**BOOTH’S ALGORITHM**

PROBLEM STATEMENT

 "Implement the Booth's algorithm for multiplying binary numbers"

SOLUTION

In this project, I have designed an implementation of Booth’s Algorithm for multiplying binary numbers. The program takes two integer number as an input. The program implements the booth’s algorithm wih the two integer inputs and prints the binary representation of the product of the two input integer numbers.

***Programming Language***: Java

ASSUMPTIONS

The following are the assumptions that we have taken while building theBooth’s Algorithm:

1. The number entered as input are integers i.e the range is -231 to 231– 1.
2. The resultant product’s binary value is printed on the screen.

GETTING STARTED

Please follow these instructions to use the Booth’s Algorithm:

1. Pick any number from the range mentioned in the assumption.
2. Load the java program on to your compiler and hit the run button.
3. When the program will rum you will be prompted to enter the two numbers you pick,after typing the numbers hit enter.
4. After that you would get the desired product in binary representation.

ERRORS HANDLED

The following are the errors that the assembler handles after reading the given assembly code:

1. ERROR: Number entered is out of range

SUPPORT

If you have any issues regarding this project, feel free to contact me.

Vani Sikka vani19395@iiitd.ac.in

**UNDERSTANDING THE BOOTH’S ALGORITHM**

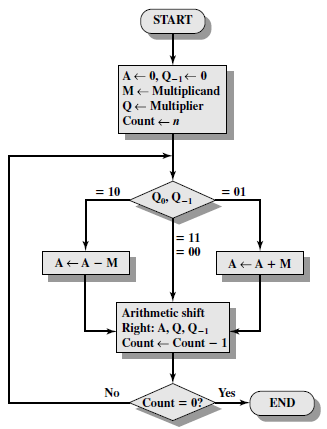
Booth algorithm gives a procedure for **multiplying binary integers** in signed 2’s complement representation **in efficient way**, i.e., less number of dditions/subtractions required. It operates on the fact that strings of 0’s in the multiplier require no addition but just shifting and a string of 1’s in the multiplier from bit weight 2^k to weight 2^m can be treated as 2^(k+1 ) to 2^m.

As in all multiplication schemes, booth algorithm requires examination **of the multiplier bits** and shifting of the partial product. Prior to the shifting, the multiplicand may be added to the partial product, subtracted from the partial product, or left unchanged according to following rules:

1. The multiplicand is subtracted from the partial product upon encountering the first least significant 1 in a string of 1’s in the multiplier
2. The multiplicand is added to the partial product upon encountering the first 0 (provided that there was a previous ‘1’) in a string of 0’s in the multiplier.
3. The partial product does not change when the multiplier bit is identical to the previous multiplier bit.

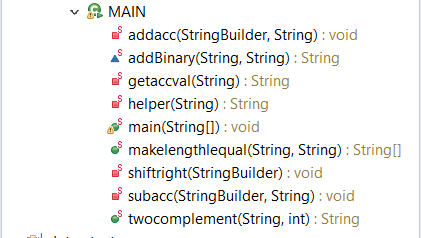
(Above imformation is from site <https://www.geeksforgeeks.org/computer-organization-booths-algorithm/>)

Chart portraying Booth’s Algorithm:



**LIST OF FUNCTIONS**

Herby, follows a list of all the functions used in the Java program for the Booth’s Algorithm:



FUNCTIONS AND THEIR EXPLAINATION

Hereby, follows the list of functions used in the program with a quick explaination about them.   
The list is sorted in the order of the appearance of the functions in the code.

**public** **static** **void** main(String[] args)

The execution of the program begins with main().This takes the two integer numbe as input it converts them to binary string also convert the negative number to the two’s complement.It performs the booth’s algorithm on the numbers prints all the partial products at each stage.In the end it print the result in the binary form on the output scene.

**private** **static** **void** subacc(StringBuilder num, String m)

The function takes the partial product and binary string to be subtracted (m)as parameters.It calls helper function to convert m to it's two's complement.Then the function picks first m's length characters from the partial product(str) and add m to it by calling addBinary.The function will return the addition result.Then the function compare if the size of the result is size of m. If it is not then it means that there is a carry which we have to ignore so I update the partial product with the substring of the addition result which start from 1.But if the length is equal I update the partial product with the addition result.

**private** **static** String helper (String s)

This function converts the binary string of a number to it's two's complement. This piece of function is referenced from <https://www.geeksforgeeks.org/efficient-method-2s-complement-binary-string/>

**private** **static** **void** addacc(StringBuilder num,String m)

The function takes the partial product and binary string to be added (m) as parameters. Then the function picks first m's length characters from the partial product(str) and add m to it by calling addBinary.The function will return the addition result. Then the function compare if the size of the result is size of m. If it is not then it means that there is a carry which we have to ignore so I update the partial product with the substring of the addition result which start from 1.But if the length is equal I update the partial product with the addition result.

**static** String addBinary(String a, String b)

The function takes two binary string numbers add the two binary number. Return the binary representation of the addition result. This piece of function is referenced from <https://www.geeksforgeeks.org/program-to-add-two-binary-strings/>

**private** **static** **void** shiftright(StringBuilder num)

This function takes in the partial product updated the parial product by shifting all the digits by one place.

**private** **static** String getaccval(String m)

this function returns the a string of 0's whose length is equal to m's length.eg m="1001"(length of m=4) so the function returns "0000"(it has length 4)

**public** **static** String twocomplement(String a,**int** val)

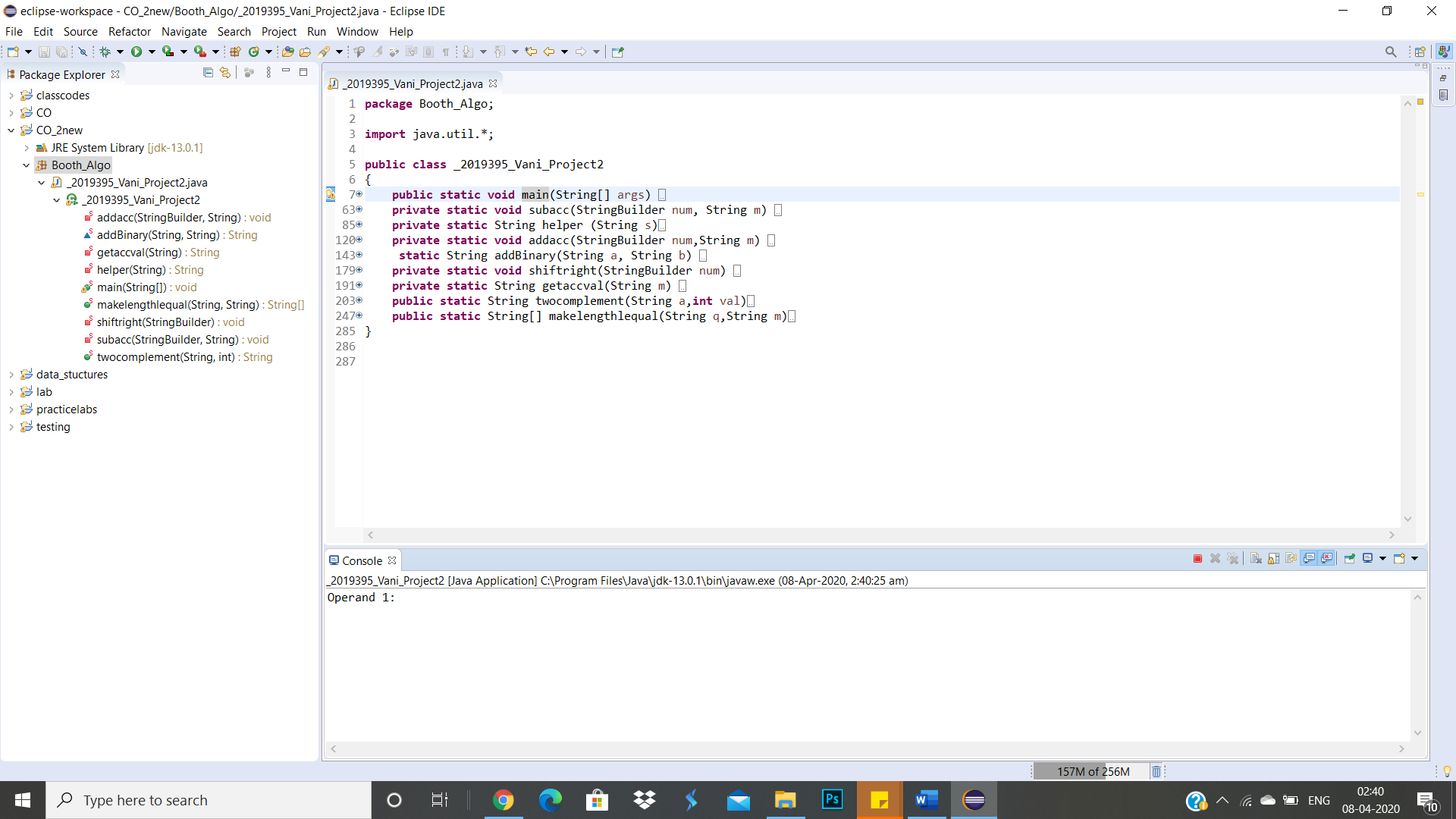
if val is more than zero then a string gets a 0 in the front and is returned as it is with additional 0 in front of it.But if the val is less than 0 then we put 0 in the front of a then calculate the 2's complement of a (which has additional 0 not the original a passed as parameter) .Hence return the String that contain the 2's complement of a. This piece of function is referenced from <https://www.geeksforgeeks.org/efficient-method-2s-complement-binary-string/>

**public** **static** String[] makelengthlequal(String q,String m)

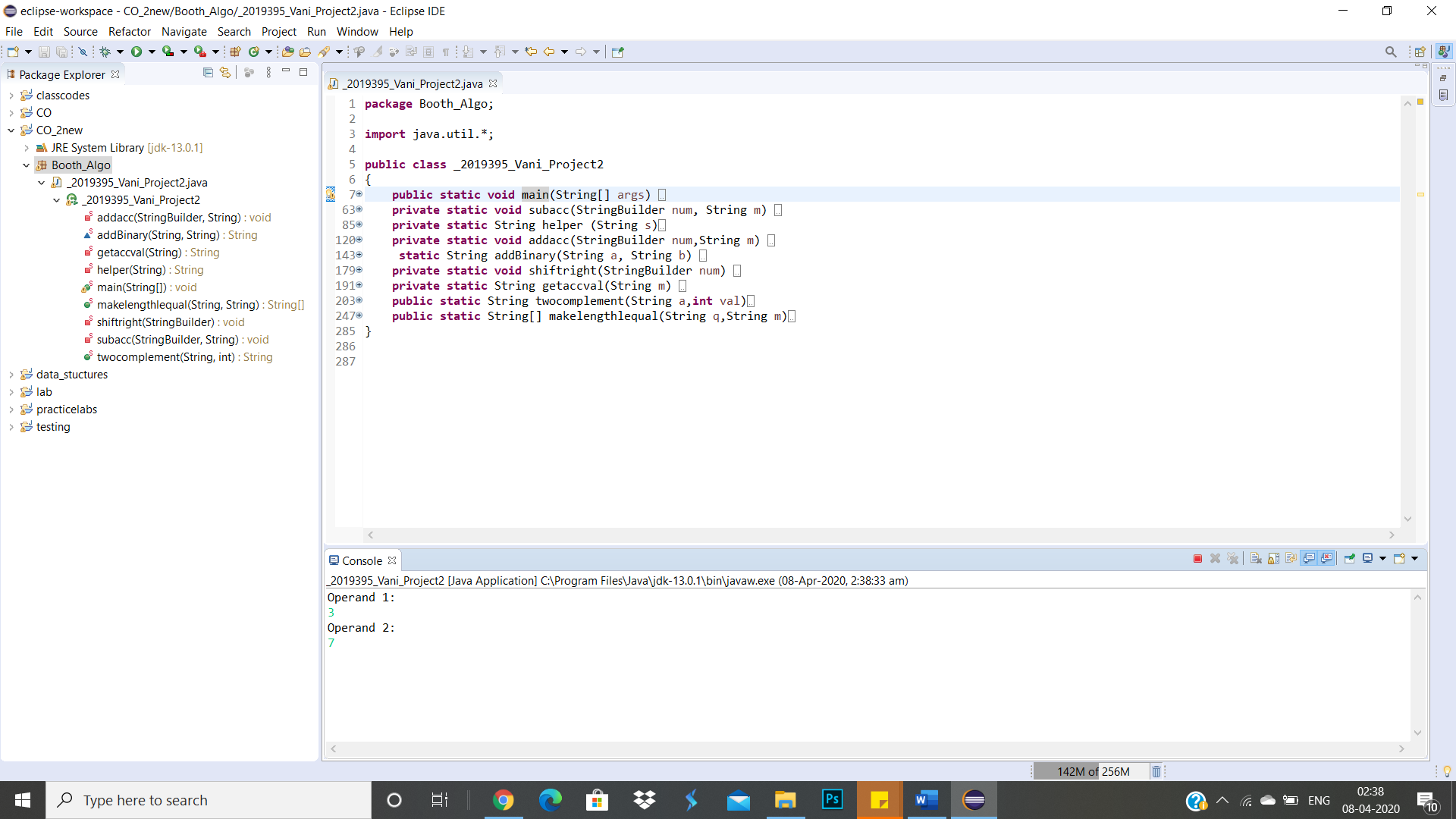
This function takes two binary number as inputs. Then it see if their length of both the binary numbers are same then it would return booth the numbers as is in the an array where the first element would be q number and second element will be m number.But if the numbers are not of the same size then the smaller number must be add with lsb bit in the front till small one's length become equal to larger one's length.eg:q="1001" m="100" so m will be changed to "1100"or q="100" m="01" so now m become "001".Once q and m is ready the it puts in the string array where q is the first element and m is the second element.

RUNNING THE BOOTH’S ALGORITHM

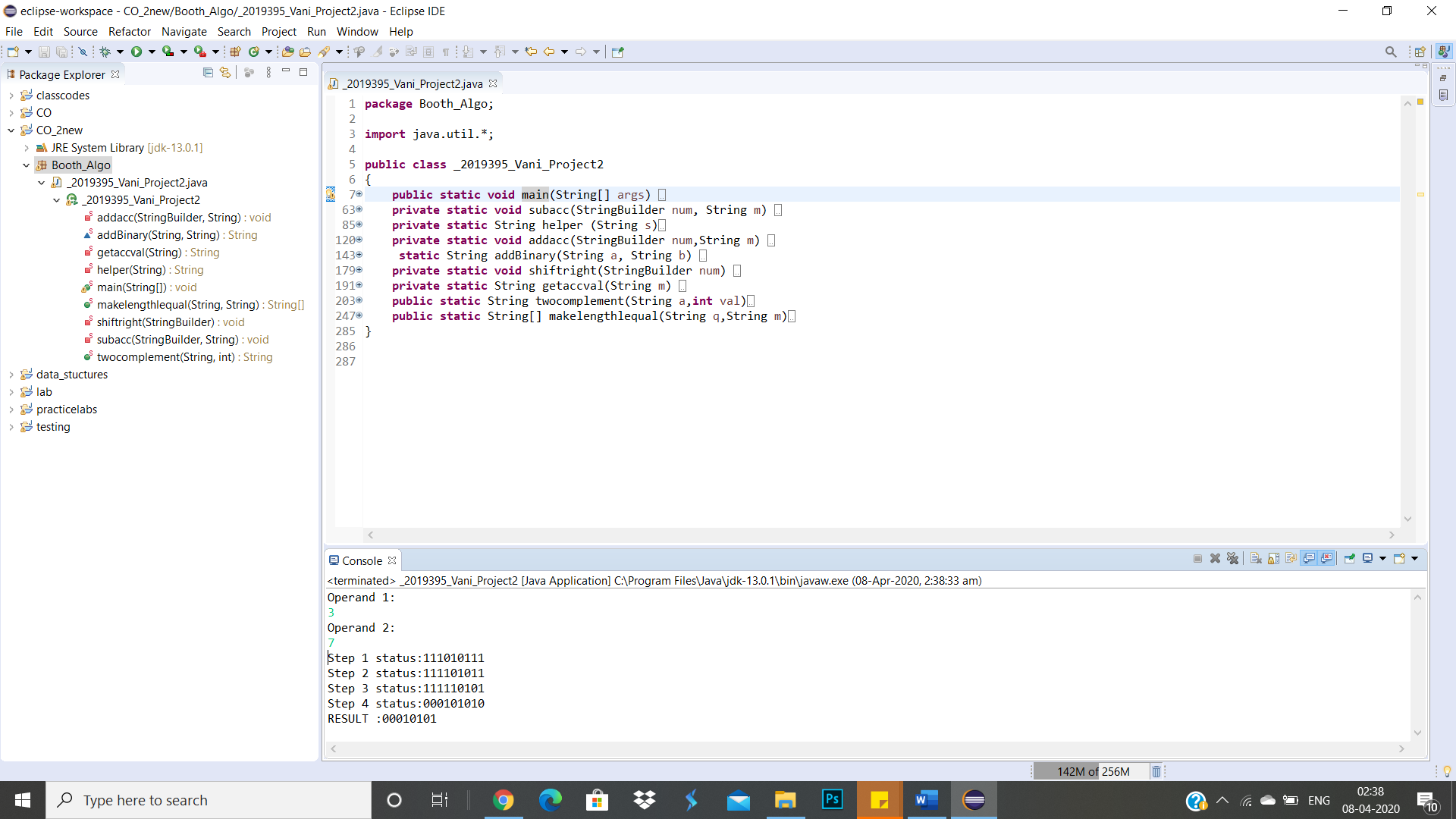
Herby are few pictures of the working of the two-pass assembler with a short description followed for each one of them.



Screen would look like the above when you would load the code.



Screen when Inputs are entered.



Screen Showing the results of the inputs entered.