CS 31 Project 3 Report

**A) Brief Description of Notable Obstacles**

In my opinion this project was substantially harder than the previous projects, and consequently there were a few major obstacles that I encountered. The first main obstacle was my initial visualization of how I wanted to organize my program and the necessary functions I would have to write. It took a decent amount of time in the beginning to figure out how I was going to structure my program, but once I had a rough outline of what to do it was easy to start writing some of the basic functionality.

After my program had some basic functionality, another problem was ensuring that it would handle all of the possible test cases. Specifically, one problem was in converting two digit strings into their respective integer values. Converting a one digit character into an integer was fairly simple, but converting multiple digit characters required some careful thought and programming. Another major problem was also in writing a proper for loop that would increment through the poll data string, finding the respective electoral votes for each state.

Finally, this was not a major obstacle per say but there were multiple small errors that I found through extensive testing that I had to make sure to account for. For example, if an electoral vote was entered as ’07’ the program would still return 2 since the string ’07’ contained the char 0. I fixed this by writing an if function that compared the int value of the string to 0 as opposed to simply looking for any 0 chars. Another small error was that if the party code was entered as D, but the state forecast said ’ca55d’, the program would not add 55 to voteCount. I fixed this by writing an if function that said if the party in the state forecast was equal to either uppercase party code or lowercase party code to increment voteCount. Ultimately I believe I was able to find all these small errors, and that my program should reasonably handle all the test cases.

**B) Description of Program Design**

*Overview*

On a high level, my program has two parts: one part which checks the validity of the entered pollData, and one part which processes the pollData based on the entered party code, and determines the value of voteCount for this party. Four boolean functions accomplish the first part, and two integer functions and one boolean function accomplish the second part. The program itself works by calling the countVotes function.

*countVotes*

countVotes can be thought of having four parts, which correspond to its four possible outputs. The first part is when it checks the validity of the entered pollData string, the second part is when it checks if any electoral votes are zero, the third part is when it checks the validity of the entered part code, and the fourth part is when it increments voteCount. Here is the countVotes pseudocode:

*if pollData does not have correct syntax*

*return 1*

*if a state forecast predicts zero electoral votes*

*return 2*

*if the entered party char is not alphabetic*

*return 3*

*run through pollData:*

*find start of first series of digits*

*add series of digits to empty string*

*convert string to int*

*if the party code in state forecast is the same as entered party code*

*add int to voteCount*

*reset int*

*finished running through poll data string*

*return 0*

countVotes calls three other functions: hasCorrectSyntax, hasNonzeroElectoralVotes, and convertVotesFromCharToInt. As evident from their names, when countVotes is checking if pollData has correct syntax it calls hasCorrectSyntax, when it checks if a state forecast has zero electoral votes it calls hasNonzeroElectoralVotes, and when it is converting the string of digits to an int it calls convertVotesFromCharToInt.

*hasNonzeroElectoralVotes*

This function will find the predicted electoral votes for each state forecast, convert it to an int, and check whether that int is equivalent to zero. Here is the hasNonzeroElectoralVotes pseudocode:

*run through pollData:*

*find start of first series of digits*

*add series of digits to empty string*

*convert string to int*

*if int equals zero*

*return false*

*return true*

If this function returns false, countVotes will then appropriately return 2.

*convertVotesFromCharToInt*

This function will take a string of digits and return the appropriate integer value. The function does this through two variables: tempValue and multiplyBy and a for loop. Here is the convertVotesFromCharToInt pseudocode:

*given a string of digits*

*multiplyBy equals zero*

*run through string:*

*convert current char to int*

*add it to (multiplyBy \* 10) and store in multiplyBy*

*store multiplyBy in finalValue*

*finished running through string*

*return finalValue*

The function ensures that regardless of how many digits are in the initial string, the proper integer value will get returned. This functionality is leveraged by countVotes to get the integer value and that makes up the entirety of the processing pollData part. Now let us see how the first part, pollData validity, works through the function hasCorrectSyntax.

*hasCorrectSyntax*

Once again this function is called from countVotes, and checks whether the pollData has correct syntax. It does this through two parts: one part checks whether pollData only has digit and alpha characters and one part checks the validity of each state forecast. Here is the hasCorrectSyntax pseudocode:

*given a string*

*if there are any characters which are not digit or alpha*

*return false*

*if any state forecast is invalid*

*return false*

*return true*

This function works by calling two other functions: hasOnlyDigitsAndAlphaCharacters and isValidStateForecast.

*hasOnlyDigitsAndAlphaCharacters*

This function simply does as it names says. It loops through the string and if there are any characters which are not digit or alpha it will return false. Here is the hasOnlyDigitsAndAlphaCharacters pseudocode:

*given a string of digits*

*run through string:*

*if a char is not alpha or not digit*

*return false*

*return true*

A correct poll data can only have alpha and digits, and this function ensures that.

*isValidStateForecast*

This function does three things which correspond to the three parts of a state forecast: it checks whether the state code for each state forecast is valid, it checks that the electoral votes has either 1 or 2 digits, and it checks that a party was indicated for each state forecast. Here is the isValidStateForecast pseudocode:

*given a string of digits*

*run through the string*

*check that the state code is valid*

*if not, return false*

*check that the number of electoral votes does not have more than 2 digits*

*if not, return false*

*check that an alpha follows electoral votes*

*if not, return false*

*return true*

Thus this function ensures each state forecast has proper syntax. It also calls another function: isValidUppercaseStateCode, which is the last function of my program.

*isValidUppercaseStateCode*

This function is fairly self explanatory; it checks that an uppercased state code is a valid one. Here is the pseudocode:

*given a list of valid state codes*

*return true if entered state code is the same as one of the state codes within the list*

Those are all the functions and their relevant descriptions for my program.

**C) Test Data**

**// Tests for Return Value**

Checks whether the function returns 1 when pollData is not valid:

*(“blahblah”, ‘d’, votes)*

Checks whether the function returns 2 when one of the state forecasts predicts 0 electoral votes:

*(“CA0D”, ‘d’, votes)*

Checks whether the function returns 3 when party is not a letter

*(“Ca55d”, ‘1’, votes)*

Checks whether the function returns 0 if everything is valid:

*(“ca55d”, ‘d’, votes)*

**// pollData tests**

Checks that pollData is valid given a correct state code regardless of its capitalization:

*(“CA55d”, ‘d’, votes)*

*(“ca55d”, ‘d’, votes)*

Checks that a space character located anywhere within pollData will cause pollData to not be valid and return 1:

*(“ CA55d”, ‘d’, votes)*

*(“ca55d ”, ‘d’, votes)*

*(“CA5 5d”, ‘d’, votes)*

Checks that a non-alphanumeric character within pollData will cause pollData to not be valid and return 1:

*(“CA&55d”, ‘d’, votes)*

Checks that pollData is valid only for a correct state code and not for any random combination of two alpha characters:

*(“CB55d”, ‘d’, votes)*

Checks that pollData is valid only if a party code follows the number of electoral votes for a correct state code:

*(“CA55”, ‘d’, votes)*

Checks that pollData is valid for an empty string:

*(“”, ‘d’, votes)*

Checks that pollData is valid given multiple correct state forecasts:

*(“TX38RCA55DMs6rnY29dUT06L”, ‘d’, votes)*

**// party code tests**

Checks that party code is valid given a correct party code regardless of its capitalization:

*(“CA55d”, ‘d’, votes)*

*(“CA55d”, ‘D’, votes)*

Checks that party code is not valid for an empty string and returns 3:

*(“CA55d”, ‘’, votes)*

**// votes test**

Checks that countVotes sets votes

*#include <cassert>*

*…*

*int main()*

*{*

*votes = -999;*

*assert(countVotes("TX38RCA55DMs6rnY29dUT06L", 'd', votes) == 0 && votes == 84);*

*votes = -999;*

*assert(countVotes("TX38RCA55D", '%', votes) == 3 && votes == -999);*

*}*

**// General tests**

Checks that countVotes depends on the indicated party code:

*(“TX38RCA55DMs6rnY29dUT06L”, ‘d’, votes) | votes = 84*

*(“TX38RCA55DMs6rnY29dUT06L”, ‘r’, votes) | votes = 44*

*(“TX38RCA55DMs6rnY29dUT06L”, ‘l’, votes) | votes = 6*

*(“TX38RCA55DMs6rnY29dUT06L”, ‘g’, votes) | votes = 0*

**// Use below if you want to copy and paste all of the above test cases**

(“blahblah”, ‘d’, votes)

(“CA0D”, ‘d’, votes)

(“Ca55d”, ‘1’, votes)

(“ca55d”, ‘d’, votes)

(“CA55d”, ‘d’, votes)

(“ca55d”, ‘d’, votes)

(“ CA55d”, ‘d’, votes)

(“ca55d ”, ‘d’, votes)

(“CA5 5d”, ‘d’, votes)

(“CA&55d”, ‘d’, votes)

(“CB55d”, ‘d’, votes)

(“CA55”, ‘d’, votes)

(“”, ‘d’, votes)

(“TX38RCA55DMs6rnY29dUT06L”, ‘d’, votes)

(“CA55d”, ‘d’, votes)

(“CA55d”, ‘D’, votes)

(“CA55d”, ‘’, votes)

(“TX38RCA55DMs6rnY29dUT06L”, ‘d’, votes) | votes = 84

(“TX38RCA55DMs6rnY29dUT06L”, ‘r’, votes) | votes = 44

(“TX38RCA55DMs6rnY29dUT06L”, ‘l’, votes) | votes = 6

(“TX38RCA55DMs6rnY29dUT06L”, ‘g’, votes) | votes = 0