SHUSHRUT KUMAR
RA1811028010049
COMPILER DESIGN LAB EXP5

FIRST AND FOLLOW

AIM: To write a program to perform first and follow using any language.

ALGORITHM:

For computing the first:

1. If X is a terminal then FIRST(X) = {X}

Example: F -> I | id

We can write it as FIRST(F) -> { (, id)

2. If X is a non-terminal like E -> T then to get FIRSTI substitute T with other productions until you get a terminal as the first symbol

3. If X -> ε then add ε to FIRST(X).

For computing the follow:

1. Always check the right side of the productions for a non-terminal, whose FOLLOW set is being found. (never see the left side).

2. (a) If that non-terminal (S,A,B…) is followed by any terminal (a,b…,*,+,(,)…) , then add that terminal into the FOLLOW set.

(b) If that non-terminal is followed by any other non-terminal then add FIRST of other nonterminal into the FOLLOW set.

CODE :

```
import re
import string
import pandas as pd


def parse(user_input,start_symbol,parsingTable):

    #flag
    flag = 0

    #appending dollar to end of input
    user_input = user_input + "$"

    stack = []

    stack.append("$")
    stack.append(start_symbol)

    input_len = len(user_input)
    index = 0


    while len(stack) > 0:

        #element at top of stack
        top = stack[len(stack)-1]

        print ("Top =>",top)

        #current input
```

```
current_input = user_input[index]

print ("Current_Input => ",current_input)

if top == current_input:
        stack.pop()
        index = index + 1
else:

        #finding value for key in table
        key = top , current_input
        print (key)

        #top of stack terminal => not accepted
        if key not in parsingTable:
                flag = 1
                break

        value = parsingTable[key]
        if value !='@':
                value = value[::-1]
                value = list(value)

                #poping top of stack
                stack.pop()

                #push value chars to stack
                for element in value:
                        stack.append(element)
        else:
                stack.pop()
```

```
    if flag == 0:
        print ("String accepted!")
    else:
        print ("String not accepted!")



def ll1(follow, productions):

    print ("\nParsing Table\n")

    table = {}
    for key in productions:
        for value in productions[key]:
            if value!='@':
                for element in first(value, productions):
                    table[key, element] = value
            else:
                for element in follow[key]:
                    table[key, element] = value

    for key,val in table.items():
        print (key,"=>",val)

    new_table = {}
    for pair in table:
        new_table[pair[1]] = {}

    for pair in table:
        new_table[pair[1]][pair[0]] = table[pair]
```

```
        print ("\n")
        print ("\nParsing Table in matrix form\n")
        print (pd.DataFrame(new_table).fillna('-'))
        print ("\n")

        return table

def follow(s, productions, ans):
        if len(s)!=1 :
                return {}

        for key in productions:
                for value in productions[key]:
                        f = value.find(s)
                        if f!=-1:
                                if f==(len(value)-1):
                                        if key!=s:
                                                if key in ans:
                                                        temp = ans[key]
                                                else:
                                                        ans = follow(key, productions, ans)
                                                        temp = ans[key]
                                                ans[s] = ans[s].union(temp)
                                else:
                                        first_of_next = first(value[f+1:], productions)
                                        if '@' in first_of_next:
                                                if key!=s:
                                                        if key in ans:
                                                                temp = ans[key]
```

```
                                                    else:
                                                        ans = follow(key, productions, ans)
                                                        temp = ans[key]
                                                    ans[s] = ans[s].union(temp)
                                                    ans[s] = ans[s].union(first_of_next) - {'@'}
                                            else:
                                                ans[s] = ans[s].union(first_of_next)
        return ans

def first(s, productions):
        c = s[0]
        ans = set()
        if c.isupper():
                for st in productions[c]:
                        if st == '@' :
                                if len(s)!=1 :
                                        ans = ans.union( first(s[1:], productions) )
                                else :
                                        ans = ans.union('@')
                        else :
                                f = first(st, productions)
                                ans = ans.union(x for x in f)
        else:
                ans = ans.union(c)
        return ans

if __name__=="__main__":
        productions=dict()
        grammar = open("grammar2", "r")
        first_dict = dict()
        follow_dict = dict()
```

```
flag = 1
start = ""
for line in grammar:
	l = re.split("( |->|\n|\|)*", line)
	lhs = l[0]
	rhs = set(l[1:-1])-{''}
	if flag :
		flag = 0
		start = lhs
	productions[lhs] = rhs

print ('\nFirst\n')
for lhs in productions:
	first_dict[lhs] = first(lhs, productions)
for f in first_dict:
	print (str(f) + " : " + str(first_dict[f]))
print ("")

print ('\nFollow\n')

for lhs in productions:
	follow_dict[lhs] = set()

follow_dict[start] = follow_dict[start].union('$')

for lhs in productions:
	follow_dict = follow(lhs, productions, follow_dict)

for lhs in productions:
	follow_dict = follow(lhs, productions, follow_dict)
```

```
for f in follow_dict:
        print (str(f) + " : " + str(follow_dict[f]))

ll1Table = ll1(follow_dict, productions)

#parse("a*(a+a)",start,ll1Table)
parse("ba=a+23",start,ll1Table)

# tp(ll1Table)
```

OUTPUT :

```
Enter the productions(type 'end' at the last of the production)
E->TA
A->+TA
A->ε
T->FB
B->*FB
B->ε
F->(E)
F->i
end
E       (i      $)
A       +ε      $)
T       (i      +$)
B       *ε      +$)
F       (i      *+$)
```

RESULT: The FIRST and FOLLOW sets of the non-terminals of a grammar were found successfully using python language.