18CSC304J/ Compiler Design

Submitted By:- ANANNYA P. NEOG (RA1911003010367)

Exp-5: FIRST AND FOLLOW computation

Aim:- To write a program to perform FIRST AND FOLLOW computation

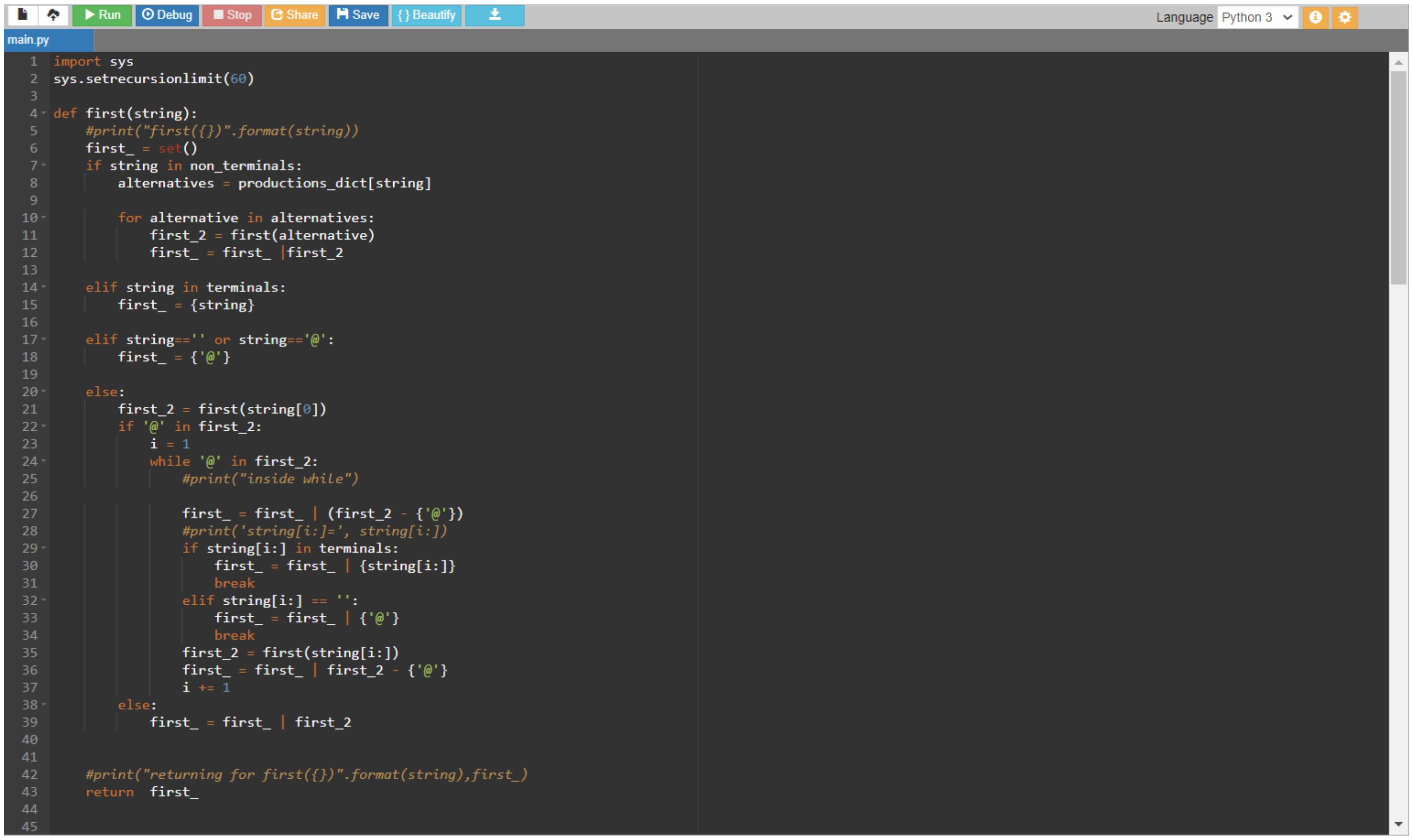
Code:-

```
import sys
sys.setrecursionlimit(60)

def first(string):
   #print("first({})".format(string))
   first_ = set()
   if string in non_terminals:
      alternatives = productions_dict[string]

      for alternative in alternatives:
         first_2 = first(alternative)
         first_ = first_ |first_2

   elif string in terminals:
      first_ = {string}

   elif string=='' or string=='@':
      first_ = {'@'}

   else:
      first_2 = first(string[0])
      if '@' in first_2:
         i = 1
         while '@' in first_2:
            #print("inside while")

            first_ = first_ | (first_2 - {'@'})
            #print('string[i:]=', string[i:])
            if string[i:] in terminals:
               first_ = first_ | {string[i:]}
               break
            elif string[i:] == '':
               first_ = first_ | {'@'}
               break
            first_2 = first(string[i:])
            first_ = first_ | first_2 - {'@'}
            i += 1
      else:
         first_ = first_ | first_2


   #print("returning for first({})".format(string),first_)
   return  first_
```

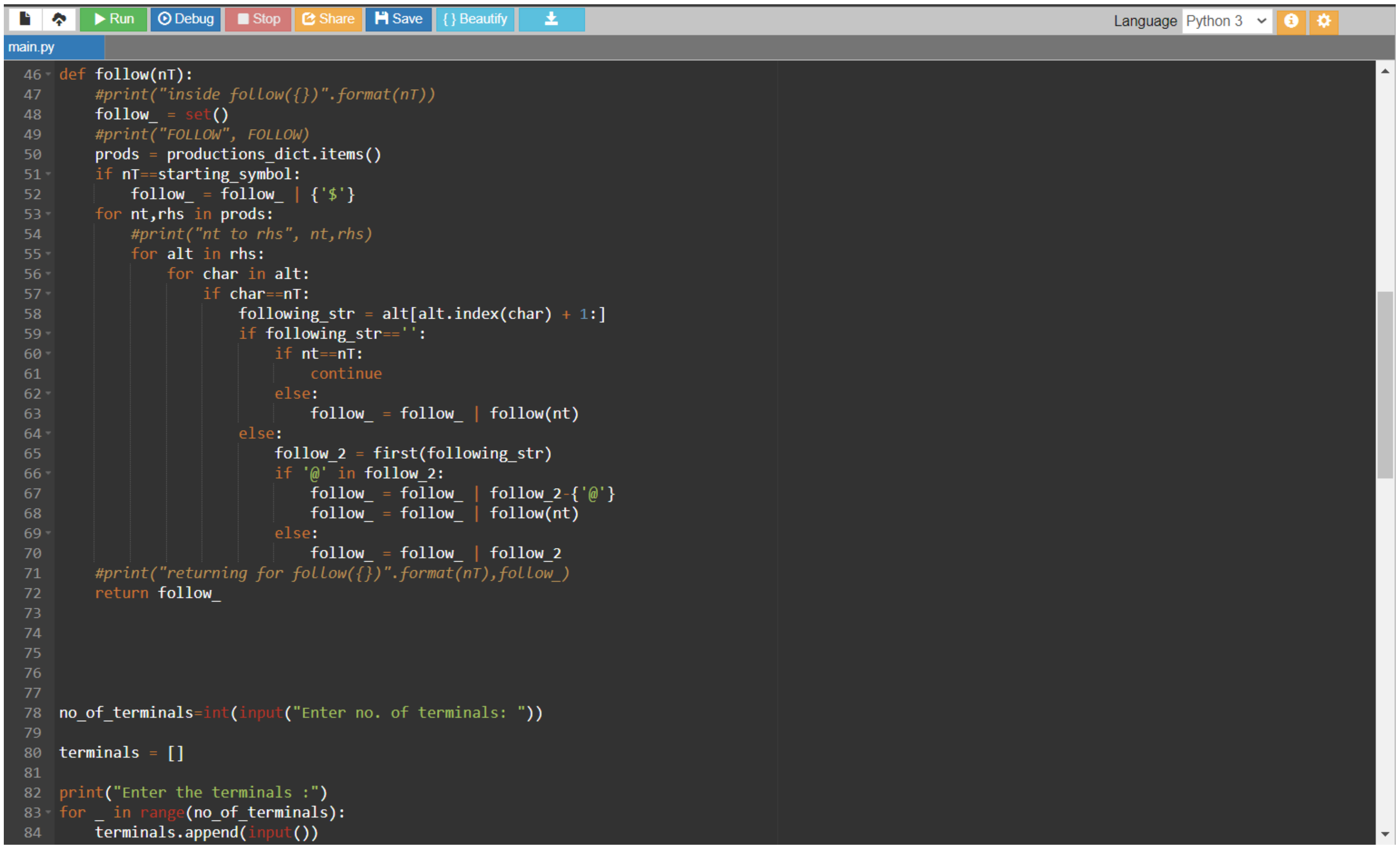
```
def follow(nT):
    #print("inside follow({})".format(nT))
    follow_ = set()
    #print("FOLLOW", FOLLOW)
    prods = productions_dict.items()
    if nT==starting_symbol:
        follow_ = follow_ | {'$'}
    for nt,rhs in prods:
        #print("nt to rhs", nt,rhs)
        for alt in rhs:
            for char in alt:
                if char==nT:
                    following_str = alt[alt.index(char) + 1:]
                    if following_str=='':
                        if nt==nT:
                            continue
                        else:
                            follow_ = follow_ | follow(nt)
                    else:
                        follow_2 = first(following_str)
                        if '@' in follow_2:
                            follow_ = follow_ | follow_2-{'@'}
                            follow_ = follow_ | follow(nt)
                        else:
                            follow_ = follow_ | follow_2
    #print("returning for follow({})".format(nT),follow_)
    return follow_



no_of_terminals=int(input("Enter no. of terminals: "))

terminals = []

print("Enter the terminals :")
for _ in range(no_of_terminals):
    terminals.append(input())

no_of_non_terminals=int(input("Enter no. of non terminals: "))

non_terminals = []

print("Enter the non terminals :")
for _ in range(no_of_non_terminals):
    non_terminals.append(input())

starting_symbol = input("Enter the starting symbol: ")

no_of_productions = int(input("Enter no of productions: "))

productions = []
```

```
print("Enter the productions:")
for _ in range(no_of_productions):
    productions.append(input())


#print("terminals", terminals)

#print("non terminals", non_terminals)

#print("productions",productions)


productions_dict = {}

for nT in non_terminals:
    productions_dict[nT] = []


#print("productions_dict",productions_dict)

for production in productions:
    nonterm_to_prod = production.split("->")
    alternatives = nonterm_to_prod[1].split("/")
    for alternative in alternatives:
        productions_dict[nonterm_to_prod[0]].append(alternative)

#print("productions_dict",productions_dict)

#print("nonterm_to_prod",nonterm_to_prod)
#print("alternatives",alternatives)


FIRST = {}
FOLLOW = {}

for non_terminal in non_terminals:
    FIRST[non_terminal] = set()

for non_terminal in non_terminals:
    FOLLOW[non_terminal] = set()

#print("FIRST",FIRST)

for non_terminal in non_terminals:
    FIRST[non_terminal] = FIRST[non_terminal] | first(non_terminal)

#print("FIRST",FIRST)


FOLLOW[starting_symbol] = FOLLOW[starting_symbol] | {'$'}
for non_terminal in non_terminals:
    FOLLOW[non_terminal] = FOLLOW[non_terminal] | follow(non_terminal)

#print("FOLLOW", FOLLOW)
```

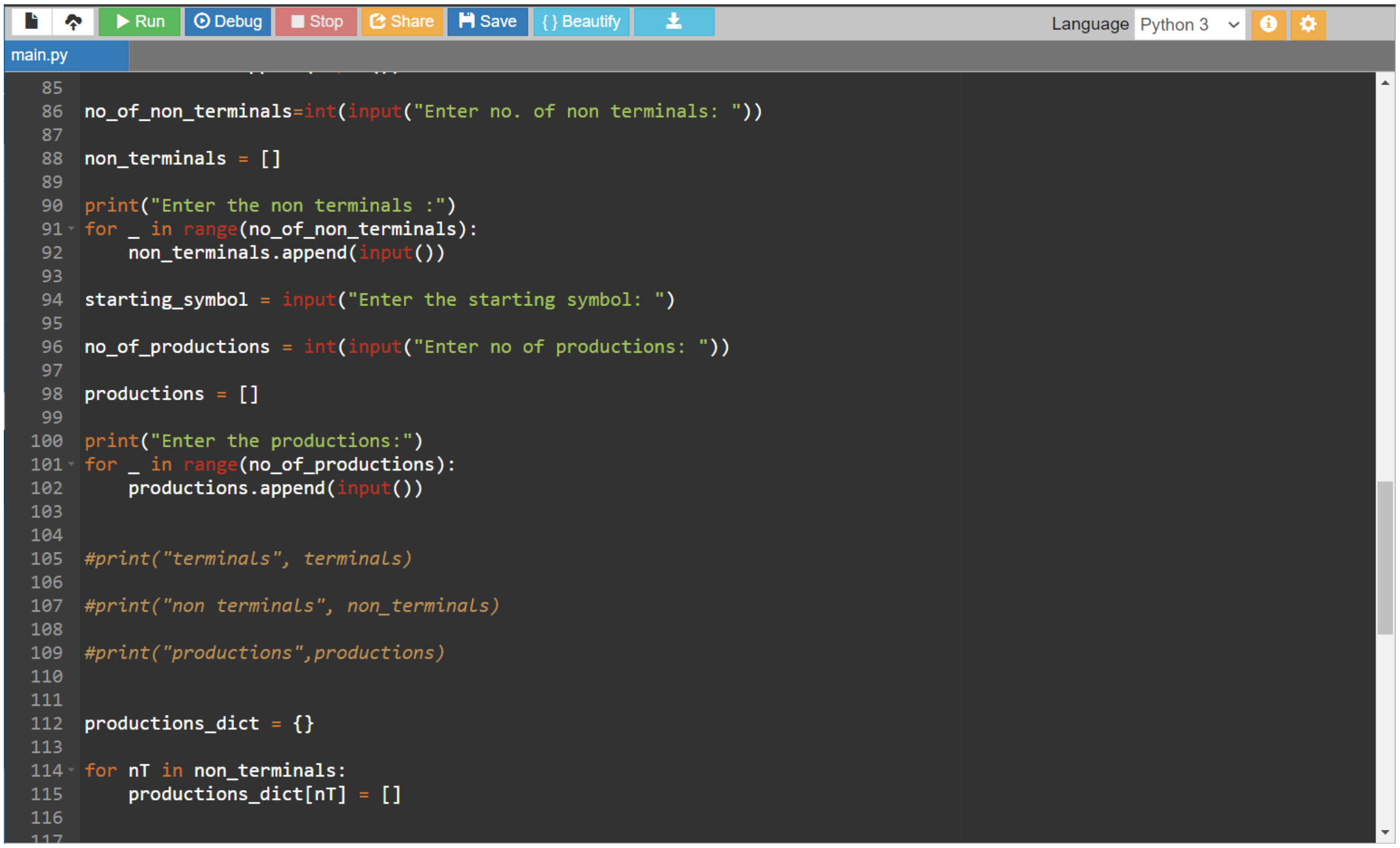
```
print("{: ^20}{: ^20}{: ^20}".format('Non Terminals','First','Follow'))
for non_terminal in non_terminals:
   print("{: ^20}{: ^20}{: ^20}".format(non_terminal,str(FIRST[non_terminal]),str(FOLLOW[non_terminal])))
```



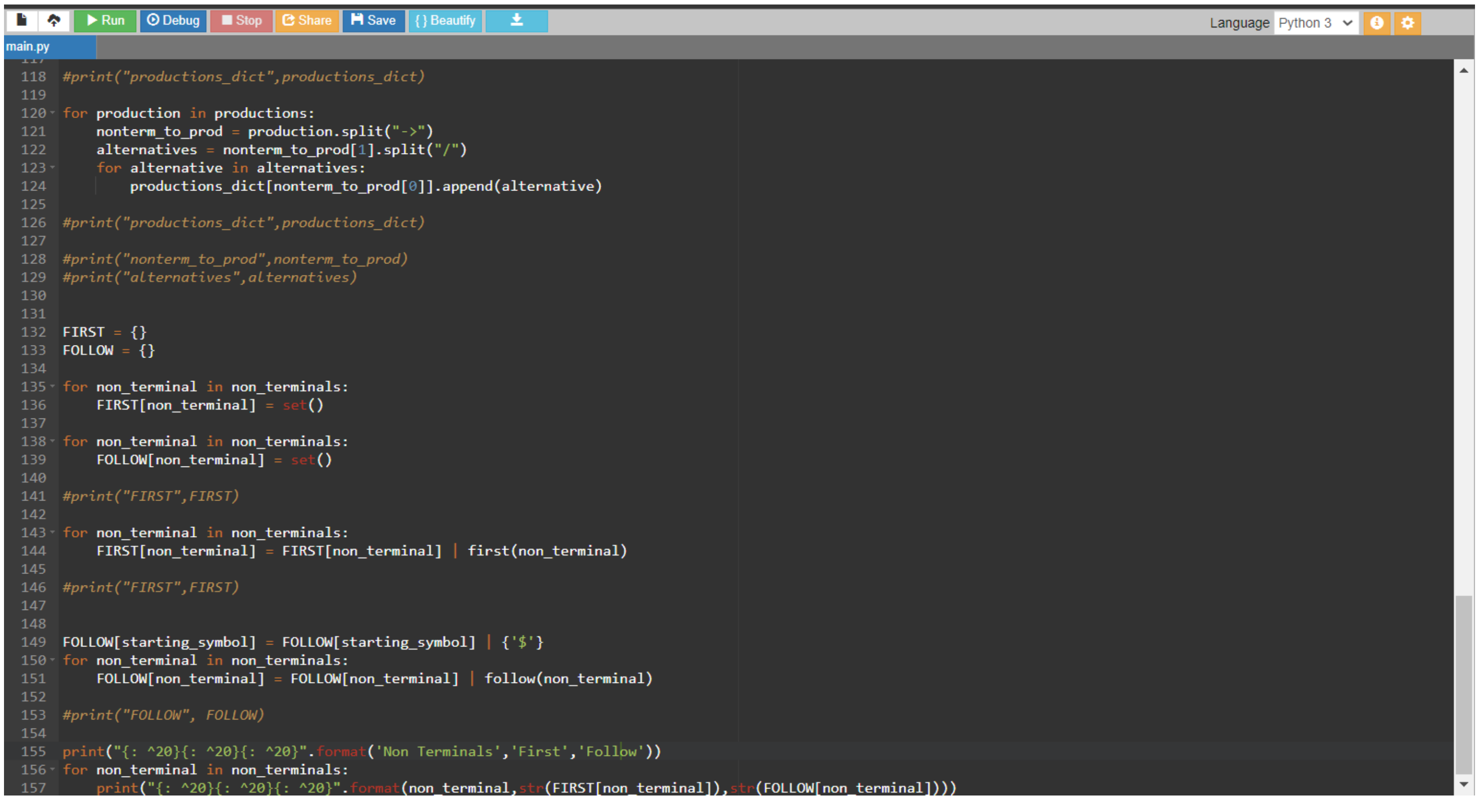
```
import sys
sys.setrecursionlimit(60)

def first(string):
    #print("first({})".format(string))
    first_ = set()
    if string in non_terminals:
        alternatives = productions_dict[string]

        for alternative in alternatives:
            first_2 = first(alternative)
            first_ = first_ |first_2

    elif string in terminals:
        first_ = {string}

    elif string=='' or string=='@':
        first_ = {'@'}

    else:
        first_2 = first(string[0])
        if '@' in first_2:
            i = 1
            while '@' in first_2:
                #print("inside while")

                first_ = first_ | (first_2 - {'@'})
                #print('string[i:]=', string[i:])
                if string[i:] in terminals:
                    first_ = first_ | {string[i:]}
                    break
                elif string[i:] == '':
                    first_ = first_ | {'@'}
                    break
                first_2 = first(string[i:])
                first_ = first_ | first_2 - {'@'}
                i += 1
        else:
            first_ = first_ | first_2


    #print("returning for first({})".format(string),first_)
    return  first_


```



```
def follow(nT):
    #print("inside follow({})".format(nT))
    follow_ = set()
    #print("FOLLOW", FOLLOW)
    prods = productions_dict.items()
    if nT==starting_symbol:
        follow_ = follow_ | {'$'}
    for nt,rhs in prods:
        #print("nt to rhs", nt,rhs)
        for alt in rhs:
            for char in alt:
                if char==nT:
                    following_str = alt[alt.index(char) + 1:]
                    if following_str=='':
                        if nt==nT:
                            continue
                        else:
                            follow_ = follow_ | follow(nt)
                    else:
                        follow_2 = first(following_str)
                        if '@' in follow_2:
                            follow_ = follow_ | follow_2-{'@'}
                            follow_ = follow_ | follow(nt)
                        else:
                            follow_ = follow_ | follow_2
    #print("returning for follow({})".format(nT),follow_)
    return follow_





no_of_terminals=int(input("Enter no. of terminals: "))

terminals = []

print("Enter the terminals :")
for _ in range(no_of_terminals):
    terminals.append(input())
```



```python

no_of_non_terminals=int(input("Enter no. of non terminals: "))

non_terminals = []

print("Enter the non terminals :")
for _ in range(no_of_non_terminals):
    non_terminals.append(input())

starting_symbol = input("Enter the starting symbol: ")

no_of_productions = int(input("Enter no of productions: "))

productions = []

print("Enter the productions:")
for _ in range(no_of_productions):
    productions.append(input())


#print("terminals", terminals)

#print("non terminals", non_terminals)

#print("productions",productions)


productions_dict = {}

for nT in non_terminals:
    productions_dict[nT] = []

```



```python
#print("productions_dict",productions_dict)

for production in productions:
    nonterm_to_prod = production.split("->")
    alternatives = nonterm_to_prod[1].split("/")
    for alternative in alternatives:
        productions_dict[nonterm_to_prod[0]].append(alternative)

#print("productions_dict",productions_dict)

#print("nonterm_to_prod",nonterm_to_prod)
#print("alternatives",alternatives)


FIRST = {}
FOLLOW = {}

for non_terminal in non_terminals:
    FIRST[non_terminal] = set()

for non_terminal in non_terminals:
    FOLLOW[non_terminal] = set()

#print("FIRST",FIRST)

for non_terminal in non_terminals:
    FIRST[non_terminal] = FIRST[non_terminal] | first(non_terminal)

#print("FIRST",FIRST)


FOLLOW[starting_symbol] = FOLLOW[starting_symbol] | {'$'}
for non_terminal in non_terminals:
    FOLLOW[non_terminal] = FOLLOW[non_terminal] | follow(non_terminal)

#print("FOLLOW", FOLLOW)

print("{: ^20}{: ^20}{: ^20}".format('Non Terminals','First','Follow'))
for non_terminal in non_terminals:
    print("{: ^20}{: ^20}{: ^20}".format(non_terminal,str(FIRST[non_terminal]),str(FOLLOW[non_terminal])))
```

(Kindly refer the output screenshot for the inputs)

Input 1:

2

x

y

3

X

Y

Z

X

4

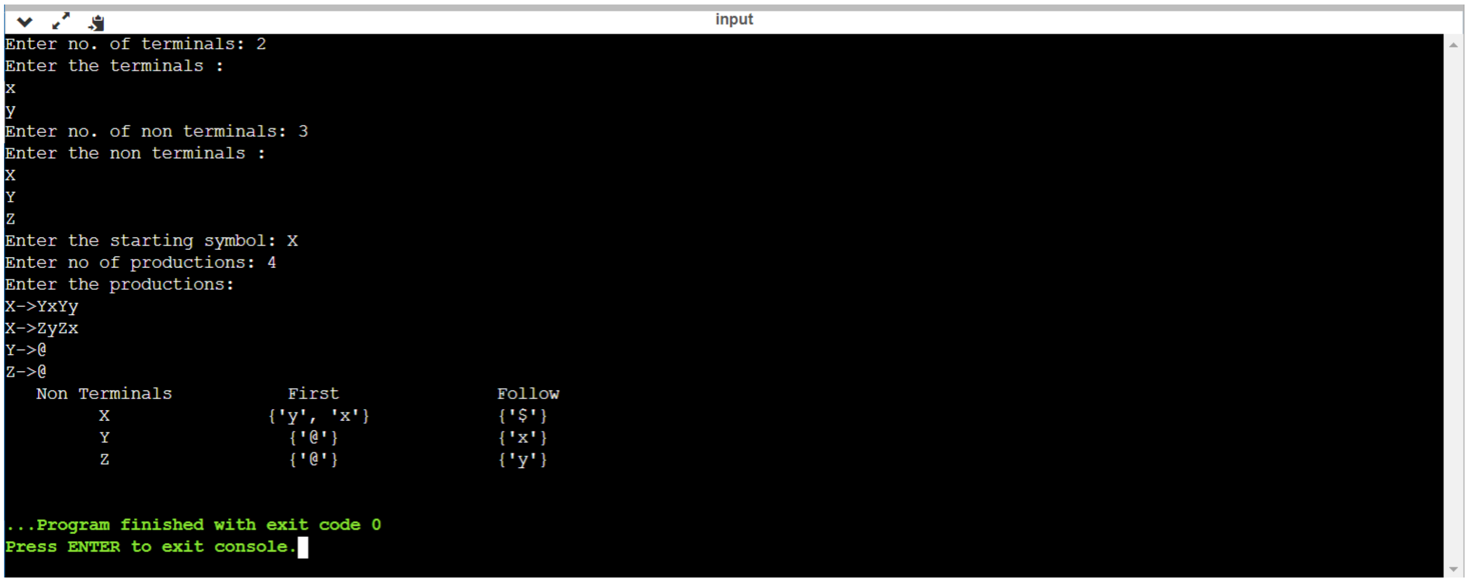
X->YxYy

X->ZyZx

Y->@

Z->@

Output 1:

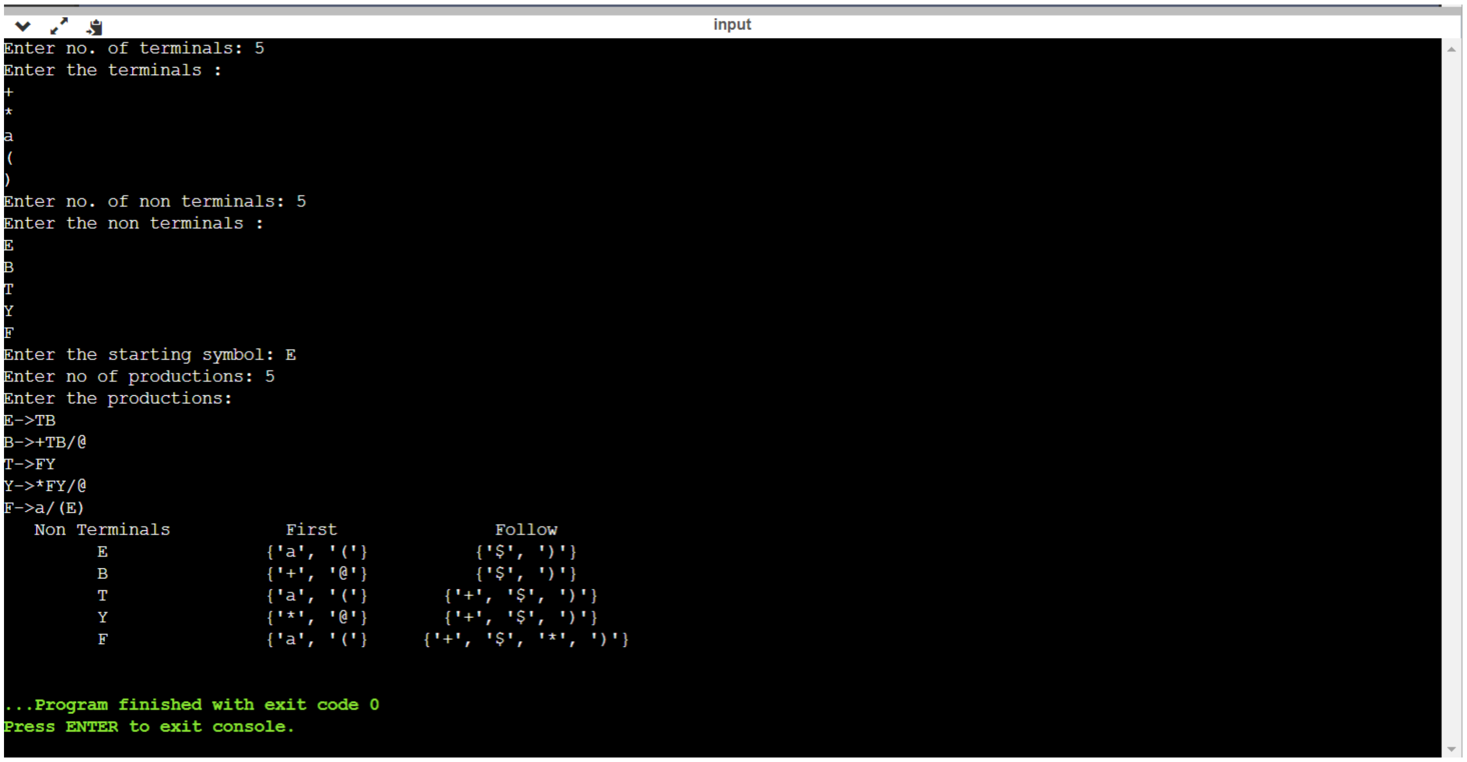


```
Enter no. of terminals: 2
Enter the terminals :
x
y
Enter no. of non terminals: 3
Enter the non terminals :
X
Y
Z
Enter the starting symbol: X
Enter no of productions: 4
Enter the productions:
X->YxYy
X->ZyZx
Y->@
Z->@
   Non Terminals           First               Follow
         X              {'y', 'x'}             {'$'}
         Y                {'@'}                {'x'}
         Z                {'@'}                {'y'}


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

Input 2:

```
5
+
*
a
(
)

5
E
B
T
Y
F

E
5
E->TB
B->+TB/@
T->FY
Y->*FY/@
F->a/(E)
```

Output 2:



```
Enter no. of terminals: 5
Enter the terminals :
+
*
a
(
)
Enter no. of non terminals: 5
Enter the non terminals :
E
B
T
Y
F
Enter the starting symbol: E
Enter no of productions: 5
Enter the productions:
E->TB
B->+TB/@
T->FY
Y->*FY/@
F->a/(E)
   Non Terminals           First               Follow
         E               {'a', '('}          {'$', ')'}
         B               {'+', '@'}          {'$', ')'}
         T               {'a', '('}       {'+', '$', ')'}
         Y               {'*', '@'}       {'+', '$', ')'}
         F               {'a', '('}     {'+', '$', '*', ')'}


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

Result:- The FIRST and FOLLOW sets of non-terminals of a grammar were found successfully.