**Name**: Shubham Lad

**Class**: MSc Computer Science (Part 1)

**Roll No**: 512

**Semester**: II

**Subject**: Design and Implementation of Modern Compilers

**Topic**: First & Follow

**Aim**: Python code for finding First and Follow

## Introduction:

FIRST and FOLLOW are two functions associated with grammar that help us fill in the entries of an M-table.

FIRST(): It is a function that gives the set of terminals that begin the strings derived from the production rule.

A symbol c is in FIRST (α) if and only if α ⇒ cβ for some sequence β of grammar symbols.

A terminal symbol a is in FOLLOW (N) if and only if there is a derivation from the start symbol S of the grammar such that S ⇒ αNαβ, where α and β are a (possible empty) sequence of grammar symbols. In other words, a terminal c is in FOLLOW (N) if c can follow N at some point in a derivation.

## Benefit of FIRST( ) and FOLLOW( )

- It can be used to prove the LL (K) characteristic of grammar.
- It can be used to promote in the construction of predictive parsing tables.
- It provides selection information for recursive descent parsers.

## Computation of *FIRST*

*FIRST (α) is defined as the collection of terminal symbols which are the first letters of strings derived from α.*

FIRST (α) = {α |α →∗ αβ for some string β}

If X is Grammar Symbol, then First (X) will be:

- If X is a terminal symbol, then FIRST(X) = {X}
- If X → ε, then FIRST(X) = {ε}
- If X is non-terminal & X → a α, then FIRST (X) = {a}
- If X → Y1, Y2, Y3, then FIRST(X) will be:
    a) If Y is terminal, then FIRST(X)=FIRST (Y1,Y2,Y3)={Y1}
    b) If Y1 is non-terminal and if Y1 does not derive to an empty string i.e., if FIRST(Y1) does not contain ε then, FIRST(X)=FIRST(Y1,Y2,Y3)=FIRST(Y1)

c) If FIRST (Y1) contains ε, then,
FIRST(X)=FIRST(Y1,Y2,Y3)=FIRST(Y1)−{ε}∪ FIRST(Y2, Y3)

Similarly, FIRST (Y2, Y3) = {Y2}, If Y2 is terminal otherwise if Y2 is non-terminal then

- FIRST (Y2, Y3) = FIRST (Y2), if FIRST (Y2) does not contain ε.
- If FIRST (Y2) contain ε, then FIRST (Y2, Y3) = FIRST (Y2) − {ε} ∪ FIRST (Y3)

## Computation of *FOLLOW*

*Follow (A) is defined as the collection of terminal symbols that occur directly to the right of A.*

FOLLOW(A) = {a|S ⇒* αAaβ where α, β can be any strings}

- If S is the start symbol, FOLLOW (S) ={$}
- If production is of form A → α B β, β ≠ ε.

a) If FIRST (β) does not contain ε then, FOLLOW (B) = {FIRST (β)}

Or

b) If FIRST (β) contains ε (i. e. , β ⇒* ε), then
FOLLOW (B) = FIRST (β) − {ε} ∪ FOLLOW (A)

ie. when β derives ε, then terminal after A will follow B.

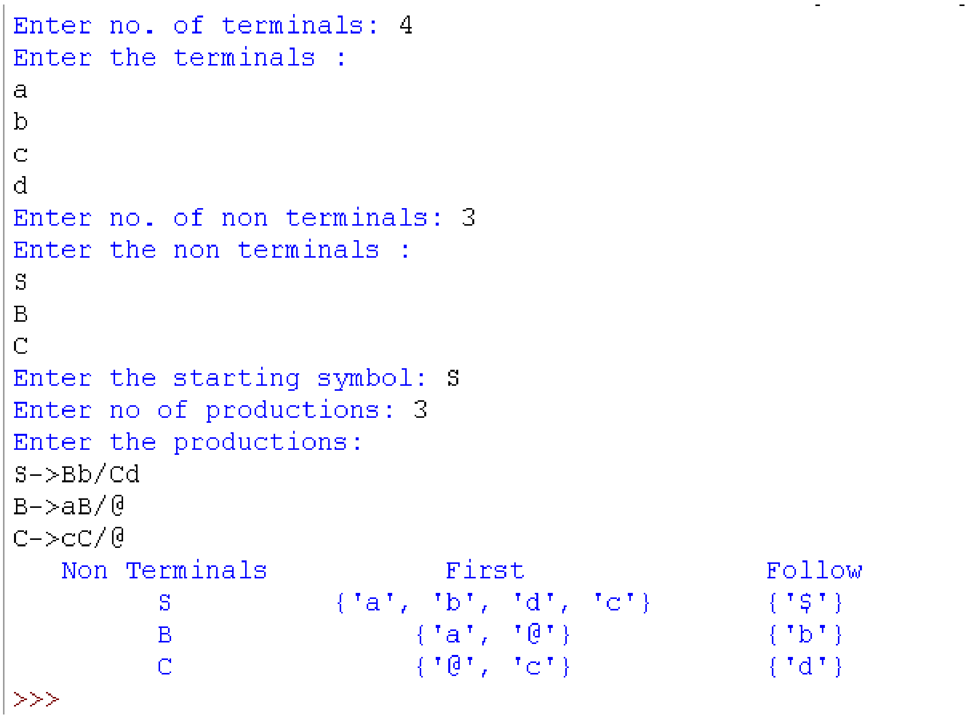
- If production is of form A → αB, then Follow (B) ={FOLLOW (A)}.

## 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])))
```

## Output:



```
Enter no. of terminals: 4
Enter the terminals :
a
b
c
d
Enter no. of non terminals: 3
Enter the non terminals :
S
B
C
Enter the starting symbol: S
Enter no of productions: 3
Enter the productions:
S->Bb/Cd
B->aB/@
C->cC/@
   Non Terminals           First               Follow
         S          {'a', 'b', 'd', 'c'}       {'$'}
         B               {'a', '@'}            {'b'}
         C               {'@', 'c'}            {'d'}
>>>
```