**Notable obstacles in development**

1. My first approach to *hasCorrectSyntax* was implementing a switch case to control logic flow for different character types in the string, as suggested by Professor Smallberg. However, since the switch case only works for integers, dealing with letters as in state and party codes would have required a lot of complexity.
2. I then decided to break up the program into multiple functions, each testing a smaller part of the string. For example, the *processStateForecast* function would call *processDigits*, which would call *processPartyCode*, and so on. This would have required a global variable to keep track of the location in the string. Developing the functions was tedious as well because I had to remember which part of the string each one analyzed.
3. My final approach was to develop *hasCorrectSyntax* as a single function and split it up into smaller pieces later. This version ended up being so efficient that I kept it mostly whole, only outsourcing the *makeUppercase* and *validState* sub-functions. However, when modifying *hasCorrectSyntax* into *tallySeats*, I kept getting compile errors such as “Signal SIGABRT” and “EXC\_BAD\_ACCESS.” After researching the problem, I discovered that the cause of the errors was that both *hasCorrectSyntax* and *tallySeats* had the same parameter name for the poll data string. Since *tallySeats* called *hasCorrectSyntax*, the parameter name would be overloaded, which caused the errors. I solved this issue by renaming the poll data string parameter in *tallySeats* from *pollData* to *pollData1*.
4. While testing *hasCorrectSyntax*, I found that it would return true for a party forecast with no digits preceding it, such as “VT9DR.” I addressed this by adding a condition that the character following the party code must be a digit.
5. *tallySeats* incremented *seatTally* by more than the number of votes contained in the string. I had removed all of the tests for validity from *hasCorrectSyntax* when making *tallySeats*. However, I discovered that I had also removed the segment that exited the loop when the end of the poll string was reached. As a result, *tallySeats* would keep running and add invalid numbers to *tallySeats*. I fixed this by reinstating the test for the end of the string from *hasCorrectSyntax*.

**Program design**

1. hasProperSyntax

This function analyzes poll data strings by performing a series of operations on each state forecast, using the comma character as a delimiter between state forecasts.

if empty string,

return true

convert all characters in string to uppercase

repeatedly:

if valid state code,

advance two characters

repeatedly while current character is a digit:

if next character is end of string or comma,

return false

else,

advance to next character

if end of poll string reached,

return true

if invalid party code after digits or more than 2 digits,

return false

advance one character

if end of poll string reached,

return true

if character after party code is not a digit,

break

else,

return false

return false if conditions have not been met

1. tallySeats

This function stores the digits of each party result in a temporary string. If the following character is the given party code, it adds this quantity to seatTally.

check if poll string and party character are properly formatted

repeatedly:

append digits to sum string

if next character matches party code,

convert sum string to integer

increment seatTally by number of votes

if end of poll string reached,

break

1. isValidUppercaseStateCode

returns whether input string is a valid uppercase state code

**Test cases**

The program passes all of the below test cases.

hasProperSyntax

* “”
  + empty string
* “C”
  + single character in string
* “KS4R, CA3D”
  + space between state forecasts
* “VTCA”
  + two states not separated by commas
* “KS 4R,CA3D”
  + space between state code and party result
* “ca3R5d6,vt3d6R,nj”
  + missing party code at end of state forecast
* “VT,CA”
  + state forecast with no party results
* “CA3r,”
  + comma at end of string
* “FL3r5d!2i,vt7r8d”
  + invalid character after party code
* “CA15dr”
  + party forecast with no digits
* “DQ7r6d,CA8r3i”
  + invalid state code
* “3r6d2i”
  + missing state code
* “CT5D,NY9R17D1I,VT,ne3D”
  + multiple states and parties
* “CA3D00r,VT8d6r00i”
  + 0 votes in party result
* “CA7d3R,,VT6r3i”
  + multiple commas between state forecasts
* “CA327D25r,NY27d”
  + more than 2 digits in state forecast
* “KS CA”
  + space between state forecast without comma
* “KS!CA”
  + invalid character between state forecasts

tallySeats

The program passes all of the below test cases.

For all cases: seats = -999 (check that value is changed)

* “”, ‘c’
  + empty poll data string
* “NY9R17D1I,VT3d,NJ3D5R4D, KS4R”, ‘d’
  + improperly formatted poll string
* “CT5D,NY9R17D1I,VT3d,ne3r7D”, ‘%’
  + invalid party code
* “VA7R3d,KS4R2i,VA4D”, ‘d’
  + more than one forecast for a state
* “VT,CA”
  + no party results in state forecast
* “NY9R17D1I,VT,NJ3D5R4D,KS4R”, ‘r’
  + party code appears in multiple state forecasts
* “CA3d4r,vt2d10r3i,tx9r”, ‘i’
  + party code only appears in one state forecast
* “CT0D,NY9R00D1I,VT,ne3r00D”, ‘d’
  + 0 votes in party result
* “VT999D3r,CA23d2i”, ‘d’
  + more than 2 digits in party forecast