**CS31 Project 3 Report**

**Benjamin He**

**UID: 804962948**

**a. Notable Obstacles**

The main obstacle I had with this program was with the isValidEbayListingString function, more specifically trying to figure out a way for the program to check for a bunch of different conditions for each character in the input string to give an accurate determination of whether the string was valid or not. The way I wrote the program was by using a bunch of different Boolean variables to keep track of the different conditions going on, so then each time a character was read, it would be filtered through a series of if statements to check whether the character is valid at its given place, time, and order in the auction string.

Another issue I had was with deciding when to switch the values of my Boolean tracking variables. Initially I would often find that the program completely a block of code that was supposed to execute with a given character didn’t execute because I changed the value of a Boolean tracking variable too early or late, so the if statement that checks the tracking variable would receive the wrong value and do the wrong action.

Finally, I had an issue with the curly brackets again. There was a portion of the code that kept executing incorrectly and I got frustrated because all the syntax was correct and when I looked at the code, the logic also seemed correct, but in the end I realized I had just forgotten to put curly brackets around a multi-line block of statements that were supposed to execute only once an if statement was satisfied. Since I was missing the brackets, only the first line was restricted by the if statement and the rest of that block would execute regardless, which I only discovered after using the debugger and going through the entire program.

**b. Program Design**

bool isValidEbayListingString(string auctionString)

The largest function by far is this one which checks whether the input for auctionstring is allowed according to the rules in the project and is used in the other required functions. The primary elements of my design for this function are a for loop and a switch statement with a ton of if-else statements nested inside. Before anything else, isValidEbayListingString creates a Boolean called isValid which is what’s returned at the end of the function. isValid is initialized to be true because I designed my function to check for any rules being broken and changes isValid to false if finds something bad, but if nothing bad is found, the function will return true by default.

Then I put an if-else statement that only allows the for loop to begin looking through the string if the string isn’t empty, otherwise the function will immediately end and return false. If auctionstring is not empty, then a bunch of Boolean tracker variables (which will be referred to as “trackers” for the rest of the report) are created and initialized (the purpose of each tracker is under the tracker declaration below) as well as two integers watch and unwatch to keep track of the number of watches and unwatches that happen in the string. Then a for loop cycles through the auctionstring (from index i = 0 to i = auctionstring.size()-1) and puts every character through a switch that determines the validity of the character using the trackers. (Description continues after box)

bool lExists = false;

/\* This tracker determines whether there is an ‘l’ or ‘L’ in the auctionstring. It becomes true and stays that way permanently once the switch statement in the for loop sees the first ‘l’ or ‘L’ that appears in the auctionstring. This tracker is checked by every other element in the switch statement and if lExists = false when the switch statement sees any other character, isValid = false because ‘l’ or ‘L’ must be the first character that appears in the auctionstring otherwise the auctionstring is not valid. The only exception is with case ‘l’ and case ‘L’ where if lExists = true then the string is invalid because that means ‘l’/’L’ has already appeared in the string and now the for loop found another ‘l’/’L’ in the string, which is not allowed. \*/

bool onL = false;

bool onB = false;

bool onPlus = false;

/\* These three trackers determine whether a listing price, bid, or bid raise is being established. When the switch sees an ‘l’/’L’, ‘b’/’B’, or ‘+’, it changes onL, onB, or onPlus to true, respectively. All the cases except case ‘l’ and case ‘L’ check if any of these “on” variables are true because if they are, then only number characters are valid and any other letter characters are invalid. There is an exception though: after the first ‘b’/’B’, any following ‘b’/’B’ characters must be followed by a ‘+’. Another tracker firstB (see below) takes care of that exception. Any of these three variables, after becoming true, can only be returned to false after a number is read fully. This ensures that the listing price, bid, or bid raise completes before any other commands happen.\*/

bool firstB = true;

/\* This variable is changed to false once the first ‘b’/’B’ in the auctionstring is found and stays that way permanently for the rest of the execution. Once firstB becomes false, any ‘b’/’B’ character must be followed by a ‘+’ otherwise the string is invalid. This is because only the initial ‘b’/’B’ is a bid and the rest of the times ‘b’/’B’ appears are for add-ons to the initial bid, which also requires a ‘+’. \*/

bool numGo = false;

/\* The purpose of this tracker is to determine when the ‘0’ is allowed. When a number starts after a ‘l’, ‘b’ or ‘+’ , the first digit cannot be ‘0’, but ‘0’ can appear after the first digit, so when a digit other than ‘0’ is read, numGo is switched to true, which means that ‘0’ is now allowed to appear. Then, if the current index is not the last one, the cases for ‘0’ through ‘9’ preemptively look at the character at index i+1 to see whether the number ends. Another switch statement is used, and if it detects that the character at i+1 is no longer a digit, the switch will turn onL, onB, and onPlus to false so that more letters can follow. \*/

In the switch statement, I have the following cases: case ‘l’, case ‘L’, case ‘b’, case ‘B’, case ‘+’, case ‘U’, case ‘u’, case ‘w’, case ‘W’, case ‘0’, case ‘1’, case ‘2’, case ‘3’, case ‘4’, case ‘5’, case ‘6’, case ‘7’, case ‘8’, and case ‘9’. For cases ‘l’ and ‘L’, the switch checks lExists. If lExists is true, then it sets isValid to false because it means that there are multiple ‘l’/’L’ characters in the string even though there should be exactly one in the beginning of the string. If lExists is false, then lExists is set to true, onL is set to true, and the for loop continues. For cases ‘B’ and ‘b’, first lExists is checked, because if lExists = false, that means the first character of the string is not ‘l’/’L’, so isValid is set to false. If lExists is true, then the switch also checks if onL, onB, or onPlus are true. If any one of these are true, that means that a previous listing price, bid, or bid raise hasn’t completed yet, so isValid is again set to false. If both those if statements are passed, then onB is set to true and the loop continues. For case ‘+’, again lExists is checked, then onB and firstB are checked with an if statement. If onB=true AND firstB=false, then the string is valid, otherwise, isValid = false because onB=true means the plus is immediately following a ‘b’/’B’, but it should only be following that character if firstB=false because there should not be a ‘+’ after the very first ‘b’/’B’ in auctionstring. For cases ‘W’ and ‘w’, lExists and the three “on” trackers are checked, then if those filters are passed, watch is incremented and the loop continues. For cases ‘u’ and ‘U’, lExists and the three “on” trackers are checked, but since the number of unwatches can never exceed the number of watches, there is another if statement where if unwatch<watch, then the unwatch will increment, otherwise isValid is set to false. For case ‘0’, numGo is checked to make sure that ‘0’ is not the first character of a number in the string. Then it continues to cases ‘1’, ‘2’,…’9’ where lExists is checked again. The three “on” trackers are checked too, but this time at least one of them should be true to indicate that a number should appear. The switch then checks the next element in the string if the current index is not the last one, and if the next character is a number, numGo is kept true, so that ‘0’ is allowed to appear, but if the next character is not a number, then numGo and all three “on” trackers are set to false so that other characters are allowed to appear. After the for loop ends, I put one more switch statement to make sure that the last character of the auctionstring is either a digit, watch, or unwatch., otherwise isValid is set to false. Finally, isValid is returned. If the auctionstring managed to pass through all the filters and if statements in this function, then it means that auctionstring is valid and isValid will still be true by the end of the function.

bool listingSold(string auctionstring)

This function also uses a switch statement inside a for loop to check through all the characters of the auctionstring to execute whatever commands are contained in the string. Because the auctionstring is verified to be legal, it can be much simpler than the one above. It does so by calling another function I created called convertToNumber which takes the auction string and the current index as parameters and converts the string of number characters into the actual integer that the characters represent. convertToNumber is another for loop with a switch statement inside that checks each number then multiplies the current sum by ten and adds the number and returns the final integer to the caller. Anyway, for L, B, or B+ commands, the switch will call convertToNumber which converts the following number to do something with it. If it’s called from L’s cases, then a variable called listingPrice is set to that number, when B’s cases calls convertToNumber, it is set as the initial value of the bid, stored in the bid variable , then subsequent B+ cases will add their sums to this initial bid to calculate the final bid and compare it with listingPrice. If bid>listingPrice, this function returns true, otherwise it returns false. This function also automatically returns false in the beginning if auctionstring is not a valid string according to isValidEbayListingString using an if statement to check.

int howMuch(string auctionstring)

This function is pretty much identical to listingSold except that it just returns the final value of bid rather than comparing with the initial listing price, which means that there is also no need to store the value of the listing price in the beginning. Before doing the calculations, though, there are two if statements, one that checks if auctionstring is valid. If it isn’t, this function returns -1. Also, if the auctionstring is valid but the object was not sold (listingSold = false), then this function returns 0.

**c. Test Data**

|  |  |
| --- | --- |
| Test Input | Reason |
| isValidEbayListingString("") | ensure program knows empty string is invalid |
| isValidEbayListingString(" ") | ensure program recognizes invalid character in string |
| isValidEbayListingString("a") | ensure program recognizes invalid character in string |
| isValidEbayListingString("L") | ensure program recognizes uncompleted listing as invalid |
| isValidEbayListingString("l50") | ensure program recognizes only listing price as valid |
| isValidEbayListingString("L50") | ensure program also recognizes uppercase |
| isValidEbayListingString("L00050") | ensure program recognizes that 0's at the beginning of a number is invalid |
| isValidEbayListingString("L-1") | recognize another invalid character |
| isValidEbayListingString("Lambasdff ") | recognize invalid characters |
| isValidEbayListingString("L50L50L50l50") | recognize that L can only appear once in string |
| isValidEbayListingString("LLLL") | recognize that L can only appear once in string |
| isValidEbayListingString("L100 B50 W B+10 U W B+40 W") | recognize that spaces aren't allowed in auctionstring |
| isValidEbayListingString("L100.50B50") | recognize that decimal point is invalid |
| isValidEbayListingString("L100B50UW") | recognize that unwatch can't be before watch |
| isValidEbayListingString("L100BW50") | recognize that bid has to complete before watch occurs |
| isValidEbayListingString("L100B50WBU+50") | recognize that raising bid also has to complete before watch occurs |
| isValidEbayListingString("L100B50UUUUUUUUU") | recognize that unwatches cannot exceed watches |
| isValidEbayListingString("L100B50WWUUUUUUUUU") | recognize that unwatches cannot exceed watches |
| isValidEbayListingString("WUL100") | recognize that watching/unwatching cannot occur before listing price is established |
| isValidEbayListingString("L100L50") | recognize that there cannot be multiple L's in string |
| isValidEbayListingString("L100B+50") | recognize that an initial bid must be established first |
| isValidEbayListingString("L100B50B+50B20") | recognize that after the initial bid, all other bids must have a + sign |
| isValidEbayListingString("LBWuUWW+") | recognize that unwatch cannot occur before watch |
| isValidEbayListingString("L99B50WB+10UWB+40W") | recognize that this string is valid |
| isValidEbayListingString("l99b50wb+10uwb+40w") | recognize lowercase versions of commands |
| isValidEbayListingString("L100B99") | recognize that this is a valid string |
| isValidEbayListingString("L100B50WWWWWWW") | recognize that this is a valid string |
| isValidEbayListingString("L100B50WUWUWUWWWWW") | recognize that this is a valid string |
| isValidEbayListingString("L100B50WWUUB+10WWUU") | recognize that this is a valid string |
| isValidEbayListingString("L100W") | recognize that this is a valid string |
| isValidEbayListingString("L100WU") | recognize that this is a valid string |
| isValidEbayListingString("L100WB50WU") | recognize that this is a valid string |
|  |  |
| listingSold("0") | recognize that if the string isn't valid, false should be returned |
| listingSold("l50") | testing program correctness |
| listingSold("L50") | testing program correctness |
| listingSold("L100B99") | testing program correctness |
| listingSold("L100B50WWWWWWW") | testing program correctness |
| listingSold("L100B50WUWUWUWWWWW") | testing program correctness |
| listingSold("L100B50WWUUB+10WWUU") | testing program correctness |
| listingSold("L100W") | testing program correctness |
| listingSold("L100WU") | testing program correctness |
|  |  |
| watchers("L99B50WB+10UWB+40W") | make sure calculations are correct |
| watchers("L100WB50WUB+30WUWUB+70") | make sure calculations are correct |
| watchers("l50b50wb+15uwb+40w") | make sure calculations are correct |
|  |  |
| howMuch("0") | make sure function's returns -1 for invalid string |
| howMuch("l50") | make sure function's calculations are correct for various strings |
| howMuch("L50") | make sure function's calculations are correct for various strings |
| howMuch("L100B99") | make sure function's calculations are correct for various strings |
| howMuch("L100B50WWWWWWW") | make sure function's calculations are correct for various strings |
| howMuch("L100B50WUWUWUWWWWW") | make sure function's calculations are correct for various strings |
| howMuch("L100B50WWUUB+10WWUU") | make sure function's calculations are correct for various strings |
| howMuch("L100W") | make sure function's calculations are correct for various strings |
| howMuch("L100WU") | make sure function's calculations are correct for various strings |
| howMuch("L99B50WB+10UWB+40W") | make sure function's calculations are correct for various strings |
| howMuch("L100WB50WUB+30WUWUB+70") | make sure function's calculations are correct for various strings |
| howMuch("l50b50wb+15uwb+40w") | make sure function's calculations are correct for various strings |