EXP:5: COMPUTATION OF FIRST AND FOLLOW FOR THE GRAMMAR

AIM: To Design a code to find first and follow for the given grammar.

LANGUAGE USED: Python 3

ALGORITHM/PROCEDURE: -

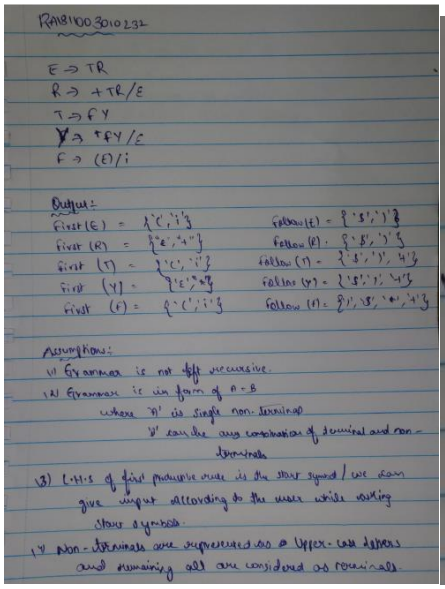
- Epsilon is represented by 'e'.
- Productions are of the form A=B, where 'A' is a single Non-Terminal and 'B' can be any combination of Terminals and Non- Terminals.
- L.H.S. of the first production rule is the start symbol.
- Each production of a non-terminal is entered on a different line.
- Only Upper-Case letters are Non-Terminals and everything else is a terminal.
- Do not use '!' or '$' as they are reserved for special purposes.

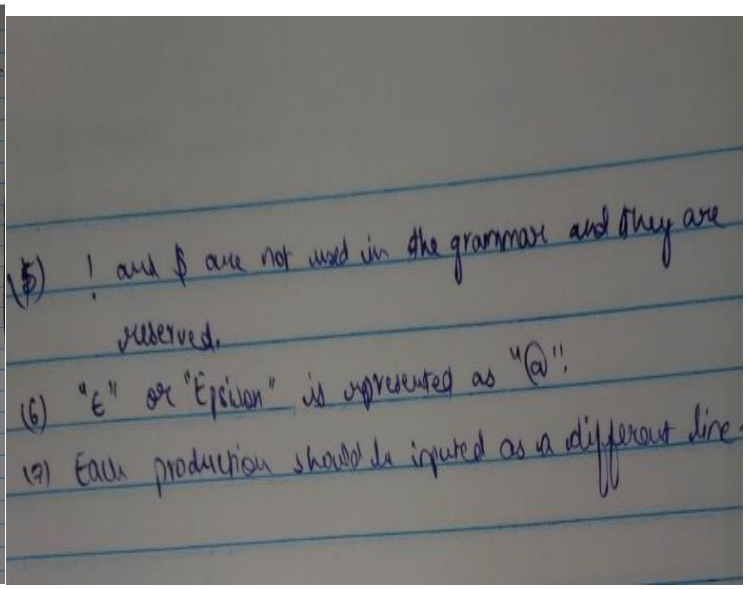
EXPLANATION:

Store the grammar on a 2D character array function is for calculating the first of any non-terminal. Calculation of **first** falls under two broad cases:

- If the first symbol in the R.H.S of the production is a Terminal then it can directly be included in the first set.
- If the first symbol in the R.H.S of the production is a Non-Terminal then call the function again on that Non-Terminal. To handle these cases like Recursion is the best possible solution. Here again, if the First of the new Non-Terminal contains an epsilon then we have to move to the next symbol of the original production which can again be a Terminal or a Non-Terminal.
  ***Note*:** For the second case it is very easy to fall prey to an INFINITE LOOP even if the code looks perfect. So, it is important to keep track of all the function calls at all times and never call the same function again.

SPACE TREE DIAGRAM/ EXPLANATION:





SOURCE CODE: -

```
gram = {
        "E":["E+T","T"],
        "T":["T*F","F"],
        "F":["(E)","i"]
}
def removeDirectLR(gramA, A):
        """gramA is dictonary"""
        temp = gramA[A]
        tempCr = []
        tempInCr = []
        for i in temp:
                if i[0] == A:
                        tempInCr.append(i[1:]+[A+"'"])
                else:
                        tempCr.append(i+[A+"'"])
        tempInCr.append(["e"])

        gramA[A] = tempCr
        gramA[A+"'"] = tempInCr
        return gramA
```

```
def checkForIndirect(gramA, a, ai):
	if ai not in gramA:
		return False
	if a == ai:
		return True
	for i in gramA[ai]:
		if i[0] == ai:
			return False
		if i[0] in gramA:
			return checkForIndirect(gramA, a, i[0])
	return False
def rep(gramA, A):
	temp = gramA[A]
	newTemp = []
	for i in temp:
		if checkForIndirect(gramA, A, i[0]):
			t = []
			for k in gramA[i[0]]:
				t=[]
				t+=k
				t+=i[1:]
				newTemp.append(t)
		else:
			newTemp.append(i)
	gramA[A] = newTemp
	return gramA
def rem(gram):
	c = 1

	conv = {}
	gramA = {}
	revconv = {}
	for j in gram:
		conv[j] = "A"+str(c)
```

```
        gramA["A"+str(c)] = []
        c+=1
for i in gram:
        for j in gram[i]:
                temp = []
                for k in j:
                        if k in conv:
                                temp.append(conv[k])
                        else:
                                temp.append(k)
                gramA[conv[i]].append(temp)
for i in range(c-1,0,-1):
        ai = "A"+str(i)
        for j in range(0,i):
                aj = gramA[ai][0][0]
                if ai!=aj :
                        if aj in gramA and checkForIndirect(gramA,ai,aj):
                                gramA = rep(gramA, ai)
for i in range(1,c):
        ai = "A"+str(i)
        for j in gramA[ai]:
                if ai==j[0]:
                        gramA = removeDirectLR(gramA, ai)
                        break
op = {}
for i in gramA:

        a = str(i)
        for j in conv:
                a = a.replace(conv[j],j)
        revconv[i] = a
for i in gramA:
        l = []
        for j in gramA[i]:
```

```
                        k = []
                        for m in j:
                                if m in revconv:
                                        k.append(m.replace(m,revconv[m]))
                                else:
                                        k.append(m)
                        l.append(k)
                op[revconv[i]] = l
        return op
result = rem(gram)
def first(gram, term):
        a = []
        if term not in gram:
                return [term]
        for i in gram[term]:
                if i[0] not in gram:
                        a.append(i[0])
                elif i[0] in gram:
                        a += first(gram, i[0])
        return a
firsts = {}
for i in result:
        firsts[i] = first(result,i)
        print(f'First({i}):',firsts[i])


def follow(gram, term):
        a = []
        for rule in gram:
                for i in gram[rule]:
                        if term in i:
                                temp = i
                                indx = i.index(term)
                                if indx+1!=len(i):
                                        if i[-1] in firsts:
```

```
                                        a+=firsts[i[-1]]
                                else:
                                        a+=[i[-1]]
                        else:
                                a+=["e"]
                        if rule != term and "e" in a:
                                a+= follow(gram,rule)
        return a
follows = {}
for i in result:
        follows[i] = list(set(follow(result,i)))
        if "e" in follows[i]:
                follows[i].pop(follows[i].index("e"))
        follows[i]+=["$"]
        print(f'Follow({i}):',follows[i])
```

OUTPUT:

First(E): ['(', 'i']

First(T): ['(', 'i']

First(F): ['(', 'i']

First(E'): ['+', 'e']

First(T'): ['*', 'e']

Follow(E): [')', '$']

Follow(T): ['+', ')', '$']

Follow(F): ['+', ')', '*', '$']

Follow(E'): [')', '$']

Follow(T'): ['+', ')', '$']

RESULT: Therefore, we successfully implemented a code for finding first and follow of the given grammar.