# Obstacles:

One of the obstacles I had to overcome while writing this program was creating the state string to send to isValidUppercaseStateCode() because I originally tried just declaring and initializing the state string as

string state = toupper(pollData[i]) + toupper(pollData[i+1]);

However, this didn’t work and the compiler said no suitable constructor existed. I realized this was due to the characters being added together as integers and returning an integer value to the string constructor. Fixing this problem was two part—I had to first concatenate the characters returned from toupper() to an empty string as such:

string state = string() + toupper(pollData[i]) + toupper(pollData[i+1]);

The second part was that this still didn’t work—the compiler now said the “+” operator didn’t match string + int so I had to cast the result from toupper back to a character type, so my final working code looked like this:

string state = string() + char(toupper(pollData[i])) + char(toupper(pollData[i+1]));

Another minor obstacle I overcame was the choice of loops – for vs while. I ran into this obstacle because I was incrementing variables I had already declared while iterating through the loops, so I did not want to declare a new variable in a for loop. However, I was still ending each iteration with a simple increment counter, which made me want to use a for loop instead of adding a line to increment the counter. The solution I found was that I could skip the initialization statement in the for loop by simply writing nothing there but keeping the semicolon e.g.

for(; [condition]; [update]) {}

# Description/Pseudocode:

The hasProperSyntax() function works by iterating through given pollData string state forecast by state forecast. Within each state forecast, the function checks if the first two characters are a valid state code by taking them, capitalizing them, and then sending them to the isValidUppercaseStateCode() function. If it is valid, the code continues; otherwise, it returns that the string is invalid. The code then moves past the state code and checks for either a party code or a comma. If neither exist, the program returns that the string is invalid. If a comma is found, it goes back to the beginning of the state forecast loop and looks for a new state forecast. If a party code is found, it checks if the party code is valid. If it is, it continues looking for either party codes or a comma, which designates a new state forecast. If the party code is found to be invalid, the function returns that the string is invalid. If the function successfully gets through the entire string without finding any errors, it returns that the string is valid. It works as such:

*repeatedly through each state forecast:*

*if first two letters of forecast are not a state code*

*return that pollData is not valid*

*repeatedly until next item is not a party code*

*if the party code is invalid*

*return that pollData is not valid*

*go to next item*

*if there is not a comma and not at the end of the string*

*return that pollData is not valid*

*return that pollData is valid*

The tallySeats() function works by first making sure each argument is legitimate by passing the pollData string to hasProperSyntax and making sure the party argument is a letter. If the string is not valid, the function returns 1 as specified and if the party code is not valid, it returns 2 as specified. It then cleans up the arguments by capitalizing all letters to make them easier to compare, then looks for party results and adds their votes together if they’re for the correct party. It then change the value of the argument seatTally to the calculated result and returns 0. It works as such:

*if poll data string is invalid*

*return that poll data string is invalid*

*if party character is invalid*

*return that party character is invalid*

*clean up strings and characters*

*repeatedly through each state forecast:*

*skip state code*

*repeatedly through each result:*

*if parties match*

*add the party result’s number of seats to seatTally*

# Test data:

|  |  |  |
| --- | --- | --- |
| hasProperSyntax() test cases | | |
| Case | Reason | pollData |
| Empty string | To make sure the function sees an empty string as valid | “” |
| No party results | To verify that the program views state forecasts with no party results as still valid | “VT” |
| Invalid symbols | To test if the program handles invalid characters correctly | “cA#” |
| Two character “party” | To check that the program correctly identifies invalid party results | “iL41de2re” |
| Party result with over 2 digits | To make sure the program responds correctly to party results with more than a two digit seat count | “CA432d2r” |
| Multiple state forecasts | To make sure the program handles multiple state forecasts correctly | “CA43d,IL3d” |
| Multiple party results per forecast | To test if the program handles multiple party results per forecast correctly | “CA43d2r” |
| Generic long string | To test if the program correctly handles all cases simultaneously (multiple state forecasts, multiple party results per forecast, and state forecast with no party results). | “CA432d2r,IL3d2r1i,VT,MN” |
| Generic long string | To test if the program correctly handles all cases simultaneously (multiple state forecasts, and multiple party results per forecast with hidden invalid character). | “CA432d2r,IL3d2r1#,VT,MN” |
| Generic long string | To test if the program correctly handles all cases simultaneously (multiple state forecasts, multiple party results per forecast with hidden party result 3 characters long). | “CA432d2r,IL3d2r221g,VT,MN” |

|  |  |  |  |
| --- | --- | --- | --- |
| tallySeats() test cases | | | |
| Case | Reason | pollData | party |
| Bad pollData string | To make sure the function returns the correct value when an invalid pollData string is given | “3F1F3” | ‘g’ |
| Empty string | To verify that the program handles an empty pollData string correctly | “” | ‘D’ |
| Bad party character | To test if the program handles bad party characters correctly | “cA3d” | ‘3’ |
| Simple valid case | To check that the program correctly handles simple cases | “TX38R2G” | ‘R’ |
| Scrambled upper and lower case | To make sure the program responds correctly to random upper and lower case characters | “tX38r2g” | ‘g’ |
| Multiple state forecasts | To make sure the program handles multiple state forecasts correctly | “CA43d,TX2g” | ‘G’ |
| Multiple party results per forecast | To test if the program handles multiple party results per forecast correctly | “CA43d2r” | ‘D’ |
| Generic long string with uppercase party | To test if the program correctly accumulates seats given a long but valid pollData string and uppercase party code. | “ID4R11G2D,RI4D,ND3R1D,CT,OR7D1R” | ‘R’ |
| Generic long string with lowercase party | To test if the program correctly accumulates seats given a long but valid pollData string and lowercase party code. | “ND3R,CT7D,OR7D,ID4R,SD3R,RI4D” | ‘d’ |
| Generic long string with scrambled upper/lower case | To test if the program correctly accumulates seats given a long but valid pollData string with scrambled uppercase and lowercase | “Nd3R,ct7D2g,oR7D,id4r,SD1g3R,Ri4D  ,Ca43d2r,IL3d2r21g,vT,MN,sD3R,rI4d” | ‘g’ |

\*In all expected failure cases for tallySeats() (tallySeats() was expected to return either 1 or 2), seatTally was checked to make sure it did not change.