Jordan Golan

UID: 405328335

1. The first obstacle I experienced when attempting to complete this project was how to account for a number that was multiple digits. For example, if a possible grocery string was “13i15d”. My original way for trying to code this project was to say that after every number, there must be an appropriate letter. However, this failed when numbers were more than one digit long. After attending office hours briefly, I learned to employ the use of Professor Stahl’s extractNumber function to aid me. This function would go through a string and extract a number, then update the position on the string. Thus, in my original loop, the next position that I could tell the code to look at is a letter. The second obstacle that I experienced in this project was getting the counters for each shopping type to accumulate as the code looped. My original code, for example, set the counter equal to the int number. My code looked like this for example: “ShipQuantity = number”. This, however, would cause logic errors when a string had the same letter more than once, as it would only count the last number. I solved this by having the variable accumulate as the code looped, setting “ShipQuantity = ShipQuantity + number”.
2. My program is designed in a manner that declared the five necessary functions; and one helper function at the beginning, has a main function that tests the functions, and lastly defines all the functions. The first function defined in my program is the extractNumber function provided by professor Stahl. Next, I defined the IsWellFormedGroceryOrderString. This function is designed to check a number of aspects of the string to make sure that it is valid. My first step in this function was to initialize variables, these include a bool variable that is initially true and changes to false if something is wrong with the string, and many counter variables. Next, I write two if statements that test two simple aspects of the string. These if statements ensure that the string is not empty, and the first char is a nonzero number. The checks to the string after, are dependent on order and thus require a loop to go through each character. I used a while loop, that ran as long as the position on the string, i, was less than the string size. This loop first created a variable number, that called the extractNumber function. It then checked if there were any other issues with the string. Using if statements, I checked if there were leading zeroes and if there were spaces and set result = false if there were. Next, I counted the number associated with each order type still within the while loop using if and else if statements. Lastly, within this function, I used the counter variables calculated in the while loop to write if statements that tested if there were more than 20 Pickup orders, more than 10 delivery orders, or more than 99 total orders, and set result = false if there was .

Next, I defined the functions pickupcount, deliverycount, shipcount, and inpersoncount. These four functions were all designed the same, with the only exception being the letter the code is searching for and the name of the counter variables. Similar to the IsWellFormedGroceryOrderString function, I used a while loop that went through the characters of a string. I first called the extractNumber function to an int variable, then used an if statement to only add this number to the order type counter if it was followed by the desired letter. For example, for pickupcount, if the next character was a ‘p’ or ’P’ then the number would accumulate in the variable PickupQuantity. The counter functions would finish by either returning the correct quantity or if IsWellFormedGroceryOrderString was false, it would return the int -1.

1. List of test data:
   1. “” – this tested IsWellFormedGroceryOrderString an empty string
   2. “p10” – this tested IsWellFormedGroceryOrderString that the first character had to be a number
   3. “ 5p03s” – this tested IsWellFormedGroceryOrderString that there should be no leading zeros
   4. “10s 5p” – this tested IsWellFormedGroceryOrderString that there are no spaces
   5. “1p1P1p1P” – this tested IsWellFormedGroceryOrderString and pickupcount that the counter variables accumulate when a letter appears more than once in a string
   6. “1s1S1s” – this tested IsWellFormedGroceryOrderString and shipcount that the counter variables accumulate when a letter appears more than once in a string
   7. “1i2I1i” – this tested IsWellFormedGroceryOrderString and inpersoncount that the counter variables accumulate when a letter appears more than once in a string
   8. “1d1D2d” - this tested IsWellFormedGroceryOrderString and deliverycount that the counter variables accumulate when a letter appears more than once in a string
   9. “25p4i” – this tested IsWellFormedGroceryOrderString if there were more than 20 pickup orders
   10. “45d5i” – this tested IsWellFormedGroceryOrderString if there were more than 10 delivery orders
   11. “100i45s” – this tested IsWellFormedGroceryOrderString if there were more than 99 total orders
   12. “fgdjhkf” – this tested all functioned to make sure a function with no numbers returned false and that the counter functions then returned -1
   13. “10p4d5i7s” – this valid function tested that all counters returned the proper result.
   14. “10p4p6d7I20S5s” – this valid string tested that all counter returned the proper result, with repeated numbers and capital letters.