Madeleine Murphy

004991129

10/29/18

Project 3 Report

1. The main obstacles I overcame in writing this project were just figuring out exactly how I wanted to go about going through the string and making sure that it was in the proper format. I had to decide that I wanted to do many different bools and other functions in a certain order that would check for certain aspects of the string, and then from there call those functions in my main bool hasProperSyntax. I also had an issue with figuring out how exactly to separate the string up. I eventually figured out that I should separate them by commas into what I called “stateNugs” and then from there separate them further into party results called “partyChunks.” From there, I had to realize that party chunks can only be 2 or 3 in size, and once I figured that out it became easier to determine how to parse out the right and wrong formats. After that, the only obstacles I really had to deal with were minor bug fixes, which I did by going through at suspected places where the code could have been failing for certain inputs and cout-ing the pollData or stateNug or partyChunk or just the word “failing” at that time to see if they had been separated out correctly.
2. Essentially my code is a series of functions that use other functions to check that the inputted string is in the right format. From there, those functions are called to in the main hasProperSyntax function. The first function checks that the code has no spaces.

Pseudocode:

*bool has spaces*

*repeatedly:*

*check for spaces*

*if has spaces, return true*

*return false*

*bool validateStateNug*

*grab first two letters and add to string stateCode*

*send to upper case*

*check that they are valid uppercase state code*

*if only 2 digits, return valid State Nug*

*else take rest of substring*

*repeatedly:*

*break party chunks into party results (e.g. 14D)*

*add them to empty string partyChunks*

*send to bool checkPartyChunks*

*return true or false*

*if true, make string empty again and move on to next party result*

*return true*

*bool checkPartyChunks*

*if pc size > 3, return false*

*if pc size == 2, must be digit followed by a letter (1R)*

*return true or false*

*if pc size == 3, must be two digits followed by letter (14R)*

*return true or false*

*return false*

*void stateCodeLetters*

*repeatedly*

*take first two letters in string stateCode sent from validStateNug*

*make upper case*

*bool isValidUpperCaseStateCode*

*take string stateCodeLetters*

*check if they are a valid upper case state code*

*return true or false*

*bool hasProperSyntax*

*if empty string case*

*return true*

*if hasSpaces is true*

*return false*

*set stateNug to empty string*

*repeatedly*

*read through string pollData as long as char is not a comma*

*adds characters before comma to stateNug string*

*if it reaches last char in string,*

*sends to validateStateNug, checks that string is valid*

*if valid, reset stateNug string to empty*

*else return false*

*break*

*reaches a comma*

*sends to validateStateNug, checks that string is valid*

*if valid, reset stateNug string to empty*

*else return false*

*if pollData starts or ends in comma*

*return false*

*else return true*

*return false*

*int tallySeats*

*sets origiSeatTally = seatTally // if empty string it returns original*

*//tally*

*sets seatTally = 0*

*if fails hasProperSyntax*

*return original seat tally*

*return 1*

*if party is not a letter*

*return 2*

*repeatedly*

*if upper case inputted party = upper case value of some k in string*

*if k-1 is not a letter*

*break*

*else if k-2 is also a digit*

*add 10\*k-2 –‘0’ (to take care of ASCII) to seatTally*

*add k-1- ‘0’ to seatTally*

*if just k-1 is a digit*

*add k-1 – ‘0’ to seatTally*

*return 0*

1. Data Tests (all work):

* “NY3rd,” party d 🡪 tests ends in comma function
* “” 🡪 tests empty string & that tallySeats is set to 0 in this case
* “Ny 4D” party d 🡪 checks what happens when there is a space
* “ny4d” party d🡪 checks string that is correct but with lower case values
* “NY4D” party d 🡪 checks with all upper case
* “Ny4D” party d 🡪 checks with mixed upper and lower case
* “PP5r” party r 🡪 checks invalid state code, different party input
* “Ca44D” party d 🡪 checks different state and 2 digit value
* “Ca44D56r8i” party r 🡪 checks multiple values, different letters, different party input
* "Ca44D56r8i,NY78d56r8i,OK3d2r90i" party i 🡪 checks with multiple state forecasts separated by commas
* "Ca44D56r8i,NY78d56r8i,OK3d2r90i" party r 🡪 same thing but with different party input
* “Ca44D56r8i,NY78d56r8i,OK3d2r90i" party 8 🡪 checks but with invalid party input
* "Ca44D56r8i,,NY78d56r8i,OK3d2r90i" party r 🡪 checks with “,,”
* “PA444D8r2i” 🡪 checks when 3 digits precede a party code
* “PAD48r9i” 🡪 checks when no digit precedes a party code
* “Pa4d5r8i,nyd4ri” 🡪 checks when no digit precedes but in second string, essentially testing that issues in second string are also read as false strings
* “Ca44D56r8i,VT,NY78d56r8i,OK3d2r90i” party i🡪 checks when there is a state with no predictions