EXP-8: Leading and Trailing

**Aim:** Implementation of leading and trailing.

**Procedure:**

1. For Leading, check for the first non-terminal.

2. If found, print it.

3. Look for next production for the same non-terminal.

4. If not found, recursively call the procedure for the single non-terminal present before the

comma or End Of Production String.

5. Include it's results in the result of this non-terminal.

6. For trailing, we compute same as leading but we start from the end of the production to

the beginning.

7. Stop

**SOURCE CODE:**

a = ["E=E+T",  
 "E=T",  
 "T=T\*F",  
 "T=F",  
 "F=(E)",  
 "F=i"]  
  
rules = {}  
terms = []  
*for* i *in* a:  
 temp = i.split("=")  
 terms.append(temp[0])  
 *try*:  
 rules[temp[0]] += [temp[1]]  
 *except*:  
 rules[temp[0]] = [temp[1]]  
  
terms = list(set(terms))  
print(rules, terms)  
  
  
*def* leading(*gram*, *rules*, term, *start*):  
 s = []  
 *if gram*[0] *not in* terms:  
 *return gram*[0]  
 *elif* len(*gram*) == 1:  
 *return* [0]  
 *elif gram*[1] *not in* terms *and gram*[-1] *is not start*:  
 *for* i *in rules*[*gram*[-1]]:  
 s += leading(i, *rules*, *gram*[-1], *start*)  
 s += [*gram*[1]]  
 *return* s  
  
  
*def* trailing(*gram*, *rules*, term, *start*):  
 s = []  
 *if gram*[-1] *not in* terms:  
 *return gram*[-1]  
 *elif* len(*gram*) == 1:  
 *return* [0]  
 *elif gram*[-2] *not in* terms *and gram*[-1] *is not start*:  
  
 *for* i *in rules*[*gram*[-1]]:  
 s += trailing(i, *rules*, *gram*[-1], *start*)  
 s += [*gram*[-2]]  
 *return* s  
  
  
leads = {}  
trails = {}  
*for* i *in* terms:  
 s = [0]  
 *for* j *in* rules[i]:  
 s += leading(j, rules, i, i)  
 s = set(s)  
 s.remove(0)  
 leads[i] = s  
 s = [0]  
 *for* j *in* rules[i]:  
 s += trailing(j, rules, i, i)  
 s = set(s)  
 s.remove(0)  
 trails[i] = s  
  
*for* i *in* terms:  
 print("LEADING(" + i + "):", leads[i])  
*for* i *in* terms:  
 print("TRAILING(" + i + "):", trails[i])

**OUTPUT:**

C:\Users\hp\AppData\Local\Programs\Python\Python39\python.exe "F:/Python/DAA/Compiler Design/predictive-parser-master/l&t.py"

{'E': ['E+T', 'T'], 'T': ['T\*F', 'F'], 'F': ['(E)', 'i']} ['T', 'E', 'F']

LEADING(T): {'i', '\*', '('}

LEADING(E): {'+', '(', 'i', '\*'}

LEADING(F): {'i', '('}

TRAILING(T): {'\*', ')', 'i'}

TRAILING(E): {'+', 'i', '\*', ')'}

TRAILING(F): {')', 'i'}

Process finished with exit code 0

**RESULT:**

Hence the leading and trailing is implemented.