Theory of Computation

To simulate Deterministic Finite Automata (DFA).

ALGORITHM

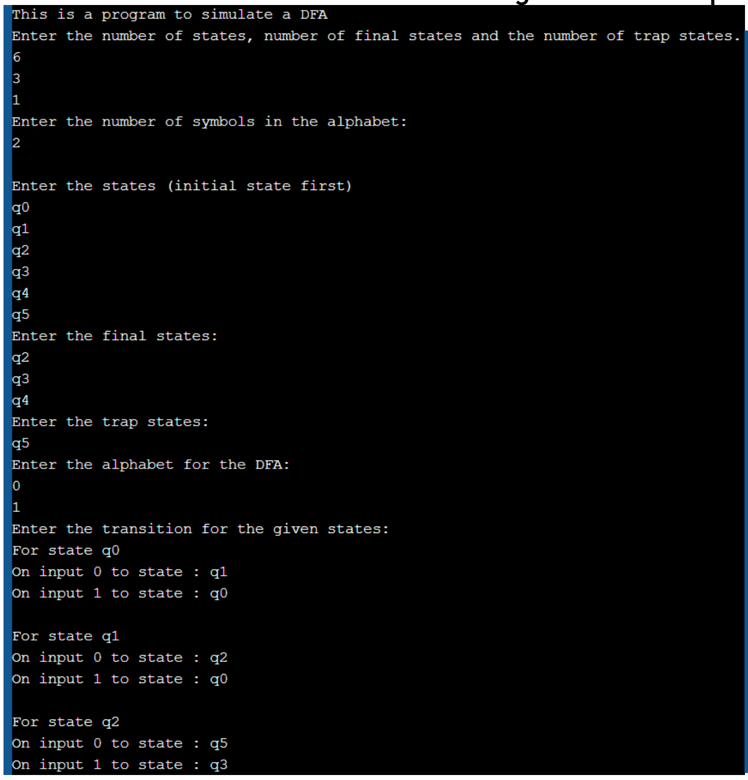
We are constructing a DFA where W ∈ { a, b }* and w consists of an even number of both a and b.

1. User enters the string of any length.
2. Now we will create a function checkString which accepts the string as input and returns whether the string is accepted or not.
3. We will declare a variable dfa to know the present state of the string, and initiate it to 0.
4. Now we will access every element of the string one by one and follow according to the below table

Dfa (present stage)	Updated dfa when input = a	Updated dfa when input = b
0	2	1
1	3	0
2	0	3
3	1	2

After iterating all the elements of the string we will see whether the string acceptable or not based on the value of dfa

If the dfa =0 or dfa = 3 then the string will be accepted otherwise the string won't be accepted.

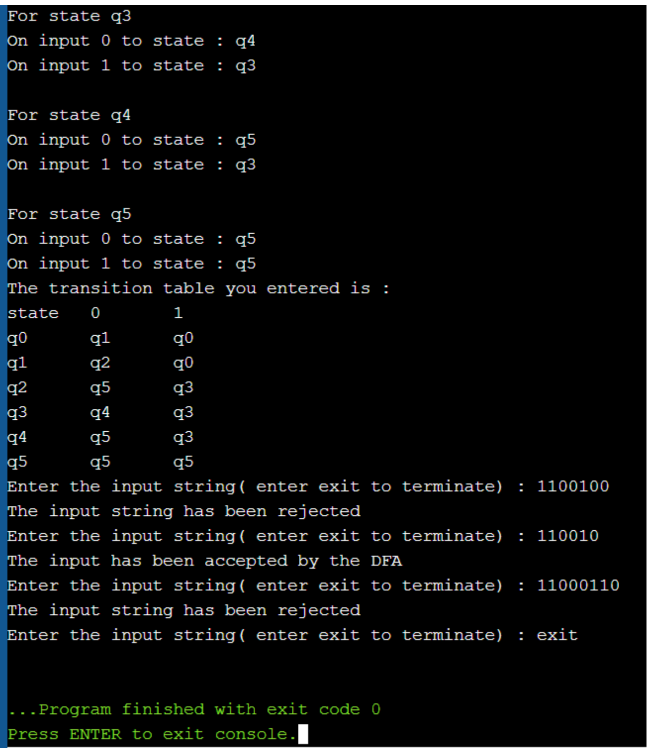


```
This is a program to simulate a DFA
Enter the number of states, number of final states and the number of trap states.
6
3
1
Enter the number of symbols in the alphabet:
2

Enter the states (initial state first)
q0
q1
q2
q3
q4
q5
Enter the final states:
q2
q3
q4
Enter the trap states:
q5
Enter the alphabet for the DFA:
0
1
Enter the transition for the given states:
For state q0
On input 0 to state : q1
On input 1 to state : q0

For state q1
On input 0 to state : q2
On input 1 to state : q0

For state q2
On input 0 to state : q5
On input 1 to state : q3
```



```
For state q3
On input 0 to state : q4
On input 1 to state : q3

For state q4
On input 0 to state : q5
On input 1 to state : q3

For state q5
On input 0 to state : q5
On input 1 to state : q5
The transition table you entered is :
state   0       1
q0      q1      q0
q1      q2      q0
q2      q5      q3
q3      q4      q3
q4      q5      q3
q5      q5      q5
Enter the input string( enter exit to terminate) : 1100100
The input string has been rejected
Enter the input string( enter exit to terminate) : 110010
The input has been accepted by the DFA
Enter the input string( enter exit to terminate) : 11000110
The input string has been rejected
Enter the input string( enter exit to terminate) : exit


...Program finished with exit code 0
Press ENTER to exit console.
```

Construct DFA from NFA.

THEORY

The steps involved in transforming an NFA into a DFA are:

Step 1 : Take ε-closure for the beginning state of NFA as beginning state of DFA.

Step 2 : Find the states that can be traversed from the present for each input symbol (union of transition value and their closures for each states of NFA present in current state of DFA).

Step 3 : If any new state is found take it as current state and repeat step 2.

Step 4 : Do repeat Step 2 and Step 3 until no new state present in DFA transition table.

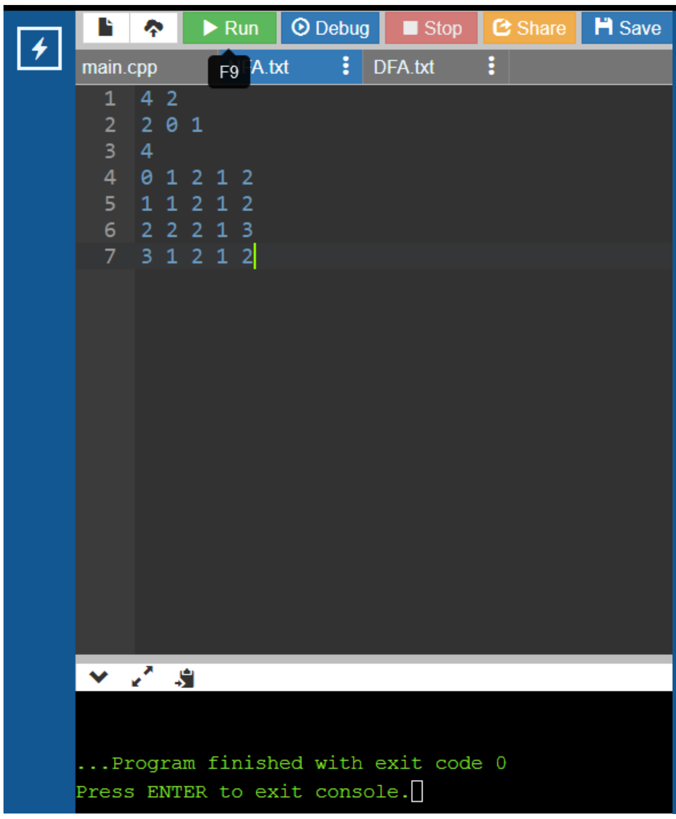
Step 5 : Mark the states of DFA which contains final state of NFA as final states of DFA.

ALGORITHM

1. Find the ε-closure of the opening state of the given NFA.
2. Store that ε-closure as a singular state, which will be the opening state of the DFA.
3. Iterate over all the states that can be reached from the given state one by one for each character in the alphabet.

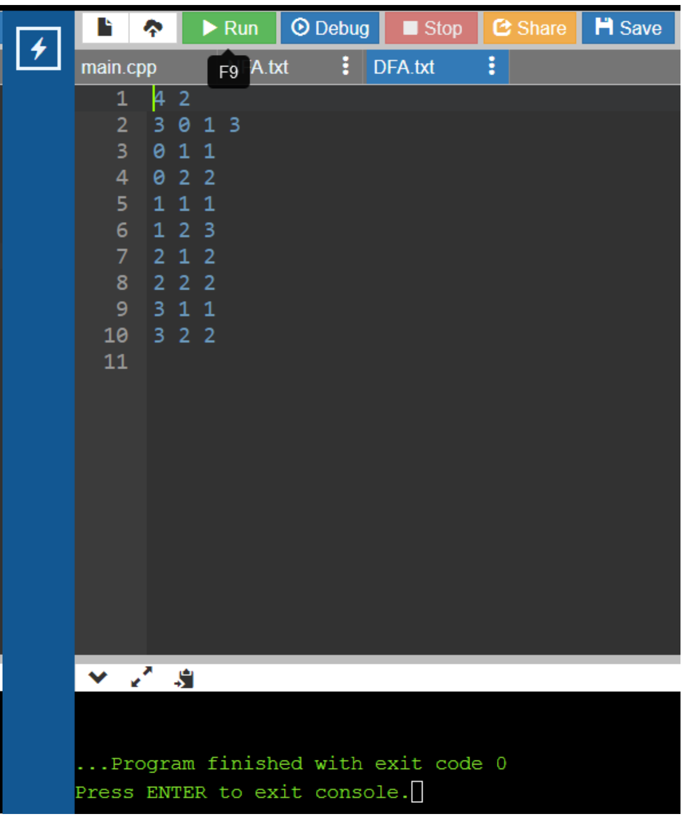
4. Store these state of NFA, along with their ε-closures, that can be reached from the former state through transition due to a specific character.
5. Store all the NFA states found in previous state as a singular state of the DFA which can be reached through transition from the former state due to a specific character.

6. Repeat steps 3-5 for each new state discovered by doing these steps.
    a. Add the state itself to the ε-closure.
    b. Iterate through each ε-transition of the given NFA state and add them to the ε-closure.
    c. Iterate through each ε-transition of each state in the ε-closure and add all ε-reachable states to the ε-closure.
    d. Repeat the above 2 steps for all the states reachable through a finite number of ε-transitions.



```
4 2
2 0 1
4
0 1 2 1 2
1 1 2 1 2
2 2 2 1 3
3 1 2 1 2
```

```
...Program finished with exit code 0
Press ENTER to exit console.
```



```
4 2
3 0 1 3
0 1 1
0 2 2
1 1 1
1 2 3
2 1 2
2 2 2
3 1 1
3 2 2

```

```
...Program finished with exit code 0
Press ENTER to exit console.
```

INPUT OUTPUT

Minimal DFA for any given DFA.

This program that takes a deterministic finite state machine and produces an equivalent one with a minimal number of states.

ALGORITHM

Given a deterministic finite state machine A = (T, Q, R, q0, F), this program constructs an equivalent reduced deterministic finite state machine A' = (T, Q', R', q'0, F') as follows:

- Remove all unreachable states from Q (using DFS).
- Construct the states Q' as a partition of states of Q by successive refinement. Initially let there be two groups, F and Q - F. Sets are represents are bitsets in C for efficiency.
- For each group G in Q', partition G into subgroups such that two states q, r of G are in the same subgroup if and only if for all t in T, states q and r have transitions on t to states in the same group of Q', or both don't have transitions on t.
- Replace group G with its subgroups and repeat last step until no group is further refined.

Constraints

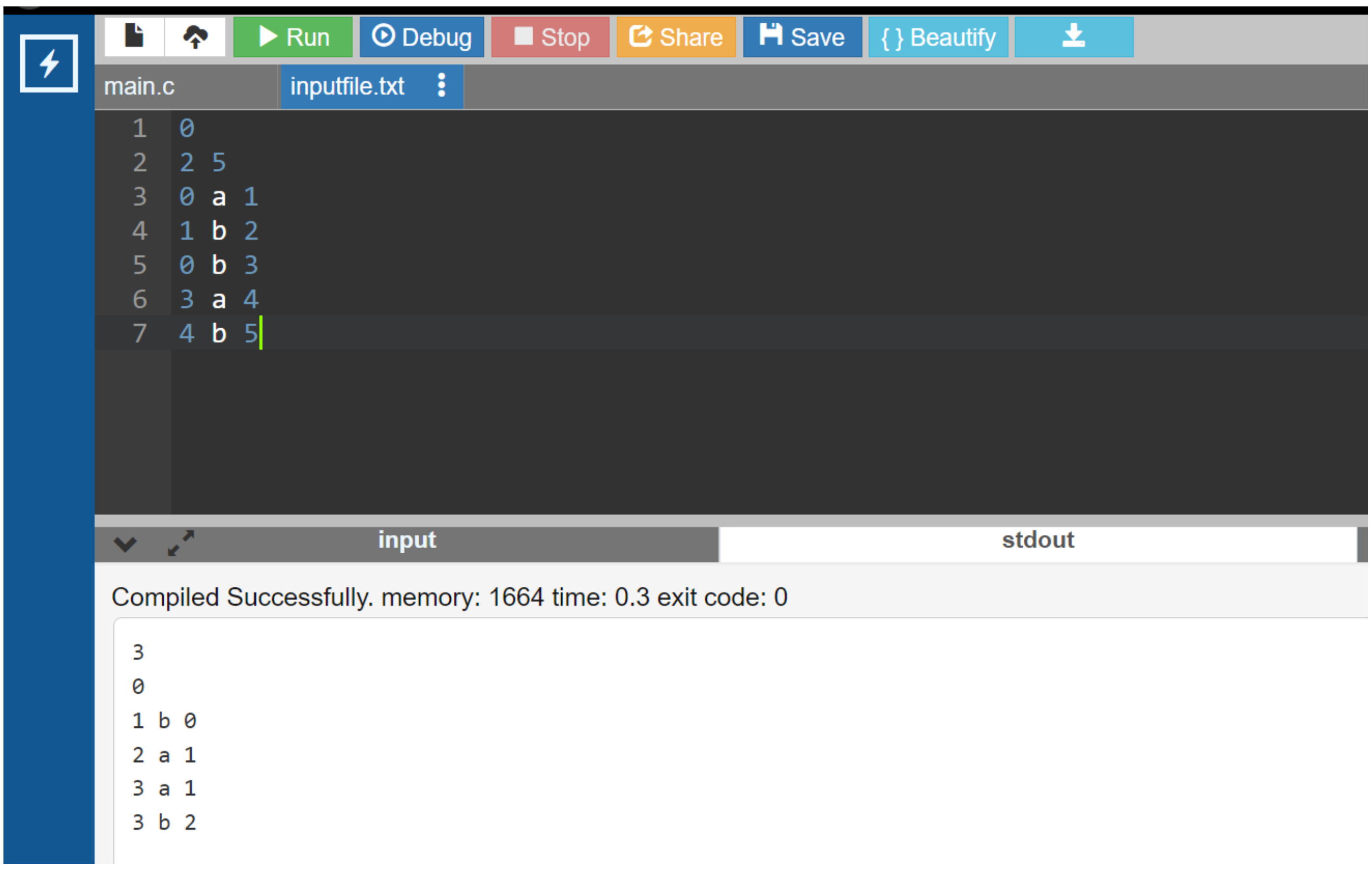
This program can only convert FSMs which have a vocabulary size of less than or equal to 26, and the vocabulary symbols must be translated to the letters a-z. Also, this program can only convert FSMs which have a state size of less or equal to 63, and these state numbers must be transated to 0-63.

Input/Output

The program reads description of a deterministic finite state machine from standard input and produce an equivalent deterministic finite state machine on standard output. The states in the input are represented by integers 0, 1,..., 63 and the symbols by characters a, b, ..., z.

- The first line of the input is the initial state.
- The second line contains the final states, separated by blanks.
- Each subsequent line is a transition asa triple with a source state, a symbol, and a target state, separated by blanks.

The output is analogous. Remember that the state numbers of the reduced FSM have no relevance to the state numbers of the original FSM.



```
0
2 5
0 a 1
1 b 2
0 b 3
3 a 4
4 b 5
```

Compiled Successfully. memory: 1664 time: 0.3 exit code: 0

```
3
0
1 b 0
2 a 1
3 a 1
3 b 2
```

A computer program which reads the rules of Context-free Grammar from a file and remove null production from given grammar.

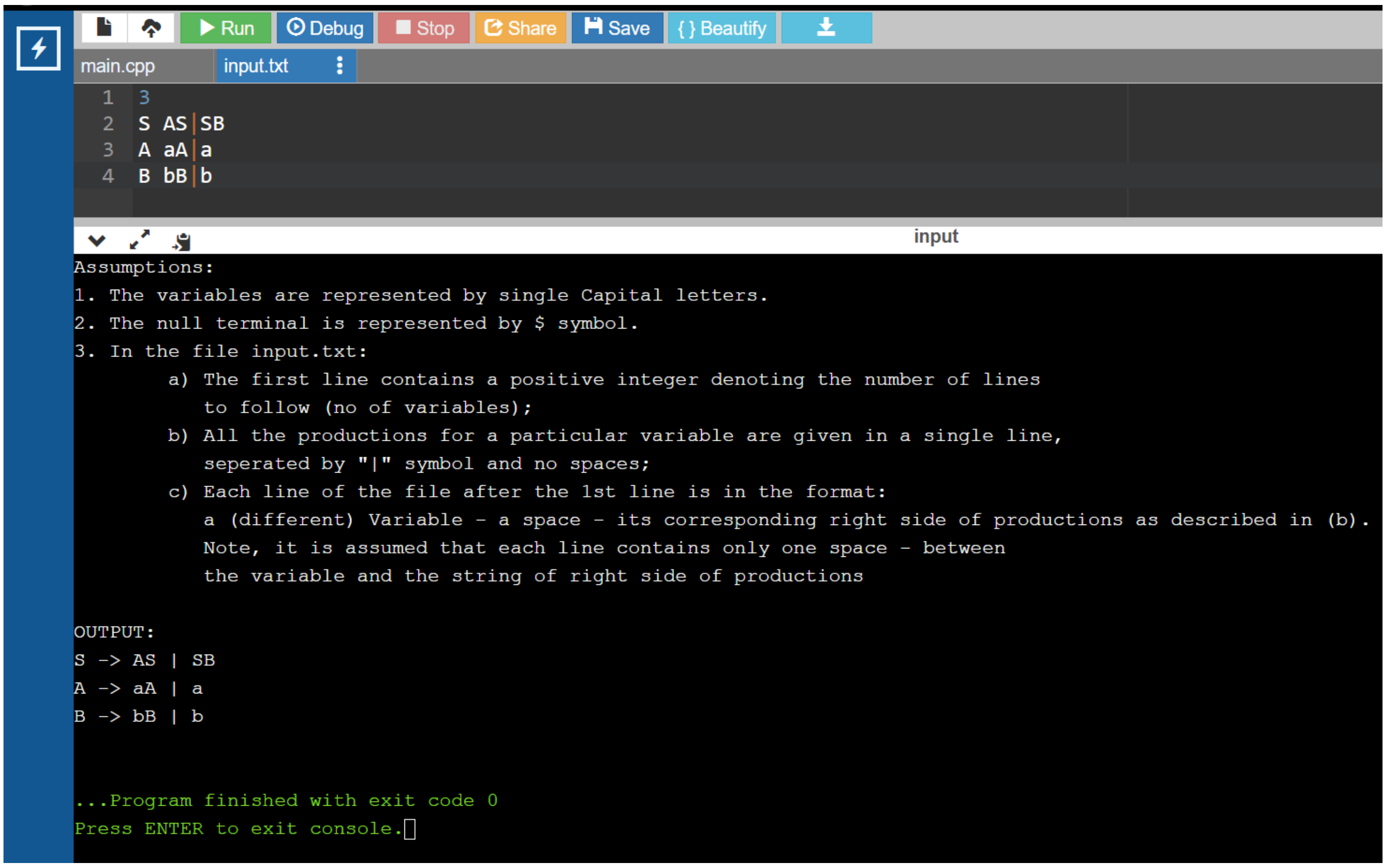
ALGORITHM

1) START
2) Open the file "input.txt" for read.
3) Read the first line and store the integer in variable n. (n denotes the number of variables on left side of some production)
4) Declare a 2d vector of strings, named v, having n no of rows, to store the right side of the productions.

```
5)  Declare a 1d vector of char named variable, to store all the variables on the left side of
    each production.
6)  Declare a queue of char named nv, to store all the nullable variables.
7)  Loop for i from 0 to n-1
        a)  Take p=0
        b)  Read a char and a string from the file and store in variables q and s
            respectively.
        c)  Push q into the vector variable
        d)  Add character '|' at the end of the string s
        e)  Loop for j for all indices of the string s (i.e., 0 to s.size()-1)
            i)  If the character at index j in the string s (i.e., s[j]) is '|', then store the
                substring of s over indices p to j-1 in a string str; and update p = j+1 ▫ If
                str is equal to "$", then push q into the queue nv (as q has a null
                production), else push str into the row i of vector v (i.e., vector v[i])
8)  While nv is not empty, Loop:
        a)  Pop the front character from queue nv, and store in q
        b)  Loop for i from 0 to n-1
            i)  Loop for j for all indices of the vector in row i of v (i.e., j = 0 to
                v[i].size()-1)
                    ❖ Loop for k for all indices of the string v[i][j] (i.e., k = 0 to
                      v[i][j].size()-1)
                          If the character at index k in the string v[i][j] is
                           equal to q (i.e., v[i][j][k] == q), then create a
                           substring from v[i][j] excluding the character at
                           index k, and store in t.
                               o If t is equal to null string (i.e., we get
                               another null production), then push the
                               element at index i of vector variable (i.e.,
                               variable[i]) in the queue nv; else push t into
                               the vector v[i].

9)  To print out the new productions, Loop for i from 0 to n-1
        a)  Erase the repeated strings from the vector v[i]
        b)  Print: " <variable[i]> -> "
        c)  Loop for j from 0 to v[i].size()-2
            i)  Print: " <v[i][j]> | "
        d)  Print: "<v[i][v[i].size() – 1]> <line> "
10) STOP
```

<<CASE 1>>



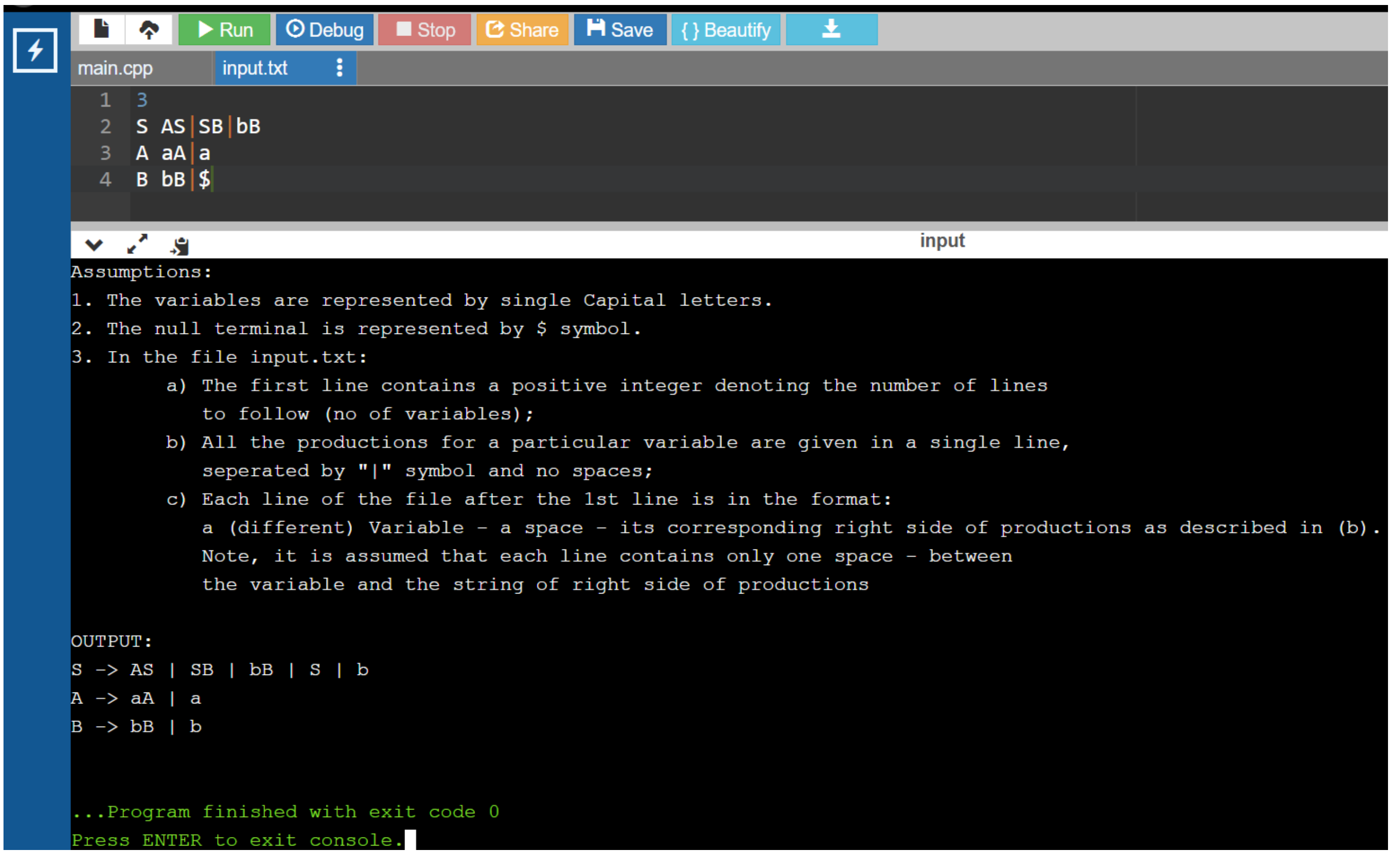
```
3
S AS|SB
A aA|a
B bB|b
```

```
Assumptions:
1. The variables are represented by single Capital letters.
2. The null terminal is represented by $ symbol.
3. In the file input.txt:
        a) The first line contains a positive integer denoting the number of lines
           to follow (no of variables);
        b) All the productions for a particular variable are given in a single line,
           seperated by "|" symbol and no spaces;
        c) Each line of the file after the 1st line is in the format:
           a (different) Variable - a space - its corresponding right side of productions as described in (b).
           Note, it is assumed that each line contains only one space - between
           the variable and the string of right side of productions

OUTPUT:
S -> AS | SB
A -> aA | a
B -> bB | b


...Program finished with exit code 0
Press ENTER to exit console.
```

<<CASE 2>>



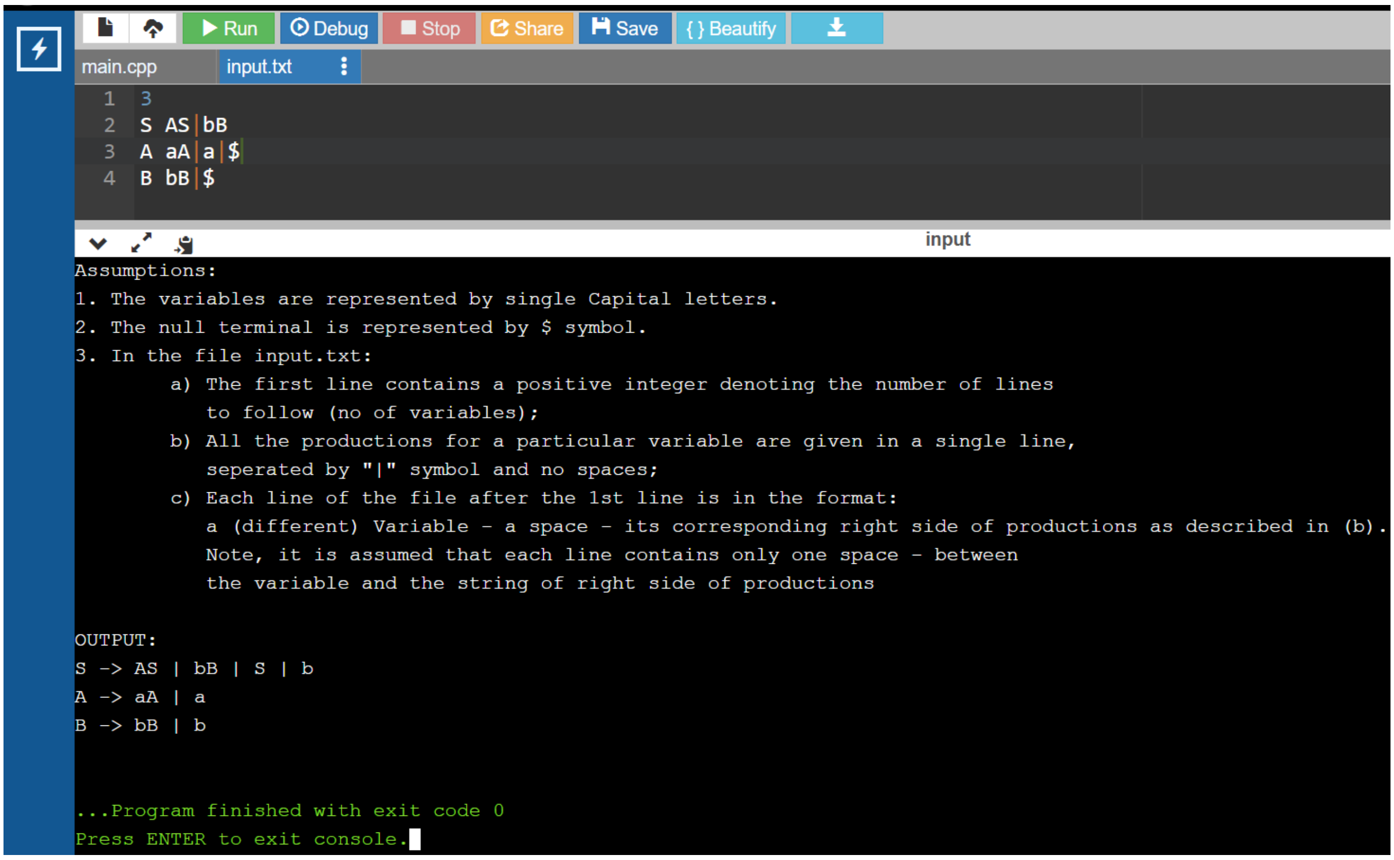
```
3
S AS|SB|bB
A aA|a
B bB|$
```

```
Assumptions:
1. The variables are represented by single Capital letters.
2. The null terminal is represented by $ symbol.
3. In the file input.txt:
        a) The first line contains a positive integer denoting the number of lines
           to follow (no of variables);
        b) All the productions for a particular variable are given in a single line,
           seperated by "|" symbol and no spaces;
        c) Each line of the file after the 1st line is in the format:
           a (different) Variable - a space - its corresponding right side of productions as described in (b).
           Note, it is assumed that each line contains only one space - between
           the variable and the string of right side of productions

OUTPUT:
S -> AS | SB | bB | S | b
A -> aA | a
B -> bB | b


...Program finished with exit code 0
Press ENTER to exit console.
```

<<CASE 3>>



```
3
S AS|bB
A aA|a|$
B bB|$
```

```
Assumptions:
1. The variables are represented by single Capital letters.
2. The null terminal is represented by $ symbol.
3. In the file input.txt:
        a) The first line contains a positive integer denoting the number of lines
           to follow (no of variables);
        b) All the productions for a particular variable are given in a single line,
           seperated by "|" symbol and no spaces;
        c) Each line of the file after the 1st line is in the format:
           a (different) Variable - a space - its corresponding right side of productions as described in (b).
           Note, it is assumed that each line contains only one space - between
           the variable and the string of right side of productions

OUTPUT:
S -> AS | bB | S | b
A -> aA | a
B -> bB | b


...Program finished with exit code 0
Press ENTER to exit console.
```

Simulate Push-down Automata.

ALGORITHM

1) Create a stack structure, which will be used to store a's.

2) Declare integers: n (length of string), flag (whether string is valid or not), c (maintain count of number of a's in stack), d (to make sure string is valid). Initialize flag to 1.

3) Ask the user to enter the string.

4) Initialize integer c to 0 (here we maintain count of number of a's in the stack), initialize d to 0.

5) Start traversing the string.

6) If we get 'a' and int d is zero then we push two 'a' into the stack and increase c by 2.

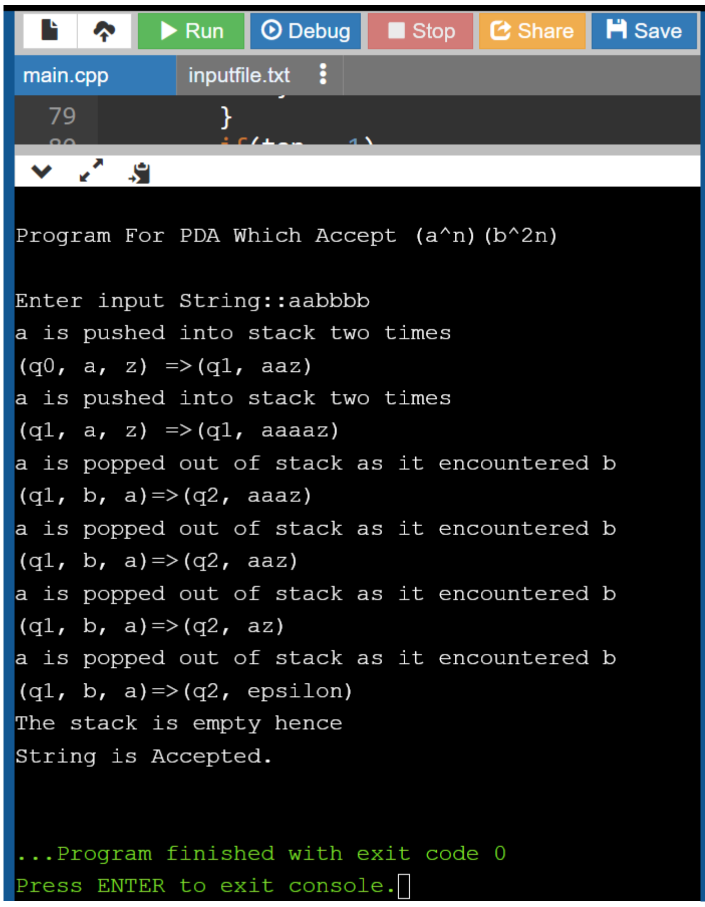
7) Print the transitions by using integer c.

8) If we get 'b' then pop one 'a' from the stack, reduce integer c by 1 and print the transition. Also change value of integer d to 1.

9) If c becomes less than zero, it means stack is empty and we haven't traversed the string completely (string is rejected, change value of integer flag to -1).

10) If we get ‘a’ and integer d is not zero then it means that string is invalid (as ‘a’ comes after ‘b’, change value of integer flag to -1).

11) If at the end the stack is empty and value of integer flag is 1, string is accepted otherwise it is rejected.

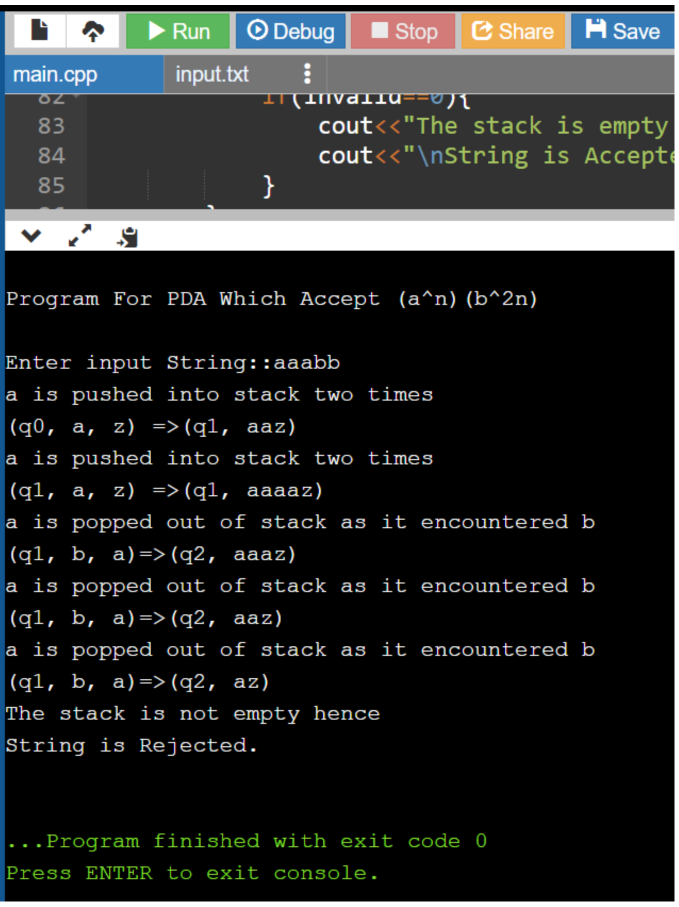


```
            }
```

```
Program For PDA Which Accept (a^n)(b^2n)

Enter input String::aabbbb
a is pushed into stack two times
(q0, a, z) =>(q1, aaz)
a is pushed into stack two times
(q1, a, z) =>(q1, aaaaz)
a is popped out of stack as it encountered b
(q1, b, a)=>(q2, aaaz)
a is popped out of stack as it encountered b
(q1, b, a)=>(q2, aaz)
a is popped out of stack as it encountered b
(q1, b, a)=>(q2, az)
a is popped out of stack as it encountered b
(q1, b, a)=>(q2, epsilon)
The stack is empty hence
String is Accepted.


...Program finished with exit code 0
Press ENTER to exit console.
```



```
            if(invalid==0){
                cout<<"The stack is empty
                cout<<"\nString is Accepte
            }
```

```
Program For PDA Which Accept (a^n)(b^2n)

Enter input String::aaabb
a is pushed into stack two times
(q0, a, z) =>(q1, aaz)
a is pushed into stack two times
(q1, a, z) =>(q1, aaaaz)
a is popped out of stack as it encountered b
(q1, b, a)=>(q2, aaaz)
a is popped out of stack as it encountered b
(q1, b, a)=>(q2, aaz)
a is popped out of stack as it encountered b
(q1, b, a)=>(q2, az)
The stack is not empty hence
String is Rejected.


...Program finished with exit code 0
Press ENTER to exit console.
```

OUTPUT 1
String : aabbbb

OUTPUT 2
String : aaabb