Ethan Duong

UID: 805124044

Notable Obstacles:

I had difficulty keeping track of my string iteration variable. The biggest bug I had was that my getIntFromString function was built to increase iteration variable past the int. The only issue with this is that at the end of the for loop the variable would be increased one additional time which caused the code to skip characters that came after a number within the string. Furthermore, it was difficult trying to create code that would detect all incorrect test cases. At a certain point, I realized that I misread the project spec which forced me to refactor much of my code.

Overall Design:

* Function: getIntFromString takes as its input a string and a position and outputs an integer.
  + The function loops through a string until it sees a non-number character. The numbers it does see are built into an integer which the function returns. The position input is passed by reference. Once the function is finished this position value will be moved to the space after the last seen number.
* Function: isWellFormedGroceryString takes as its input a string and returns a Boolean: true if the string is valid and false if it is not.
  + The function tests if the string input is a valid grocery string by doing the following:
    - The function loops through the string checking each character
      * If the character is a number then it calls getIntFromString to obtain the entire number and move past the number
      * If the character is a non-number then it checks to see if it is a valid character. If the character is invalid the function returns false. Invalid characters include leading zeros and ‘-‘.
      * The function always expects the string to start with a number and end with a valid character. It checks to see that every number is followed by a character and vice versa. If any of the above checks fail the function returns false.
    - As the function loops it keeps track of the number of orders. If the orders exceed any limit cap then the function returns false.
    - If the function finishes reading the string without returning false then it returns true.
* Function(s): pickupCount/deliveryCount/inpersonCount/shipCount
  + The function takes as its input a grocery string and outputs an int: either -1 if the string is invalid or an int that represents the number counted.
    - The function uses isWellFormedGroceryString to test if the string is valid. If it is invalid, then return -1.
    - The function loops through the string storing in one variable each number it comes across using getIntFromString. After storing a number the function will come across a valid character (given that the string is valid). If the valid character represents the correct type of order (pickpupCount expects ‘P’ or ‘p’ for example) then the function adds the number to the total (if this were the pickupCount function then the total represents the total number of pickups). The variable is then set to the next number and the process continues until the end of the string. The function then returns the total.

Example Test Data:

* assert(isWellFormedGroceryOrderString("happyDays10") == false);

assert(isWellFormedGroceryOrderString("000001P") == false);

assert(isWellFormedGroceryOrderString("+1P") == false);

assert(isWellFormedGroceryOrderString("1p1d XYZ") == false);

assert(isWellFormedGroceryOrderString("10p10p10p1d1s1i") == false);

assert(isWellFormedGroceryOrderString("5d5d5d1p1s1i") == false);

assert(isWellFormedGroceryOrderString("50i50i50s") == false);

assert(isWellFormedGroceryOrderString("PDSI") == false);

assert(isWellFormedGroceryOrderString("P0DSI") == false);

assert(shipCount("5d5d5d1p1s1i") == -1);

assert(deliveryCount("5d5d5d1p1s1i") == -1);

* + The above test data checks if the program detects invalid grocery strings. Invalid strings include strings with invalid characters, strings that do not follow the correct pattern of int->char->int->char, and strings that contain invalid numbers such as numbers with leading zeros.
* assert(isWellFormedGroceryOrderString("1p1d1s1i") == true);

assert(isWellFormedGroceryOrderString("5d5p") == true);

assert(isWellFormedGroceryOrderString("1D10P4d4p") == true);

assert(pickupCount("5d5p") == 5);

assert(pickupCount("1p1d1s1i") == 1);

assert(pickupCount("5p1d1s1i6P") == 11);

assert(deliveryCount("5d5p") == 5);

assert(deliveryCount("4D1d1s3d") == 8);

assert(shipCount("5d5p") == 0);

assert(shipCount("1p1d1s1i") == 1);

assert(inpersonCount("5d5p") == 0);

assert(inpersonCount("1p1d1s1i") == 1);

assert(inpersonCount("5p1d1s31i6P") == 31);

* + This next set of test data tests how the program handles valid input. The count functions should be able to correctly count the number of their respective orders and all functions should be able to recognize the validity of their input strings.