# Name – Atharva

# Rewatkar

# Section - E

# Roll Number – 32

In [5]:
```
from collections import OrderedDict

def isterminal(char):
    if(char.isupper() or char == "`"):
        return False
    else:
        return True

def insert(grammar, lhs, rhs):
    if(lhs in grammar and rhs not in grammar[lhs] and grammar[lhs] != "null"):
        grammar[lhs].append(rhs)
    elif(lhs not in grammar or grammar[lhs] == "null"):
        grammar[lhs] = [rhs]
    return grammar

def first(lhs, grammar, grammar_first):
    rhs = grammar[lhs]
    for i in rhs:
        k = 0
        flag = 0
        current = []
        confirm = 0
        flog = 0
        if(lhs in grammar and "`" in grammar_first[lhs]):
            flog = 1
        while(1):
            check = []
            if(k>=len(i)):
                if(len(current)==0 or flag == 1 or confirm == k or flog == 1):
                    grammar_first = insert(grammar_first, lhs, "`")
                break
            if(i[k].isupper()):
                if(grammar_first[i[k]] == "null"):
                    grammar_first = first(i[k], grammar, grammar_first)
                  # print("state ", lhs, "i ", i, "k, ", k, grammar_first[i[k]])
                for j in grammar_first[i[k]]:
                    grammar_first = insert(grammar_first, lhs, j)
                    check.append(j)
            else:
                grammar_first = insert(grammar_first, lhs, i[k])
                check.append(i[k])
            if(i[k]=="`"):
                flag = 1
            current.extend(check)
            if("`" not in check):
```

```
if(flog == 1):
    grammar_first = insert(grammar_first, lhs, "`")
 break
```

```
            else:
                confirm += 1
                k+=1
                grammar_first[lhs].remove("`")
    return(grammar_first)

def rec_follow(k, next_i, grammar_follow, i, grammar, start, grammar_first, lhs):
    if(len(k)==next_i):
        if(grammar_follow[i] == "null"):
            grammar_follow = follow(i, grammar, grammar_follow, start)
        for q in grammar_follow[i]:
            grammar_follow = insert(grammar_follow, lhs, q)
    else:
        if(k[next_i].isupper()):
            for q in grammar_first[k[next_i]]:
                if(q=="`"):
                    grammar_follow = rec_follow(k, next_i+1, grammar_follow, i, gramm
                else:
                    grammar_follow = insert(grammar_follow, lhs, q)
        else:
            grammar_follow = insert(grammar_follow, lhs, k[next_i])

    return(grammar_follow)

def follow(lhs, grammar, grammar_follow, start):
    for i in grammar:
        j = grammar[i]
        for k in j:
            if(lhs in k):
                next_i = k.index(lhs)+1
                grammar_follow = rec_follow(k, next_i, grammar_follow, i, grammar, st
    if(lhs==start):
        grammar_follow = insert(grammar_follow, lhs, "$")
    return(grammar_follow)

def  show_dict(dictionary):
    for key in dictionary.keys():
        print(key+"  :  ", end = "")
        for item in dictionary[key]:
            if(item == "`"):
                print("Epsilon, ", end = "")
            else:
                print(item+", ", end = "")
        print("\b\b")

def get_rule(non_terminal, terminal, grammar, grammar_first):
    for rhs in grammar[non_terminal]:
        #print(rhs)
        for rule in rhs:
            if(rule == terminal):
                string = non_terminal+"~"+rhs
                return string

            elif(rule.isupper() and terminal in grammar_first[rule]):
                string = non_terminal+"~"+rhs
                return string

def generate_parse_table(terminals, non_terminals, grammar, grammar_first, grammar_fo
    parse_table = [[""]*len(terminals) for i in range(len(non_terminals))]
```

```
    for non_terminal in non_terminals:
        for terminal in terminals:
            #print(terminal)
            #print(grammar_first[non_terminal])
            if terminal in grammar_first[non_terminal]:
                rule = get_rule(non_terminal, terminal, grammar, grammar_first)
                #print(rule)

            elif("`" in grammar_first[non_terminal] and terminal in grammar_follow[no
                rule = non_terminal+"~`"

            elif(terminal  in  grammar_follow[non_terminal]):
                rule = "Sync"

            else:
                rule = ""

            parse_table[non_terminals.index(non_terminal)][terminals.index(terminal)]

    return(parse_table)

def  display_parse_table(parse_table,  terminal,  non_terminal):
    print("\t\t\t\t",end = "")
    for terminal in terminals:
        print(terminal+"\t\t", end = "")
    print("\n\n")

    for non_terminal in non_terminals:
        print("\t\t"+non_terminal+"\t\t", end = "")
        for terminal in terminals:
            print(parse_table[non_terminals.index(non_terminal)][terminals.index(term
        print("\n")


def parse(expr, parse_table, terminals, non_terminals):
    stack = ["$"]
    stack.insert(0,  non_terminals[0])

    print("\t\t\tMatched\t\t\tStack\t\t\tInput\t\t\tAction\n")
    print("\t\t\t-\t\t\t", end = "")
    for i in stack:
        print(i,  end = "")
    print("\t\t\t", end = "")
    print(expr+"\t\t\t", end = "")
    print("-")

    matched = "-"
    while(True):
        action = "-"

        if(stack[0] == expr[0] and stack[0] == "$"):
            break

        elif(stack[0] == expr[0]):
            if(matched == "-"):
                matched = expr[0]
            else:
                matched = matched + expr[0]
            action = "Matched "+expr[0]
            expr = expr[1:]
```

```
            stack.pop(0)

        else:
            action  =  parse_table[non_terminals.index(stack[0])][terminals.index(expr[
            stack.pop(0)
            i = 0
            for item in action[2:]:
                if(item != "`"):
                    stack.insert(i,item)
                i+=1

        print("\t\t\t"+matched+"\t\t\t", end = "")
        for i in stack:
            print(i,  end = "")
        print("\t\t\t", end = "")
        print(expr+"\t\t\t", end = "")
        print(action)




################################              Main_Driver             ###############




grammar = OrderedDict()
grammar_first = OrderedDict()
grammar_follow = OrderedDict()

f = open('grammar2.txt')
for i in f:
    i = i.replace("\n", "")
    lhs = ""
    rhs = ""
    flag = 1
    for j in i:
        if(j=="~"):
            flag = (flag+1)%2
            continue
        if(flag==1):
            lhs += j
        else:
            rhs += j
    grammar = insert(grammar, lhs, rhs)
    grammar_first[lhs] = "null"
    grammar_follow[lhs] = "null"

print("Grammar\n")
show_dict(grammar)

for lhs in grammar:
    if(grammar_first[lhs] == "null"):
        grammar_first = first(lhs, grammar, grammar_first)

print("\n\n\n")
print("First\n")
show_dict(grammar_first)
```

```
start = list(grammar.keys())[0]
for lhs in grammar:
    if(grammar_follow[lhs] == "null"):
        grammar_follow = follow(lhs, grammar, grammar_follow, start)

print("\n\n\n")
print("Follow\n")
show_dict(grammar_follow)


non_terminals = list(grammar.keys())
terminals = []

for i in grammar:
    for rule in grammar[i]:
        for char in rule:

            if(isterminal(char) and char not in terminals):
                terminals.append(char)

terminals.append("$")

#print(non_terminals)
#print(terminals)


print("\n\n\n\n\t\t\t\t\t\t\tParse  Table\n\n")
parse_table = generate_parse_table(terminals, non_terminals, grammar, grammar_first,
display_parse_table(parse_table, terminals, non_terminals)


#expr = input("Enter the expression ending with $ : ")
expr = "i+i*i$"

print("\n\n\n\n\n\n")
print("\t\t\t\t\t\t\tParsing  Expression\n\n")
parse(expr, parse_table, terminals, non_terminals)
```

Grammar

```
E  :  TL,
L  :  +TL, Epsilon,
T  :  FK,
K  :  *FK, Epsilon,
F  :  i, (E),
```

First

```
E  :  i, (,
L  :  +, Epsilon,
T  :  i, (,
K  :  *, Epsilon,
F  :  i, (,
```

Follow

```
E  :  ), $,
L  :  ), $,
T  :  +, ), $,
K  :  +, ), $,
F  :  *, +, ), $,
```

Parse Table

| | + | * | i | ( | ) | $ |
|---|---|---|---|---|---|---|
| E | | | E~TL | E~TL | Sync | Sync |
| L | L~+TL | | | | L~` | L~` |
| T | Sync | | T~FK | T~FK | Sync | Sync |
| K | K~` | K~*FK | | | K~` | K~` |
| F | Sync | Sync | F~i | F~(E) | Sync | Sync |

Parsing Expression

| Matched | Stack | Input | Action |
|---|---|---|---|
| - | E$ | i+i*i$ | - |
| - | TL$ | i+i*i$ | E~TL |
| - | FKL$ | i+i*i$ | T~FK |
| - | iKL$ | i+i*i$ | F~i |
| i | KL$ | +i*i$ | Matched i |
| i | L$ | +i*i$ | K~` |
| i | +TL$ | +i*i$ | L~+TL |
| i+ | TL$ | i*i$ | Matched + |
| i+ | FKL$ | i*i$ | T~FK |
| i+ | iKL$ | i*i$ | F~i |
| i+i | KL$ | *i$ | Matched i |
| i+i | *FKL$ | *i$ | K~*FK |
| i+i* | FKL$ | i$ | Matched * |
| i+i* | iKL$ | i$ | F~i |
| i+i*i | KL$ | $ | Matched i |
| i+i*i | L$ | $ | K~` |
| i+i*i | $ | $ | L~` |

In [ ]: