Name : Sejal Pawar

Class : Msc . Computer science ( Part 1 )

Rollno : 504

Subject : Compiler

Topic : First Follow Program

# Compiler



A compiler is a [software](https://www.toppr.com/guides/computer-aptitude-and-knowledge/basics-of-computers/hardware-and-software/) that converts the source code to the object code. In other words, we can say that it converts the [high-level language](https://www.toppr.com/guides/computer-aptitude-and-knowledge/basics-of-computers/computer-languages/) to machine/binary language.

Moreover, it is necessary to perform this step to make the program executable. This is because the computer understands only binary language.

Some compilers convert the high-level language to an assembly language as an intermediate step. Whereas some others convert it directly to machine code. This process of converting the source code into machine code is called **compilation.**Let us learn more about it in detail.

## Uses/Application of Compilers

* Helps to make the code independent of the platform.
* Makes the code free of syntax and semantic errors.
* Generate executable files of code.
* Translates the code from one language to another.

# First Follow Program

**FIRST and FOLLOW** i

In compiler design are two grammatical functions that help you enter table entries. We will discuss the First and Follow in detail below. If the compiler knew ahead of time what the "initial character and follow up of the string produced when a production rule is applied," it might carefully choose which production rule to apply by comparing it to the current character or token in the input string it sees.

### Types of grammar symbols

There are two types of grammar symbols: Terminal symbols and Non-terminal symbols.

Terminal Symbol contains identifiers, lower case alphabets, and special characters on the production's right-hand side. Non -Terminals contain the upper case alphabets, which are on the left-hand side of the production.

## First ( )

**FIRST ()** is a function that specifies the set of terminals that start a string derived from a production rule. It is the first terminal that appears on the right-hand side of the production. For example,

If the Input string is

T->\*FT’/ε

Here we find out that T has two productions like T->\*FT’ and T->ε, after viewing this we found the first of T in both the production statement which is \* and ε.

Then the first of the string will be {\*,ε}.

### Rules to find First()

To find the first() of the grammar symbol, then we have to apply the following set of rules to the given grammar:-

* If X is a terminal, then First(X) is {X}.
* If X is a non-terminal and X tends to aα is production, then add ‘a’ to the first of X. if X->ε, then add null to the First(X).
* If X\_>YZ then if First(Y)=ε, then First(X) = { First(Y)-ε} U First(Z).
* If X->YZ, then if First(X)=Y, then First(Y)=teminal but null then First(X)=First(Y)=terminals.

## Follow ( )

**Follow ()** is a set of terminal symbols that can be displayed just to the right of the non-terminal symbol in any sentence format. It is the first non-terminal appearing after the given terminal symbol on the right-hand side of production.

For example,

If the input string is

E->TE’, F->(E)/id

Here we found that on the right-hand side of the production statement where is the E occurs, we only found E in the production F->(E)/id through which we found the follow of E.

Then the output Follow of E = { ) }, as ‘)’ is the non-terminal in the input string on the right-hand side of the production.

### Rules to find Follow ( )

To find the follow(A) of the grammar symbol, then we have to apply the following set of rules to the given grammar:-

* $ is a follow of ‘S’(start symbol).
* If A->αBβ,β!=ε, then first(β) is in follow(B).
* If A->αB or A->αBβ where First(β)=ε, then everything in Follow(A) is a Follow(B).

## Example

Let us consider grammar to show how to find the first and follow in compiler design.

E->TE’

E’->+TE’/ε

T->FT’

T’->\*FT’/ε

F->(ε)/id

Here,

Terminals are id, \*, +, ε, (, )

Non-terminals are E, E’, T, T’, F

Now let’s try to find the first of ‘E’. here on the right-hand side of the production E->TE’ is T which is a non-terminal but we have to find the terminals so to find terminals we move to the production T->FT’ in which the

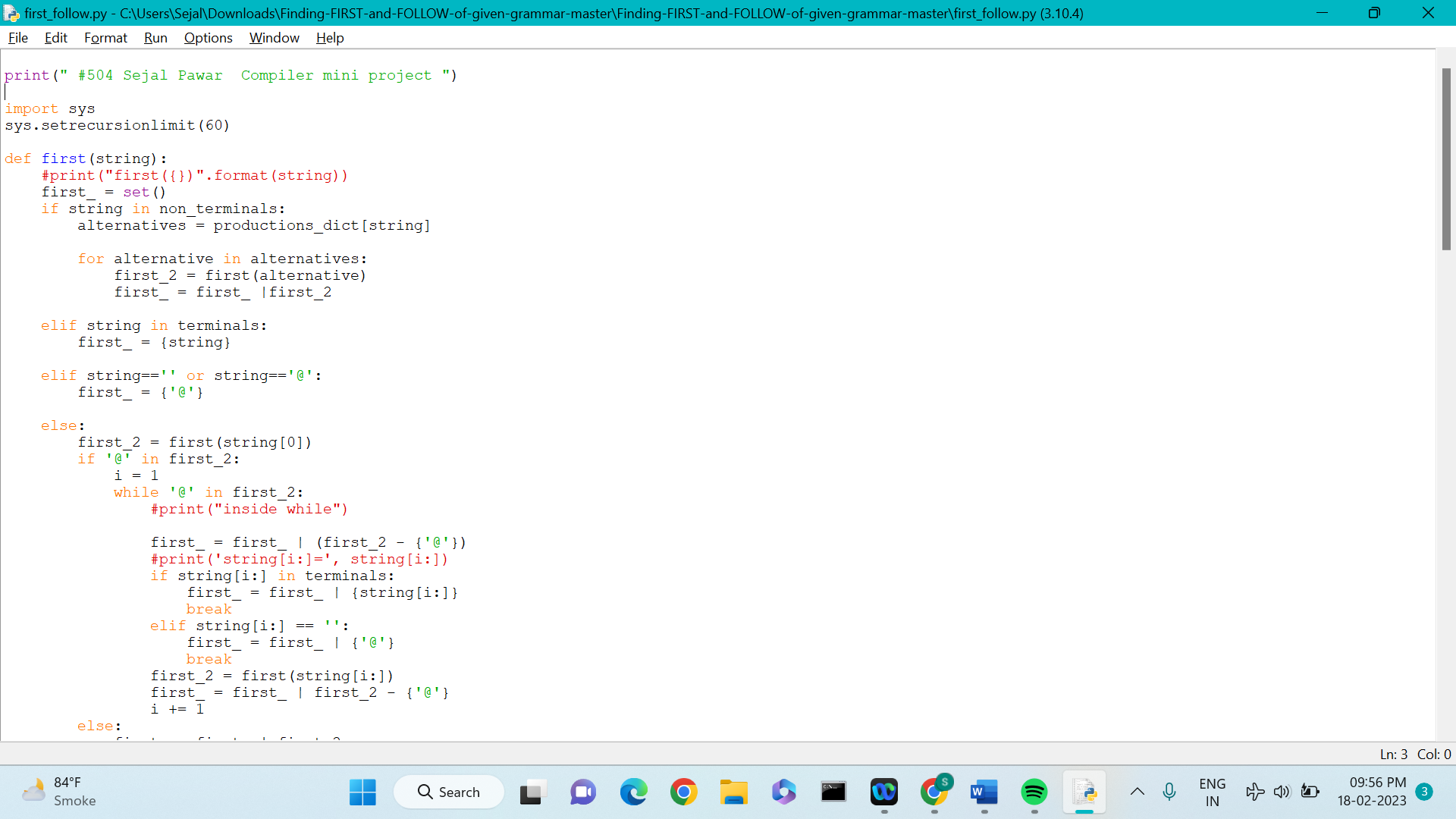
first element is again a non-terminal, so we move to the third production F->(ε)/id in which the first element is a terminal which will be the first of E.

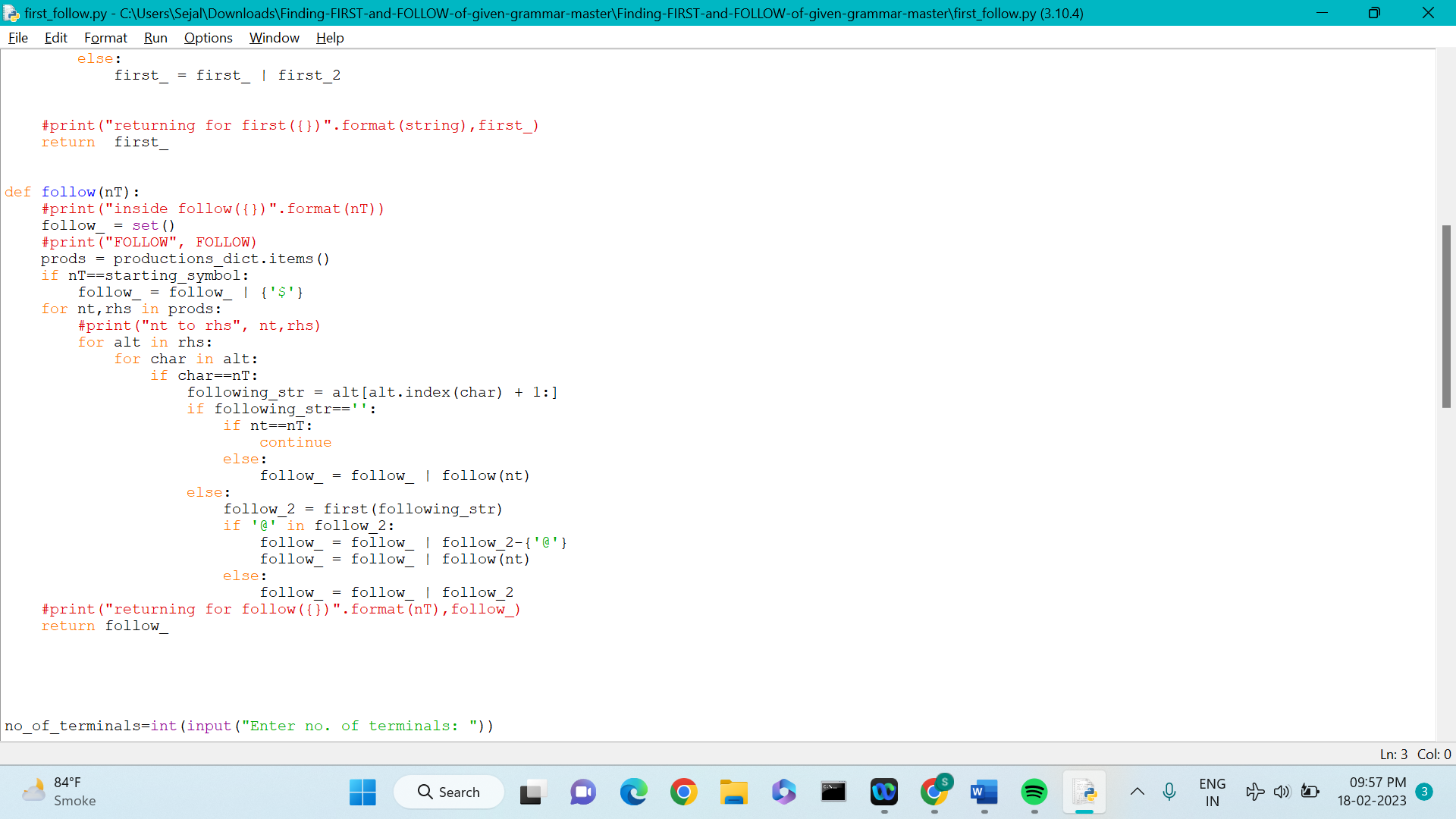
So, First(E)={(, id}

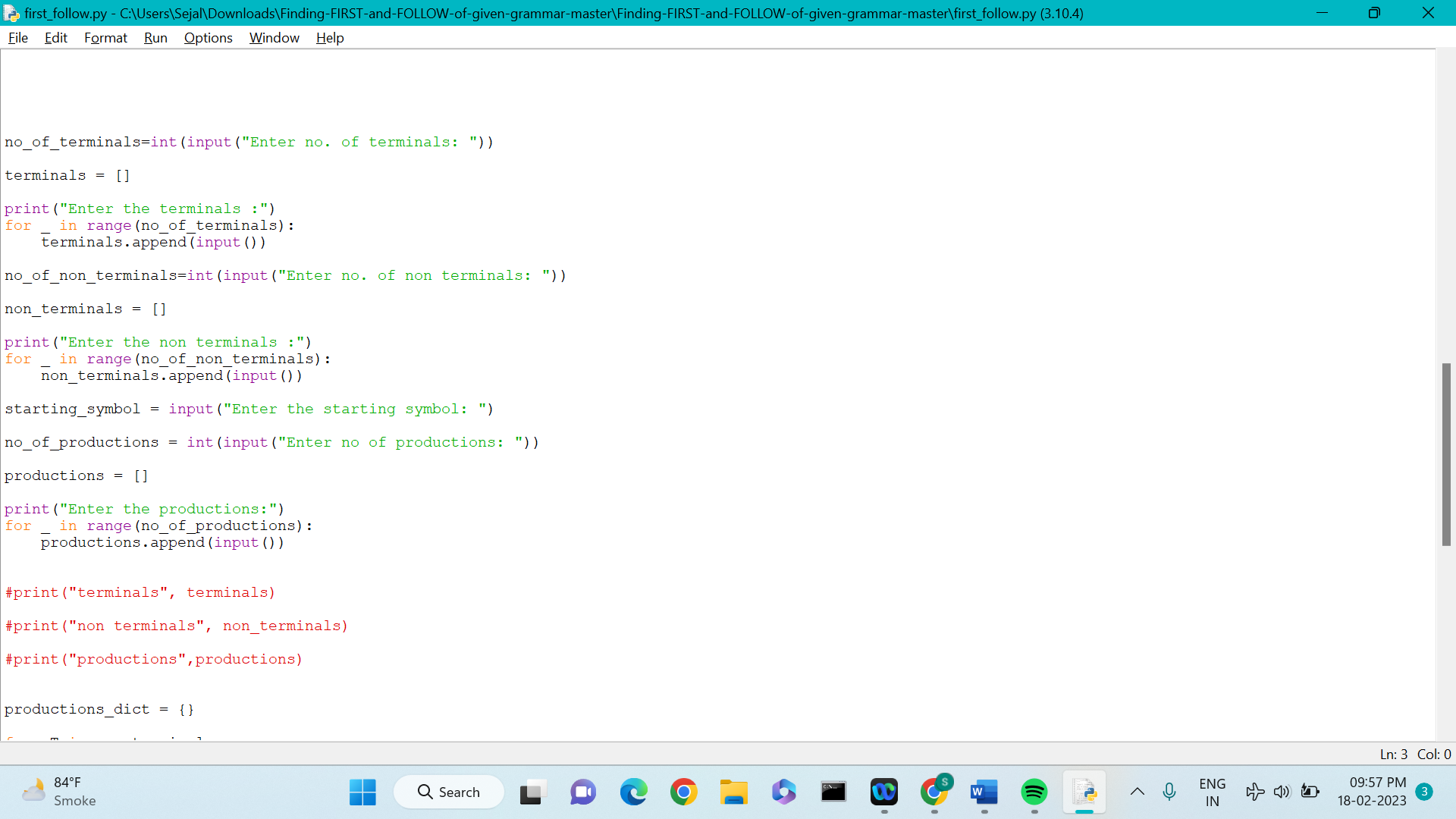
Now let’s try to find the follow of ‘E’, to find this we find the production in which ‘E’ is on the right-hand side and we get production which is F->(E)/id, so the follow will be the next non-terminal followed by the terminals which are ‘)’ and in the follow ‘$’ is always added. So the follow(E)={$,)}

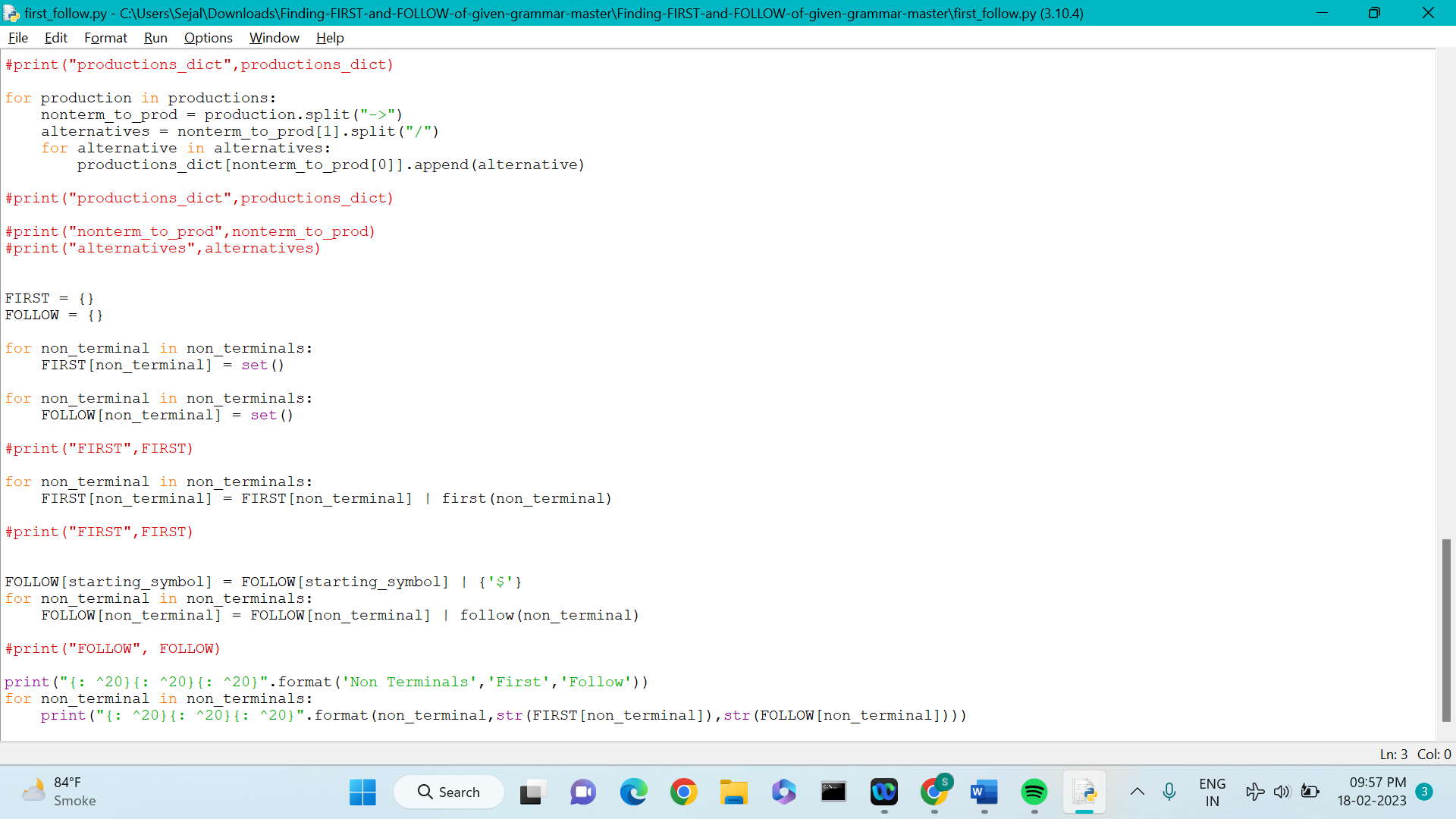
On repeating the above steps to find the first and follow in compiler design, we get

# Program Code Image









# Source Code :

print(" #504 Sejal Pawar Compiler mini project ")

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 :

# compiler

# 504 Sejal Pawar Compiler Mini Project

# 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'} {'+', ')', '$', '\*'}

# Output image

