Atij Mahesh

Professor Smallberg

Computer Science 31

26 October 2022

Project 3 Report

Project 3 presented some challenges, as I had to decide the most effective and working way to parse through the input string. I also had some challenges regarding tracking down the correct index and doing all proper checks to ensure I was not checking an out-of-bounds index on a string. Overall, the most tedious work and challenges of the project came from catching as many errors or strange yet valid cases as possible.

For the “isValidPollString” method, I utilized delimiters and parsed through the string through the indexes of commas. I also used recursion to loop through the program and shorten the input string, which would only focus on the “first” forecast on hand. I also catch strange cases, such as when there is nothing after the state code, or when the poll data starts or ends with a comma.

| *Find first index of comma If no comma, the forecast length is the length of the string. If comma, the forecast length is the length before the comma. If the length is zero, the method is finished and is true*  *If the input starts or ends with a comma, return false Make input string completely uppercase  If state forecast starts with a valid uppercase state code then  Parse through the state forecast  If the first number is a digit then  If the second is a letter, then it's good so far, continue through forecast  Second number a digit, then check if third is a letter, if it continue through   forecast  If none of the above cases happen, it cannot be valid  If no more commas and it's good so far, the poll string is valid Repeat but start from where the last comma ended* |
| --- |

For the “countSeats” method, the method first checks if the input results in a case where the return value is 1 or 2, then set seat count to zero, and finally utilizes the recursive method to comb through the poll data string.

| *return 1 if poll data is not a valid poll data string if the party isn't a letter, return 2 set seatCount to 0 return recursive helper method which looks at poll data string and updates seatCount* |
| --- |

For the “recurCountSeats” method, I looked to separate out the forecasts by delimiting through the commas and analyzing each forecast independently. I checked if the forecast had an instance of the party, and also double-checked if there is potential room for more of the same party reports in a single forecast. Upon checking, if there was, I would add it to the sum, but if not, I’d skip to the next one using recursion.

| *if it isn't a valid poll string, return 1 if the party isn't an alphabet, return 2 if the seat count is negative, make it zero if the length of data is zero, its finished parsing make party char and input string uppercase  find nearest comma position  if the first forecast is just the state code, skip to next forecast    find the first instance of the party letter  if the state code has the same letter, skip to next instance  if there are no instances of the party, function done    if there's no party in current forecast, skip to next one  else go through forecast  check if number is 2 digits long  add double digit number to seat sum  if not, add single digit number to seat sum  if there's potential for more of the same party to be in the same forecast  look at it again minus the seats we just looked at  if there are more forecasts  look at the poll again, ignoring the forecast we just looked at* |
| --- |

My main method is relatively simple, as it only directly counts on countSeats to analyze the user’s input. countSeats then utilize the 2 other methods to check if the input is a valid poll string and recursively parse through it. I also catch an error where a user does not input a single character for the party name. My program catches many errors, as it only focuses on one state forecast at a given instance.

Test Data Cases:

**assert(isValidPollString(“Ct5D,NJ16D4I,nH,ne3r00D”))**

Useful for checking for Uppercase, lowercase, and mixed state codes.

Useful for checking for forecasts with multiple parties, with single and double-digit inputs.

Useful for checking for forecasts with no party results

**assert(isValidPollString(“,,,”))**

Useful for testing cases of multiple empty forecasts, which is not valid.

**assert(isValidPollString(“”))**

Useful for testing cases of singular empty forecasts, which are valid.

**assert(isValidPollString(“C4”))**

Useful for testing cases of no state codes or invalid poll string.

**assert(isValidPollString(“,MI16D, , ne3r&\*D”))**

Useful for testing cases when there are spaces or invalid characters in poll string.

**assert(isValidPollString(“NY9R16D1I,Vt,NJ3d5r4D,KS4R”))**

Useful for checking when there are multiple of the same party inputs in a single forecast.

**seats = -999;**

**assert(countSeats("CT5D,NY9R16D1I,VT,ne3r00D", 'd', seats) == 0 && seats == 21);**

Useful when seats is a weird number at first and for checking if countSeats works.

**seats = -999;**

**assert(countSeats("CT5D,NY9R16D1I,VT,ne3r00D", '%', seats) == 2 && seats == -999);**

Useful when input party isn’t valid in countSeats and to check if seats is unchanged.

**assert(countSeats("NY9R16D1I,Vt,NJ3d5r4D,KS4R", 'y', seats) == 0);**

Useful as character y is in the state code, which the method sees and then ignores.