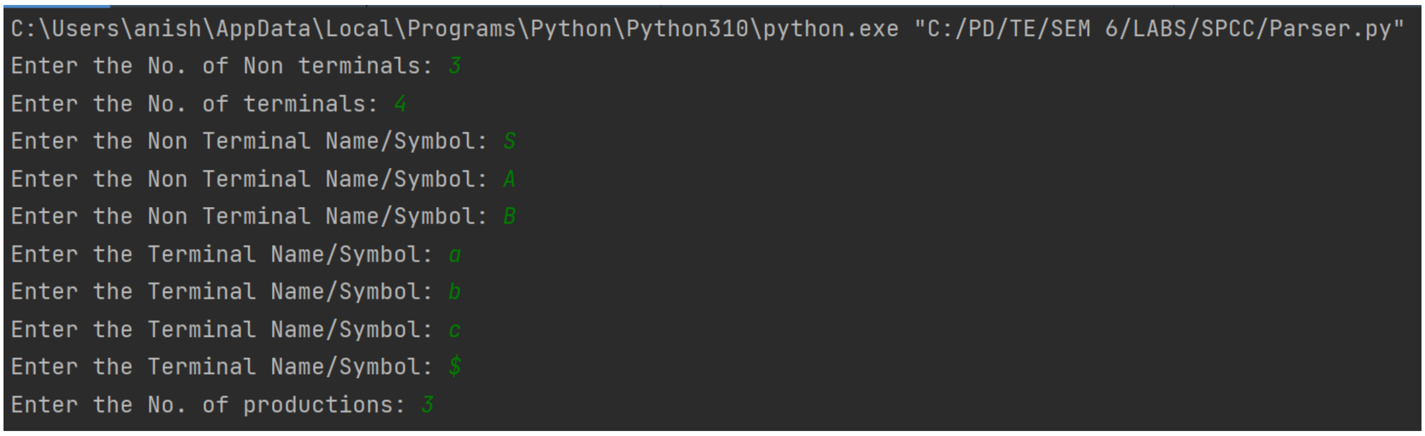
Code:

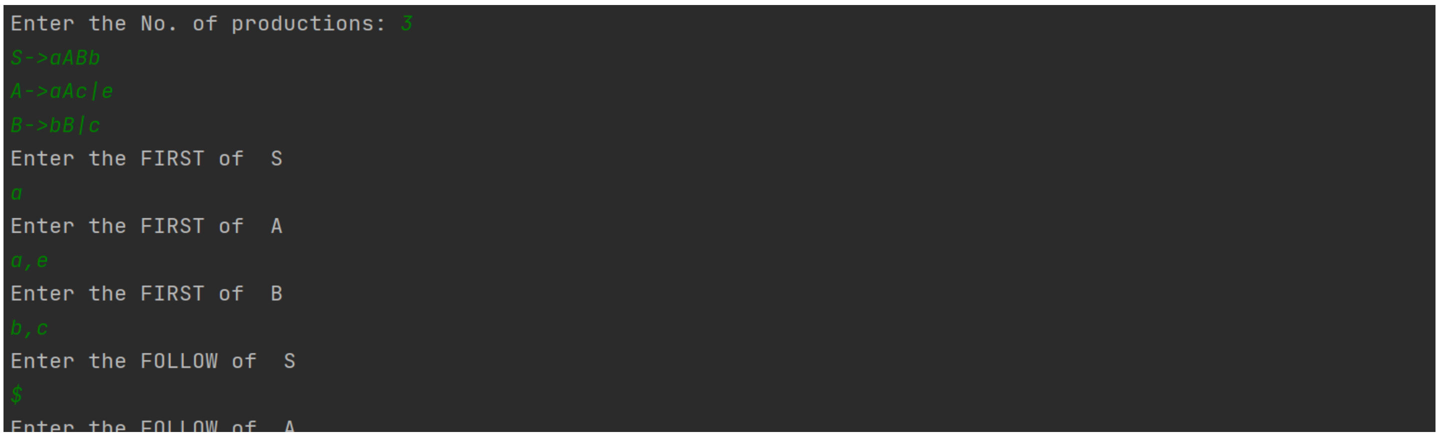
```
from tabulate import tabulate


separator = '->'
epsilon = 'e'

nt = int(input("Enter the No. of Non terminals: "))
t =  int(input("Enter the No. of terminals: "))
# productions = []
# terminals = []
# variables = []

variables = []
terminals = []
for i in range(0, nt):
    variables.append(input("Enter the Non Terminal Name/Symbol: "))
for i in range(0, t):
   terminals.append(input("Enter the Terminal Name/Symbol: "))
p =  int(input("Enter the No. of productions: "))
productions = []

for i in range(0,p):
    productions.append(input())

FIRST = {}
for i in range(0,nt):
    print("Enter the FIRST of ", variables[i])
    FIRST[variables[i]] = input().split(",")

FOLLOW = {}

for i in range(0,nt):
    print("Enter the FOLLOW of ", variables[i])
    FIRST[variables[i]] = input().split(",")

print(FIRST[variables[0]])
# countTerminals = int(input("Enter no. of terminals : "))
# print("Enter terminals : ")
# for i in range(0, countTerminals):
#     terminals.append(input())
# terminals.append('$')

# for i in range(no_of_prod):
    # temp = input(f"Production {i+1} : ").replace(" ", "").split(separator)

parseTable = []
temp = [''] + terminals
parseTable.append(temp)

for j in FIRST.keys():
    row = ['' for x in terminals]
    # print(row)
    row = [j] + row
    # print(row)
    for i in FIRST[j]:
        for k in terminals:
            # print(k, '   ', i)
            if (i == epsilon):
                # print('null', terminals.index(k))
                for z in FOLLOW[j]:
                    if z == k:
                        row[terminals.index(k) + 1] = f'{j} -> {epsilon}'
                        # print(terminals.index(k) + 1)
                        # print(row[terminals.index(k) + 1])
            elif(i == k):
                # print('match', terminals.index(k))
                production = ''
                for prod in productions:
                    if prod.startswith(j):
                        production = prod
                        break
                temp = production.split(separator)[1].split('|')
                # print(temp)
                production = temp[0]
                for x in temp:
                    if x.startswith(i):
                        production = x
                # print("I am using FISRT")
                # print(terminals.index(k) + 1)
                row[terminals.index(k) + 1] = f'{j} -> {production}'
    parseTable.append(row)

print(tabulate(parseTable, tablefmt="simple_grid"))


def validateStringUsingStackBuffer(parsing_table, grammarll1,table_term_list, input_string,term_userdef,start_symbol):
  print(f"\nValidate String => {input_string}\n")
  if grammarll1 == False:
    return f"\nInput String = " \
      f"\"{input_string}\"\n" \
      f"Grammar is not LL(1)"
  stack = [start_symbol, '$']
  buffer = []
  input_string = input_string.split()
  input_string.reverse()
  buffer = ['$'] + input_string
  print("{:>20} {:>20} {:>20}".
    format("Buffer", "Stack","Action"))
  while True:
    if stack == ['$'] and buffer == ['$']:
      print("{:>20} {:>20} {:>20}".format(' '.join(buffer),' '.join(stack),"Valid"))
      return "\nValid String!"
    elif stack[0] not in term_userdef:
      x = list(diction.keys()).index(stack[0])
      y = table_term_list.index(buffer[-1])
```

```
        if parsing_table[x][y] != '':
            entry = parsing_table[x][y]
            print("{:>20} {:>20} {:>25}".format(' '.join(buffer),' '.join(stack),f"T[{stack[0]}][{buffer[-1]}] = {entry}"))
            lhs_rhs = entry.split("->")
            lhs_rhs[1] = lhs_rhs[1].replace('#', '').strip()
            entryrhs = lhs_rhs[1].split()
            stack = entryrhs + stack[1:]
        else:
            return f"\nInvalid String! No rule at " \
                f"Table[{stack[0]}][{buffer[-1]}]."
    else:
        if stack[0] == buffer[-1]:
            print("{:>20} {:>20} {:>20}".format(' '.join(buffer),' '.join(stack),f"Matched:{stack[0]}"))
            buffer = buffer[:-1]
            stack = stack[1:]
        else:
            return "\nInvalid String! " \
                "Unmatched terminal symbols"
```

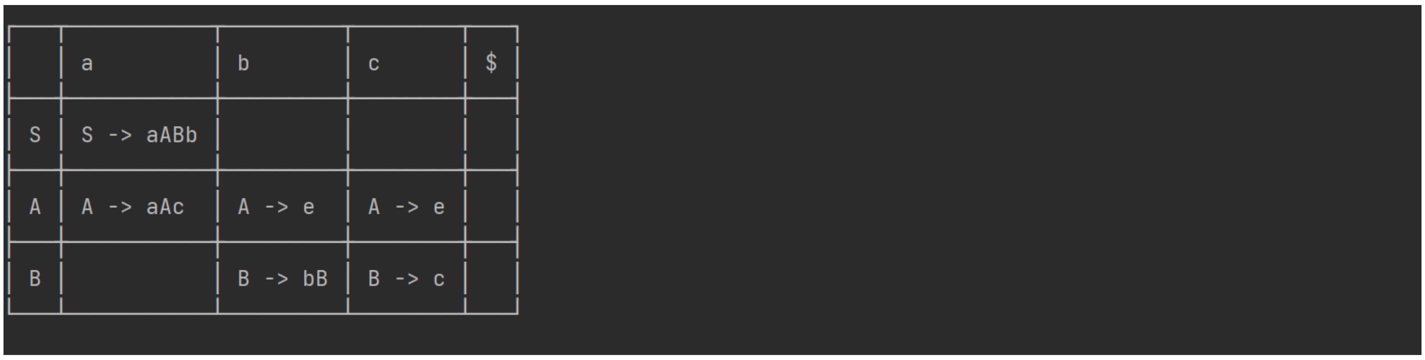
Output:



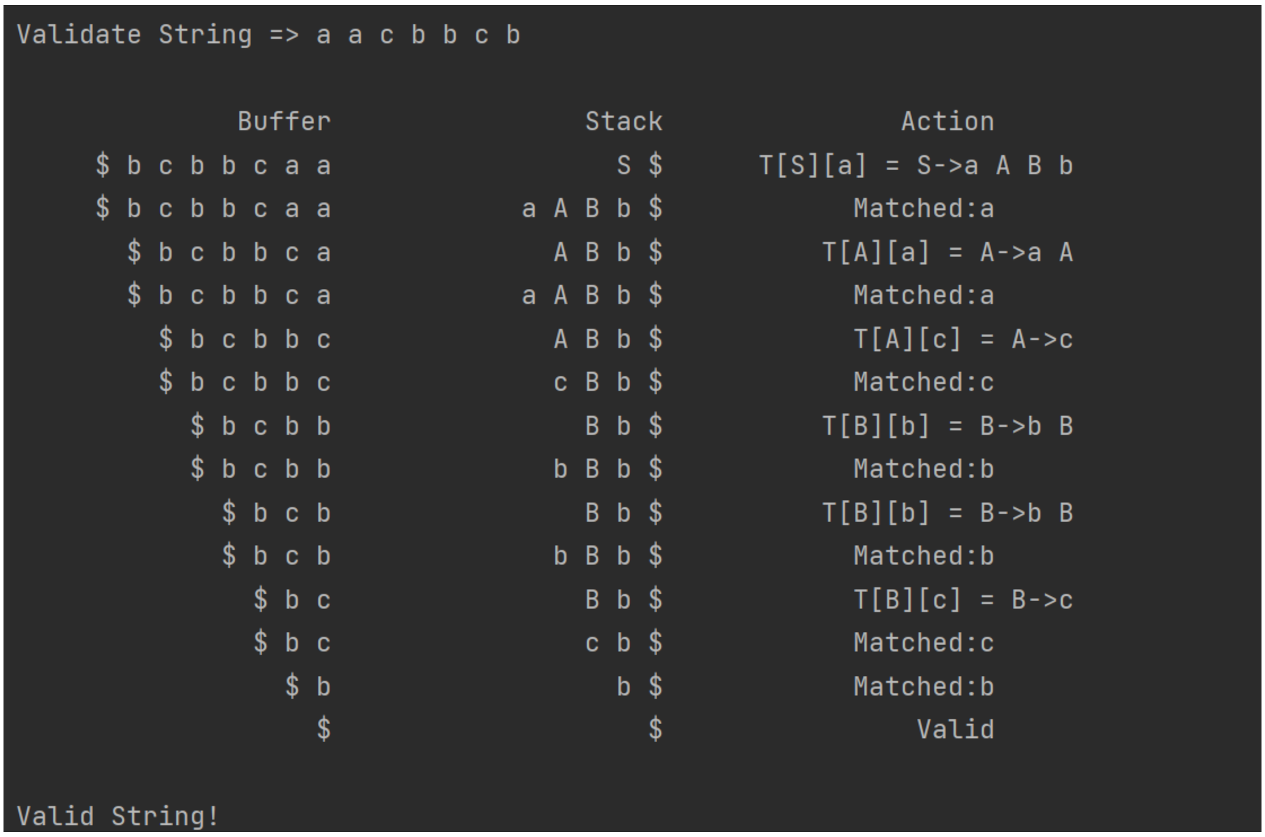
```
C:\Users\anish\AppData\Local\Programs\Python\Python310\python.exe "C:/PD/TE/SEM 6/LABS/SPCC/Parser.py"
Enter the No. of Non terminals: 3
Enter the No. of terminals: 4
Enter the Non Terminal Name/Symbol: S
Enter the Non Terminal Name/Symbol: A
Enter the Non Terminal Name/Symbol: B
Enter the Terminal Name/Symbol: a
Enter the Terminal Name/Symbol: b
Enter the Terminal Name/Symbol: c
Enter the Terminal Name/Symbol: $
Enter the No. of productions: 3
```



```
Enter the No. of productions: 3
S->aABb
A->aAc|e
B->bB|c
Enter the FIRST of  S
a
Enter the FIRST of  A
a,e
Enter the FIRST of  B
b,c
Enter the FOLLOW of  S
$
Enter the FOLLOW of  A
```



```
+---+-------------+----------+----------+---+
|   | a           | b        | c        | $ |
+---+-------------+----------+----------+---+
| S | S -> aABb   |          |          |   |
+---+-------------+----------+----------+---+
| A | A -> aAc    | A -> e   | A -> e   |   |
+---+-------------+----------+----------+---+
| B |             | B -> bB  | B -> c   |   |
+---+-------------+----------+----------+---+
```



```
Validate String => a a c b b c b

              Buffer                  Stack                 Action
    $ b c b b c a a                     S $       T[S][a] = S->a A B b
    $ b c b b c a a               a A B b $             Matched:a
      $ b c b b c a                 A B b $         T[A][a] = A->a A
      $ b c b b c a               a A B b $             Matched:a
        $ b c b b c                 A B b $             T[A][c] = A->c
        $ b c b b c               c B b $             Matched:c
          $ b c b b                 B b $         T[B][b] = B->b B
          $ b c b b               b B b $             Matched:b
            $ b c b                 B b $         T[B][b] = B->b B
            $ b c b               b B b $             Matched:b
              $ b c                 B b $             T[B][c] = B->c
              $ b c                 c b $             Matched:c
                $ b                   b $             Matched:b
                  $                     $                 Valid

Valid String!
```