LEADING AND TRAILING SYMBOLS

EX. NO. 8

YUVRAJ SINGH CHAUHAN

RA1911027010058

AIM: To write a program for leading and trailing symbols.

ALGORITHM:

Leading and Trailing are functions specific to generating an operator-precedence parser, which is only applicable if you have an operator precedence grammar. An operator precedence grammar is a special case of an operator grammar, and an operator grammar has the important property that no production has two consecutive non-terminals.

(An operator precedence grammar is, loosely speaking, an operator grammar which can be parsed with an operator precedence parser)

Given an operator grammar, the function Leading (resp. Trailing) of a non-terminal produces the set of terminals which could be (recursively) the first (resp. last) terminal in a production for that non-terminal.

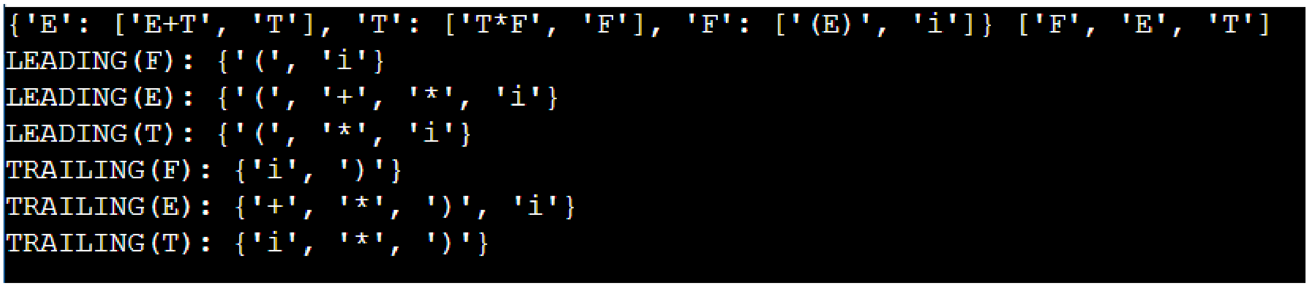
Another way to think of that a terminal is in the Leading set for a non-terminal if it is "visible" from the beginning of a production. We consider non-terminals to be "transparent", so a terminals could be visible through a non-terminal or by looking into a visible non-terminal.

PROGRAM:

```
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 :



```
{'E': ['E+T', 'T'], 'T': ['T*F', 'F'], 'F': ['(E)', 'i']} ['F', 'E', 'T']
LEADING(F): {'(', 'i'}
LEADING(E): {'(', '+', '*', 'i'}
LEADING(T): {'(', '*', 'i'}
TRAILING(F): {'i', ')'}
TRAILING(E): {'+', '*', ')', 'i'}
TRAILING(T): {'i', '*', ')'}
```

RESULT :

Leading and trailing symbols was implemented successfully using python language.