## Obstacles Overcame

One of the obstacles I overcame was combining characters to form strings. Originally I wrote

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

this did not work because, on the left side of the equation is a string, but on the right are integers — the ascii code of the respective characters. However, I know for a fact *that characters can be added to a string using ‘+’ sign, hence I fixed the code by changing it to:*

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

First of all, I used char() to convert the ascii codes to characters. Secondly, I added the two characters to the end of an empty string, so that they are converted to string types and the value could be given to the string state.

Another obstacles I overcame was how to skip the state codes. in isValidPollString function I wanted to skip the state code after checking it already in checkState function. So I worte *i+=2*, thinking that the two state code characters would be skipped. But in execution, this ran into some erroneous results. After debugging, I realized that i++ executes at the beginning of each loop, hence i+=2 would in fact skip 3 characters and miss the immediate character follows the state code. After changing i+=2 to i++, this problem was solved.

## Pseudocode and Description

**isValidPollString function**

*scan through each character in the string*

*if the string includes spaces*

*return that pollData is invalid*

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

*return that pollData is invalid*

*else, meaning checking after the state code*

*if there are characters other than digits, letters and comma*

*return that pollData is invalid*

*if a letter is not immediately followed by a digit*

*return that pollData is invalid*

*if the state forecast does not end with a letter*

*return that pollData is invalid*

*check if a new state forecast starts by checking comma*

*return that pollData is valid*

Additionally, I introduced the variable pos to indicate if the loop is scanning the start of a new state forecast string. It was set to 0 at the start of the loop, and sets back to 0 every time a comma is scanned.

**coutSeats function**

*if pollData is invalid*

*return 1*

*if party letter is invalid*

*return 2*

*make sure the counter is reset to 0*

*scan through each character in the string*

*if counting a new state forecast, skip the state code*

*if the party letter matches*

*search backwards for the digits before*

*convert the digits to compute the party seats*

*add the converted number to the counter*

*return 0*

When searching backwards from party code, the first digit encountered should be multiplied by 1, the second by 10 and the third by 100 due to the numerical positions they occupy. The technique I used in this conversion is to introduce cnt variable to keep track of that. cnt is initialized to 1; each time a new digit is scanned, cnt multiplies itself by 10, until all the digits has been scanned and converted.

## Test Data List

| Reason | pollData | party | count |
| --- | --- | --- | --- |
| given example | NY9R16D1I,Vt,NJ3d5r4D,KS4R |  |  |
| wrong state code | AB55T,Ct9r7d |  |  |
| space included | KS4R, NV3D1R |  |  |
| invalid characters like + | ct5d,ny9r17+1i |  |  |
| two party characters | ca16de7re |  |  |
| does not end with a letter | tn6d2r5,ct5d4 |  |  |
| space in state code | L 45R |  |  |
| given example | NY9R16D1I,Vt,NJ3d5r4D,KS4R | d | 0 |
| invalid polldata | KS4R, NV3D1R | d | 0 |
| invalid party character | ct5d,ny9r17d1i | + | 0 |
| both invalid polldata and party | ct5d,ny9r17+1i | + | 0 |
| long digits | CA5678D2234R | D | 0 |
| inquires party without data | KS4R, NV3D1R | I | 0 |
| counter initialized as -888 | ct5d,ny9r17d1i | d | -888 |