);**   * Testing when Batches is zero   **assert(isValidQC("R1") == false);**   * Wrong character   **assert(isValidQC("Q1p1d0") == true);**   * True case   **assert(isValidQC("Q1f1d0") == false);**   * Incorrect character for p   **assert(isValidQC("Q1d1p0") == true);**   * True case   **assert(isValidQC("Q2p1d1") == true);**   * True case   **assert(isValidQC("Q10p5d5") == true);**   * True case   **assert(isValidQC("Q2p0d2") == true);**   * True case   **assert(isValidQC("Q2pd") == false);**   * No value for p & d   **assert(isValidQC("Q22p22d0") == true);**   * Multi digit value   **assert(isValidQC("Q2p0d0") == false);**   * Pass and Defect not equal to tests   **assert(isValidQC("Q2p00001d0001") == false);**   * Leading zeros   **assert(isValidQC("Q1p0d1 akswld") == false);**   * Extra characters after batch   **assert(isValidQC("Q1p0p1") == false);**   * Only pass values, no defects   **assert(isValidQC("Q1d1d0") == false);**   * Only defect values, no pass   **assert(isValidQC("Q1p1d0Q1p0d1") == true);**   * Two batches   **assert(isValidQC("Q20p2d18Q1d1p0Q12p1d11") == true);**   * Two batches w/ multi-digit numbers   **assert(isValidQC("Q1000p500d500Q1d0p1") == true);**   * Big boi numbers | **assert(isValidQC("Q100p20d80 ") == false);**   * White space   **assert(isValidQC("Q26p13d13Q12d5p1") == false);**   * D and P don’t add to equal tests   **assert(isValidQC("Q100p50d50Q10d5p5") == true);**   * True case   –--------------------------------------------------------  **assert(totalQC("Q2p1d1") == 2);**   * Normal case   **assert(totalQC("Q100p50d50") == 100);**   * Multi Digit Number   **assert(totalQC("Q2p1d1Q3p2d1") == 5);**   * Two batches   **assert(totalQC("Q23p21d2Q20p0d0") == -1);**   * Parameter not valid   **–----------------------------------------------------------**  **assert(batches("Q2p1d1") == 1);**   * Normal case   **assert(batches("Q1p1d0Q100p50d50") == 2);**   * Multi Digit, Two Batches   **assert(batches("Q1p0d1 wldm") == -1);**   * Parameter not Valid   **–----------------------------------------------------------**  **assert(passQC("Q2p2d0") == 2);**   * Normal Case   **assert(passQC("Q22p20d2Q10p5d5") == 25);**   * Multi Digit, Two Batches   **assert(passQC("Q2d1p1Q7d2p5") == 6);**   * Multi Digit   **–----------------------------------------------------------**  **assert(defectQC("Q2p0d2") == 2);**   * Normal case   **assert(defectQC("Q3p0d3Q12d4p8") == 7);**   * Multi Digit, Two Batches   **assert(defectQC("Q2p0d2Q4p1d3") == 5);**   * Multi Digit |
| --- | --- |