**Obstacles Overcome:**

Managing the index so that it would never go out of bounds of the string was challenging, especially as the string could be anything.

Making sure the code was clean and keeping track of all of the code during debugging was also a challenge.

**Design of Program:**

process state prediction:

while current char is integer

if more than two chars consecutively, return false

store integers

if next two chars are a valid state code, return false

hasRightSyntax:

for every state prediction

if first char isn’t a letter, return false

run process state prediction

if returns false, return false

return true if process state prediction never returns false

computeVotes:

for every state prediction

if hasRightStynax returns false, return 1

store first char as current party

run process state prediction, store int as numVotes

if numVotes is 0, return 2

if current party is not letter return 3

if current party is same as party, add numVotes to total number of votes

set voteCount equal to total number of votes

return 0 if no other error occurs

**List of Test Data:**

First char is imputed party, rest of string is pollData

voteCount always starts at 20.

"RR40TXD54CAr6Msd28nYL06UT",

"DR40TXD54CAr6Msd28nYL06UT",

"rR40TXD54CAr6Msd28nYL06UT",

"lR40TXD54CAr6Msd28nYL06UT",

Test basic functionality with different party codes

"RR",

Test hasRightSyntax with invalid string with only 1 char

"RR40TXD0CA",

Test error when numVotes is 0, also test if voteCount is set back to original

"RR40 TX",

Test has RightSyntax with space

"2R40TX",

Test if non letter party returns 2

"RR R",

Test has RightSyntax again

"SR40CA D2CA",

Test has RightSyntax with space

"KM20ILD20CA",

Test code with party not in string

"R"

Test code with empty string