005994409

Timothy Liu

Dis 2A

1. A brief description of notable obstacles you overcame.

I had trouble going through the index of the order string and determining whether all the components of the order (quantity, “:”, menu item) were all in the correct order. First, I had trouble checking whether a number would be a single or double digit, so I created two variables that would read the indices of the string. One would go through each character of the string to check all the components, while another index would simply designate the start index for the extractNumber function to begin reading. That way, whether there was a one digit or two digit number, the difference of the reading index and the extractNumber index would be 1 if you had a 1 digit number and 2 if you had a 2 digit number. So, the extractNumber function would always begin reading at the correct place, and could change the reading position depending on the number of digits you had.

Another problem I had was getting my isValidOrderString function to continue reading if there was an “\_” in place. Originally, I tried checking if the last character would be “\_” but orders wouldn’t end in “\_”. Then, I tried checking if the last character would be an item menu, but I couldn’t check for “\_” because I couldn’t account for whether there was an “\_” after. My solution was to check if the reading index would go out of bounds for the string, which would indicate that the order was done entirely. If this wasn’t the case, I could continue on to the next character and check if there was an “\_” and if there was, I would just go through the loop again.

1. A description of the design of your program.

For the isValidOrderString function, my code will use a for loop to go through the entire order string and check if there are any unsupported characters which aren’t the numbers 0-9, “:”, and “\_”. Then another for loop is used to read the entire string and first check for the quantity, the “:” character, the menu item, and an “\_” character in that order. The program will read up to the first “:” and will then extract the number before the “:” and check whether that number is valid. It will also add that number to a variable called totalQuantity, which keeps track of how many items in total there are. The program will then continue reading onto the menu item and checks whether the character before is a colon. After, it will check if the character at the current position of the string matches with a menu item. If it’s valid, it will do nothing and read the next position of the string and check if that’s still in bounds or not. If it’s not in bounds, it indicates that the order has ended. If the next position of the string is still in bounds, it’ll check for a “\_” to continue reading onto the next order. If the next position of the string is still in bounds, but it doesn’t have an “\_”then the order string isn’t valid

For the cost function, the program will use a for loop to go through the entire order string and check for the quantity, the “:” character, the menu item, and an “\_” character in that order. The program will read up to the first “:” and will then extract the number before the “:” and check whether that number is valid. It will also add that number to a variable called quantity, which keeps track of how many items in the current order there are. Then, it will continue reading to the menu item and checks whether the character before is a colon. After, it will check if the character at the current position of the string matches with a menu item. If it matches with a menu item, it will multiply the quantity of the current order by the price of that menu item they ordered. Then, the program will continue on to read the next position of the string and check if that’s still in bounds or not. If it’s not in bounds, it indicates that the order has ended. If the next position of the string is still in bounds, it’ll check for a single “\_” to continue reading onto the next order. If there are multiple “\_”, the order string will be considered invalid. If the next position of the string is still in bounds, but it doesn’t have an “\_”then the order string isn’t valid

For howManyCombos()

For the howManyCombos() function, the program will first check whether the combo they are checking is a 1, 2, or 3 otherwise that combo is not an item on the menu. Then, a for loop is used to go through the entire order string and check for the quantity, the “:” character, the menu item, and an “\_” character in that order. The program will read up to the first “:” and will then extract the number before the “:” and check whether that number is valid. It will also add that number to a variable called quantity, which keeps track of how many items in the current order there are. Then, it will continue reading to the menu item and checks whether the character before is a colon. After, it will check if the character at the current position of the string matches with the combo item they’re checking. If it matches with the combo item the user is checking, it will add the quantity of the current order to the total quantity of the combos the user is checking. Then, the program will continue on to read the next position of the string and check if that’s still in bounds or not. If it’s not in bounds, it indicates that the order has ended. If the next position of the string is still in bounds, it’ll check for a “\_” to continue reading onto the next order. If the next position of the string is still in bounds, but it doesn’t have an “\_”then the order string isn’t valid

For howManyShakes()

For the howManyShakes() function, the program will first check whether the shake they are checking is a “S”,”V”, or “C” otherwise that shake is not an item on the menu. Then, a for loop is used to go through the entire order string and check for the quantity, the “:” character, the menu item, and an “\_” character in that order. The program will read up to the first “:” and will then extract the number before the “:” and check whether that number is valid. It will also add that number to a variable called quantity, which keeps track of how many items in the current order there are. Then, it will continue reading to the menu item and checks whether the character before is a colon. After, it will check if the character at the current position of the string matches with the shake they’re checking. If it matches with the shake the user is checking, it will add the quantity of the current order to the total quantity of the shake the user is checking. Then, the program will continue on to read the next position of the string and check if that’s still in bounds or not. If it’s not in bounds, it indicates that the order has ended. If the next position of the string is still in bounds, it’ll check for a “\_” to continue reading onto the next order. If the next position of the string is still in bounds, but it doesn’t have an “\_”then the order string isn’t valid

1. A list of the test data that could be used to thoroughly test your program, along with the reason for each test. You don't have to include the results of the tests, but you must note which test cases your program does not handle correctly. (This could happen if you didn't have time to write a complete solution, or if you ran out of time while still debugging a supposedly complete solution.)

|  |  |  |
| --- | --- | --- |
| String | Output | Comment |
| 20:1\_20:1 | Correctly handles | Duplicate order of a menu item |
| 1:1\_1:2\_1:3\_1:S\_1:V\_1:C | Correctly handles | One of every item |
| 01:2 | Correctly handles | Leading zero quantity |
| 4:4 | Correctly handles | Item not on the menu |
| \_2:1 | Correctly handles | Leading “\_” character |
| S:1\_2:2 | Correctly handles | Leading menu item character |
| 20:2\_20:1\_10:V | Correctly handles | Exactly 50 items |
| 2:1\_3: | Correctly handles | Missing menu item |
| 2:V\_ | Correctly handles | Ending “\_” character |
| 2:3\_0:S\_9:V | Correctly handles | 0 quantity order |
| ^3:S | Correctly handles | Unsupported character in order |
| 2:1\_\_3:V | Correctly handles | Multiple “\_\_” between orders |
| 20:1\_-2:1 | Correctly handles | Negative quantity |
| 4:c | Correctly handles | Lowercase letter |
| 4: 1 | Correctly handles | White space in string |
| 2 :1 | Correctly handles | White space in string |
| 20.2:2 | Correctly handles | Decimal number |
| :1\_2:3 | Correctly handles | Missing quantiy |