Practical File
of

Compiler Design (CS654)

Practical File submitted in partial fulfillment of

the requirements for the award of

Bachelor in Engineering
(COMPUTER SCIENCE AND ENGINEERING)

Submitted by

Rajwinder Singh
(Roll no: CO15340)

Under the supervision of
Dr. Gulshan Goyal
AP, CSE Department

CHANDIGARH COLLEGE OF ENGINEERING AND TECHNOLOGY

(DEGREE WING)

Government Institute under Chandigarh (UT) Administration, Affiliated to Panjab University , Chandigarh

Sector-26, Chandigarh. PIN-160019

January-June, 2018

Table of Contents

S.No	Topic	Date	Page No	Signature
1	Introduction to Compiler Design	January 16, 2018	1	
2	To check whether the given string is recognised by the DFA or not	January 23, 2018	6	
3	Write a program to construct DFA i.e. Deterministic Finite Automata for given condition on the string that the user will provide to the code as input.	January 30, 2018	11	
4	Write a program to find and locate the comments in given program.	Feburary 06,2018	18	
5	Write a program to check for the lexemes from the code written in the file and classify it under various tokens.	Feburary 13,2018	22	
6	Write a program to check for the lexemes from the code entered by the user and classify it under various token categories.	Feburary20, 2018	26	
7	Write a program to convert a given NFA to DFA	Feburary 27,2018	30	
8	Write a program to derive a string from the rules of given grammar by the means of left most and right most derivation (i.e. LMD and RMD).	March 06, 2018	36	
9	Write a program to implement operator precedence parser and show the step by step stack implementation of parsing.	March20, 2018	48	
10	Write a program to implement predictive parser or LL parser and show the step by step stack implementation of parsing.	March 27, 2018	55	
11	Write a program to implement SLR parser.	April 3, 2018	64	
12	Write a program to implement CLR parser.	April 10, 2018	74	
13	Write a program to implement LALR parser.	April 10, 2018	81	

Acknowledgement

I have taken efforts in this lab file. However, it would not have been possible without the kind support and help of many individuals and organization. I am thankful for their aspiring guidance, invaluably constructive criticism and friendly advice during the work. I would like to extend my sincere thanks to all of them.

First of all, I would like to extend my gratitude towards my parents Mr. Kuldeep Singh and Mrs. Paramjeet Kaur for providing me resources and for always being there to support and motivate me. Sincere thanks to my beloved family for their endless love throughout my studies

I express my warm thanks to our Compiler Design professor Dr. Gulshan Goyal, for his guidance and constant supervision as well as for providing necessary information regarding the lab file & also for his support in completing this lab file.

I would like to express my gratitude towards my friends and members of CCET for their kind co-operation and encouragement which helped me in completion of this report.

Rajwinder Singh

Practical-1 January 23, 2018

Aim:

To check whether the given string is recognised by the DFA or not

Input:

The user is supposed to enter the total number of states in the DFA (including the final state), the number of input symbols, the number of final states (can be more than one too). Along with this he must enter the input symbols too for example a,b or 1,2 etc.

Expected Output:

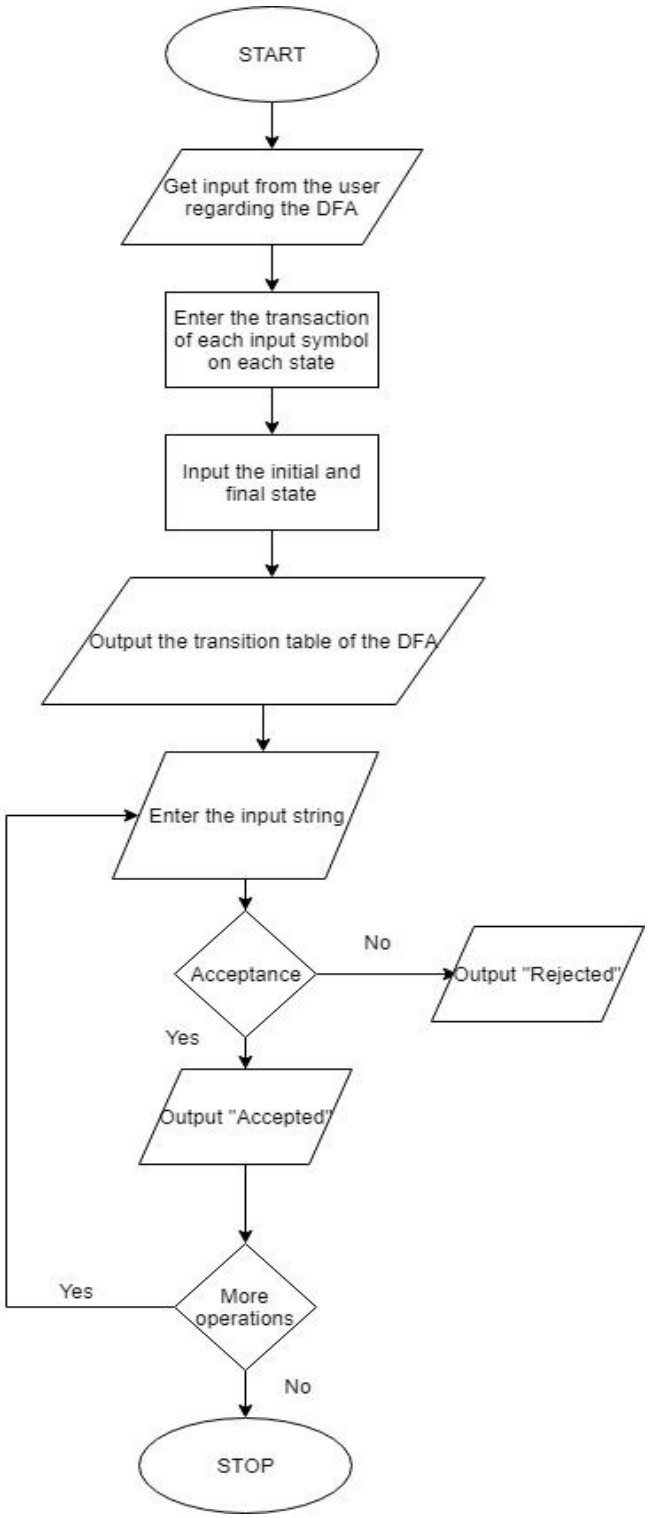
After using the Transition table entered by the user the program executes the output "Given string is acceptable." Or "Given string is not acceptable".

Algorithm:

1. START
2. Ask the user to enter the following information at the start of the program:
    a) Total number of states and number of input symbols
    b) Values of the input symbols
    c) Total number of final states
    d) Values of final state
3. For each state and for each input symbol call the transition function which will set the transition for that state for the given input symbol.
4. Input the initial and the final states of the DFA.
5. Print the transition table of the DFA that user has entered.
6. Enter the string to be checked
7. If the string is accepted then
    a) Output "Accepted"
8. Else output "Rejected"
9. Ask the user if there are any more operations to be performed
10. If the user input Y goto STEP6 else proceed.

11. STOP

Flowchart:



Source code and Output:

jupyter CDlab1 Last Checkpoint: an hour ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Python program to get dfa from user and check for string acceptance

Input States

In [1]:
```
while True:
   try:
      print("Enter number of states :")
      nstates=int(input())
      break
   except:
      print("Enter integer value :")

states=[]
print("Enter name of states :")
for i in range(nstates):
    states.append(input())

initial=states[0]
print("Initial state :"+initial)
```

```
Enter number of states :
3
Enter name of states :
q1
q2
q3
Initial state :q1
```

Input Symbols

In [2]:
```
num_input=int(input(("\n Enter number of input symbols :")))
symbol=[]
print("Enter input alphabets :")
for i in range(num_input):
    symbol.append(input())
```

```
 Enter number of input symbols :2
Enter input alphabets :
a
b
```

Transitions

In [3]:
```
transition={}
for j in range(num_input):
    for i in range(nstates):
        while True:
               print("Enter transition ("+states[i]+","+symbol[j]+"):" )
               t=input()
               if t not in states:
                   print("Enter valid state ")
               else:
                   break
        var=(states[i],symbol[j])
        #print(var)
        transition[var]=t

print(transition)
```

```
Enter transition (q1,a):
q2
Enter transition (q2,a):
q3
Enter transition (q3,a):
q1
Enter transition (q1,b):
q1
Enter transition (q2,b):
```

jupyter CDlab1 Last Checkpoint: an hour ago (autosaved)

```
q2
Enter transition (q3,b):
q3
{('q1', 'a'): 'q2', ('q2', 'a'): 'q3', ('q3', 'a'): 'q1', ('q1', 'b'): 'q1', ('q2', 'b'): 'q2', ('q3', 'b'): 'q3'}
```

Final State

In [4]:
```
num_final=int(input("Enter number of final states"))
final=[]
print("Enter final states/state: ")
for i in range(num_final):
    final.append(input())
```

```
Enter number of final states1
Enter final states/state:
q3
```

Transition Table

In [5]:
```
print("********** Transition Table **********")
print("States    ",end='')
for i in range(num_input):
    print(symbol[i]+"     ",end='')

print("")

for i in range(nstates):
    if states[i] in final:
        print("**" + states[i] + "     ", end='')
        key = transition.keys()
        for k in key:
            if states[i] in k:
                print(transition[k] + "     ", end='')
        print("")
```

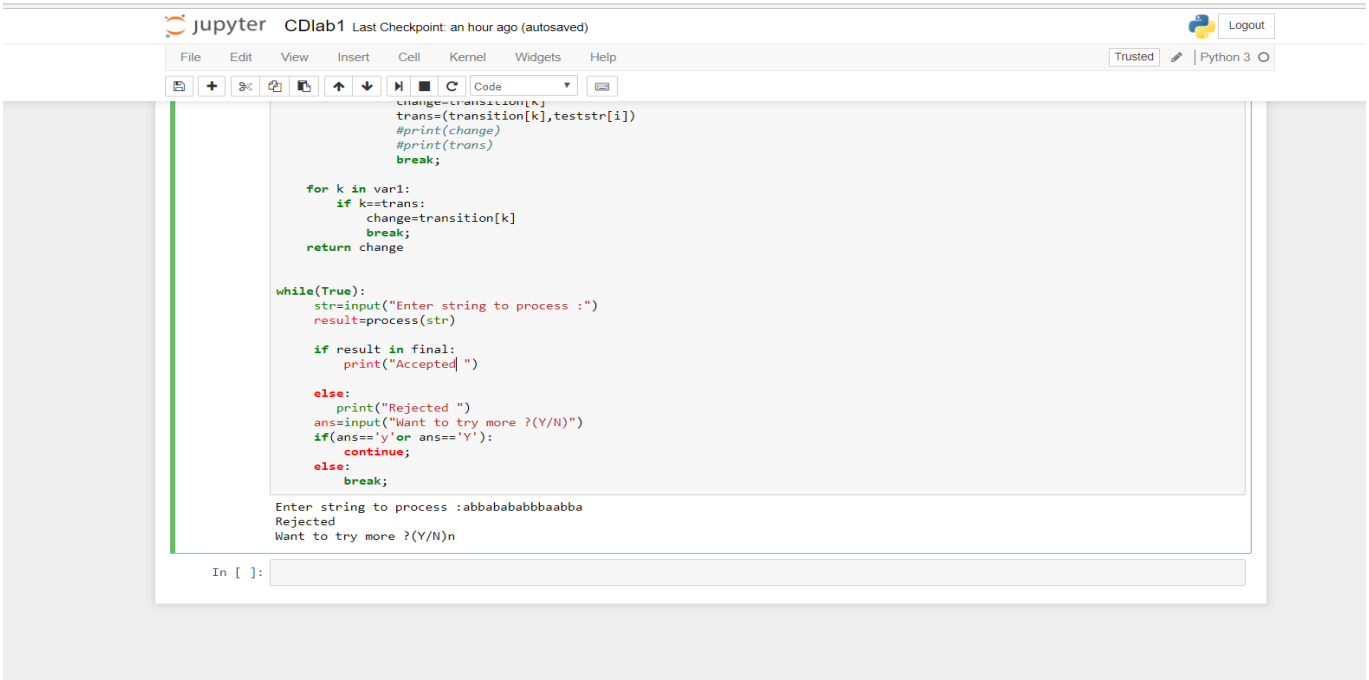
jupyter CDlab1 Last Checkpoint: an hour ago (autosaved)

```
    elif states[i]==initial:
        print("->"+states[i] + "     ", end='')
        key = transition.keys()
        for k in key:
            if states[i] in k:
                print(transition[k] + "     ", end='')
        print("")

    else:
         print("  "+states[i]+"     ",end='')
         key = transition.keys()
         for k in key:
            if states[i] in k:
                print(transition[k]+"     ",end='')
         print("")
```

```
********** Transition Table **********
States    a     b
->q1     q2     q1
  q2     q3     q2
**q3     q1     q3
```

String Processing

In [6]:
```
def process(teststr):
    if teststr==':
        return
    var1=transition.keys()
    trans=(initial,teststr[0])
    loopcount=len(teststr)
    for i in range(1,loopcount):
        for k in var1:
            if k==trans:
                change=transition[k]
                trans=(transition[k],teststr[i])
                #print(change)
                #print(trans)
                break;
```



Frequently Asked Questions(FAQs)

Q1. Which mathematical expression is used to define regular languages?

Ans. Regular expressions.

Q2. Name three types of finite automata.

Ans. DFA, NFA, NFA with null transitions.

Q3.Kleen's theorem is used to generate which type of automata?

Ans. NFA with null transition.

Q4. Write the notation of alphabet having symbols a and b?

Ans. $\Sigma=\{a,b\}$.

Q5. At max how many transitions are possible from a state for a given input symbol?

Ans. Three.

Practical-2 January 30, 2018

Aim:

Write a program to construct DFA i.e. Deterministic Finite Automata for given condition on the string that the user will provide to the code as input.

Input:

In this program user is required to enter the substring of the string for which the DFA(i.e. Deterministic Finite Automata) to be constructed. User will input the string and according to this string the DFA will be constructed.

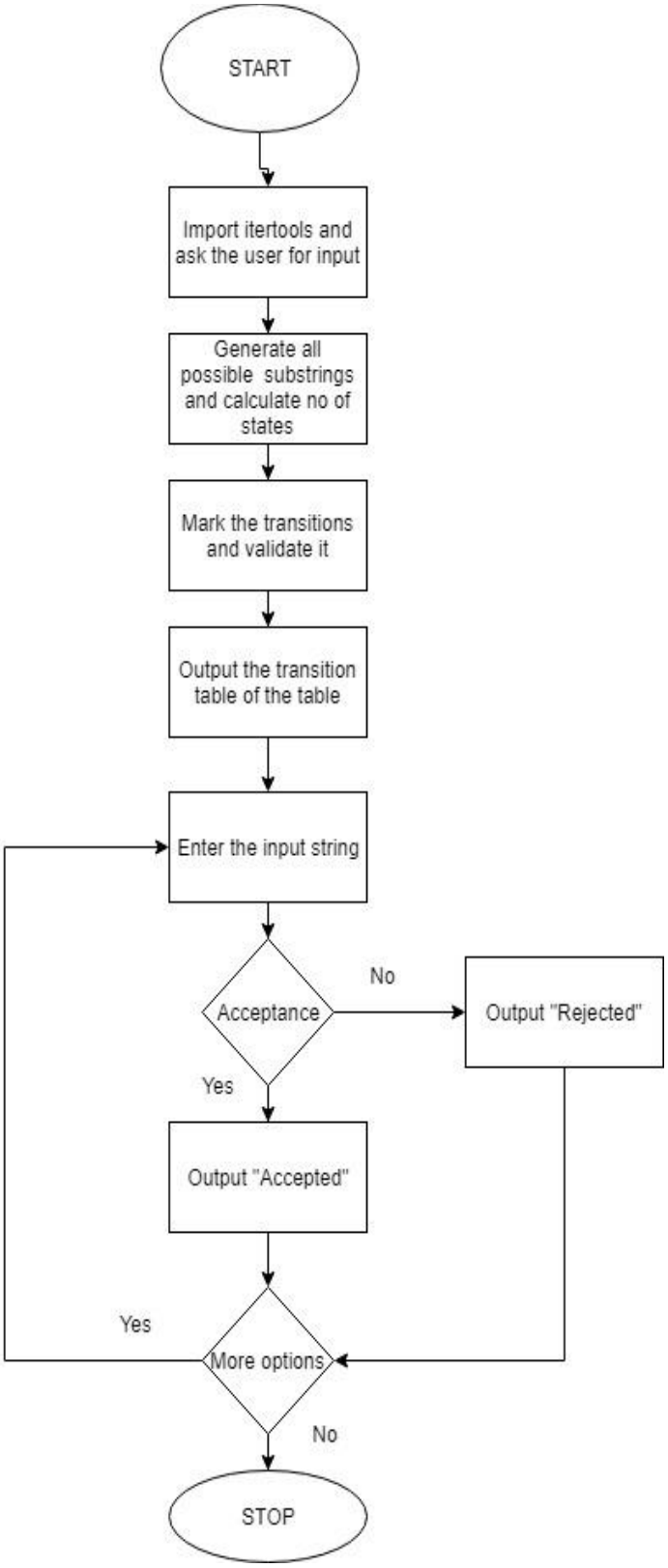
Expected Output:

When the program will complete its execution the transition table will be generated for the deterministic finite automata (i.e. DFA). The table will have the columns state and the input symbol. A row is dedicated for each state and the transitions. This table can be used for further processing using the DFA.

Algorithm:

1. START
2. Import the itertools library of python.
3. Ask the user to input the substring with which the string ends.
4. Generate all possible substrings with the input alphabet.
5. Calculate the number of states.
6. Make the transitions and validate those transitions.
7. Set the transitions and go for string processing.
8. If the string is accepted then
    a. Output “Accepted”
9. Else output “Rejected”
10. Ask the user if there are any more operations to be performed in the code because according to this option next step will be executed.
11. If the user input Y goto STEP6 else proceed.
12. STOP

Flowchart:


START
Import itertools and ask the user for input
Generate all possible substrings and calculate no of states
Mark the transitions and validate it
Output the transition table of the table
Enter the input string
Acceptance
No
Output "Rejected"
Yes
Output "Accepted"
Yes
More options
No
STOP

Source code and output:

jupyter CDlab2 Last Checkpoint: 3 hours ago (autosaved)

Python program to generate a generalised DFA

Importing Libraries

In [1]:
```
import itertools
```

Input

In [2]:
```
print("Enter ending with string :")
st=input()
symbol=['a','b']
for i in range(len(st)):
    if st[i] not in symbol:
        symbol.append(st[i])
```

```
Enter ending with string :
abb
```

Generating possible strings

In [3]:
```
st1=''
for s in symbol:
    st1=st1+s
print(st1)
possiple_strings=[]
for string in map(''.join, itertools.product(st1, repeat=len(st))):
    possiple_strings.append(string.lower())
possiple_strings.remove(st)
```

```
ab
```

jupyter CDlab2 Last Checkpoint: 3 hours ago (autosaved)

Calculation of states

In [4]:
```
nstates=len(st)+1
states=[]
for i in range(nstates):
    states.append('q'+str(i))

initial=states[0]
final=states[len(st)]

print("States used :"+str(states))
```

```
States used :['q0', 'q1', 'q2', 'q3']
```

Function to add transitions

In [5]:
```
transition={}
def setTrans(states,st):
    for i in range(len(st)):
        var=(states[i],st[i])
        transition[var]=states[i+1]

setTrans(states,st)

temptransition={}
temptransition=transition.copy()
```

Checking the validity of transitions

In [6]:
```
def process(teststr,transitions,startstate,i,new,sym):
    newtransitionset={}
    if teststr[0]==sym:
        newinitial=i
    else:
```

jupyter CDlab2 Last Checkpoint: 3 hours ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

```
        newinitial=new
    # print("initial "+newinitial)
     start=startstate
     n=0
     if teststr=='':
         return
     var1=transitions.keys()
     while start < nstates:
         for k in var1:
             if states[start] in k:
                 newtransitionset[k]=transitions[k]
         start+=1
    # print(newtransitionset)
     trans=(newinitial,teststr[0])
     loopcount=len(teststr)
     var2=newtransitionset.keys()
     for i in range(1,loopcount):
         if teststr[i]==='':
             return
         for k in var2:
             if k==trans:
                 change=newtransitionset[k]
                 trans=(newtransitionset[k],teststr[i])
                 #print(change)
                 #print(trans)
                 n+=1
                 break;

     for k in var2:
         if k==trans:
             change=newtransitionset[k]
             n+=1
             break;

     if n>=len(teststr):
        return change
     else:
         return
```

jupyter CDlab2 Last Checkpoint: 3 hours ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Function to set transition

In [7]:

```
def Transtions(states,symbol):
    for i in range(nstates):
        newi=nstates-i-1
        count=newi+1
        for j in range(len(symbol)):
            key=transition.keys()
            var=(states[newi],symbol[j])
            if var not in key:
                if newi==0:
                    transition[var]=states[newi]
                    temptransition[var]=states[newi]
                else:
                    if var not in key:
                        for n in range(count):
                            #print(states[newi-n])
                           # if i == newi - 1 and states[newi] == states[newi - n]:
                            #    continue;
                            temptransition[var]=states[newi-n]
                            #print(temptransition)
                            value=process(st,temptransition,newi-n,states[newi],states[newi-n],symbol[j])
                            val=[]
                            for p in possiple_strings:
                                val.append(process(p,temptransition,newi-n,states[newi],states[newi-n],symbol[j]))
                            #print(final+","+str(value))
                            if value==final and final not in val:
                                 transition[var]=states[newi-n]
                                 break;
                            else:
                                del temptransition[var]

Transtions(states,symbol)

sorted(transition,key=transition.__getitem__)
print( "**********************Transitions:**********************")
print(transition)

key=transition.keys()
```

jupyter CDlab2 Last Checkpoint: 3 hours ago (autosaved)

```
key=transition.keys()

#transitions a and b  aplphabet
trans_A=[]
trans_B=[]
for each in key:
    if 'a' in each:
        trans_A.append(each)

    else:
        trans_B.append(each)

print("********** Transition Table **********")
print("States    ",end='')
symbol.sort()
for i in range(len(symbol)):
    print(symbol[i]+"     ",end='')

print("")

for i in range(nstates):
    if states[i] == final:
        print("**" + states[i] + "     ", end='')
        for k in trans_A:
            if states[i] in k:
                print(transition[k] + "     ", end='')

        for k in trans_B:
            if states[i] in k:
                print(transition[k] + "     ", end='')
        print("")


    elif states[i]==initial:
        print("->"+states[i] + "     ", end='')
        for k in trans_A:
            if states[i] in k:
                print(transition[k] + "     ", end='')

        for k in trans_B:
```

jupyter CDlab2 Last Checkpoint: 3 hours ago (autosaved)

```
        for k in trans_B:
            if states[i] in k:
                print(transition[k] + "     ", end='')

        print("")

    else:
        print("  "+states[i]+"     ", end='')
        for k in trans_A:
            if states[i] in k:
                print(transition[k] + "     ", end='')

        for k in trans_B:
            if states[i] in k:
                print(transition[k] + "     ", end='')

        print("")
```

```
**********************Transitions:**********************
{('q0', 'a'): 'q1', ('q1', 'b'): 'q2', ('q2', 'b'): 'q3', ('q3', 'a'): 'q1', ('q3', 'b'): 'q0', ('q2', 'a'): 'q1', ('q1', 'a'):
'q1', ('q0', 'b'): 'q0'}
********** Transition Table **********
States    a     b
->q0     q1     q0
  q1     q1     q2
  q2     q1     q3
**q3     q1     q0
```

String Processing

In [8]:

```
def processstr(teststr):
    if teststr==':
        return
    var1=transition.keys()
    trans=(initial,teststr[0])
    loopcount=len(teststr)
    for i in range(1,loopcount):
        for k in var1:
```

jupyter CDlab2 Last Checkpoint: 3 hours ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

String Processing

In [8]:
```
def processstr(teststr):
    if teststr==':
        return
    var1=transition.keys()
    trans=(initial,teststr[0])
    loopcount=len(teststr)
    for i in range(1,loopcount):
        for k in var1:
            if k==trans:
                change=transition[k]
                trans=(transition[k],teststr[i])
                #print(change)
                #print(trans)
                break;

    for k in var1:
        if k==trans:
            change=transition[k]
            break;
    return change

while(True):
     str=input("Enter string to process :")
     result=processstr(str)

     if result == final:
         print("ACCEPTED ")

     else:
        print("Rejected ")
     ans=input("Want to try more ?(Y/N)")
     if(ans=='y'or ans=='Y'):
         continue;

     else:
         break;
```

jupyter CDlab2 Last Checkpoint: 3 hours ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

```
                break;

    for k in var1:
        if k==trans:
            change=transition[k]
            break;
    return change

while(True):
     str=input("Enter string to process :")
     result=processstr(str)

     if result == final:
         print("ACCEPTED ")

     else:
        print("Rejected ")
     ans=input("Want to try more ?(Y/N)")
     if(ans=='y'or ans=='Y'):
         continue;

     else:
         break;
```

```
Enter string to process :abbabababbbaabbaab
Rejected
Want to try more ?(Y/N)y
Enter string to process :abbababbababababbaabbabb
ACCEPTED
Want to try more ?(Y/N)n
```

In [ ]:

Frequently Asked Questions(FAQs)

Q1. Define Kleen Star.

Ans. Given Σ, then the Kleene Star Closure of the alphabet Σ, denoted by Σ*, is the collection of all strings defined over Σ, including Λ. It is to be noted that Kleene Star Closure can be defined over any set of strings.

Q2. What is meant by equivalent FAs?.

Ans. FAs that accept the same set of languages are called Equivalent FAs.

Q3. Define a regular expression.

Ans. These are mathematical expressions used to describe regular languages. In compiler design regular expressions are used for description of tokens.

Q4. Which machine is accepts the strings of regular languages?

Ans. Finite Automatas.

Q5. How many states will be there in a DFA constructed for acceptance of string starting with ab?

Ans. Four.

Practical-3 Feburary 06, 2018

Aim:

Write a program to find and locate the comments in given program.

Input:

This program is written in python and it asks the user for the path for the file containing the program to be checked. The file which contains the code can be in any format. This program is designed to check the comments in the code written in python language only. Therefore the input file should contain the source code in python language.

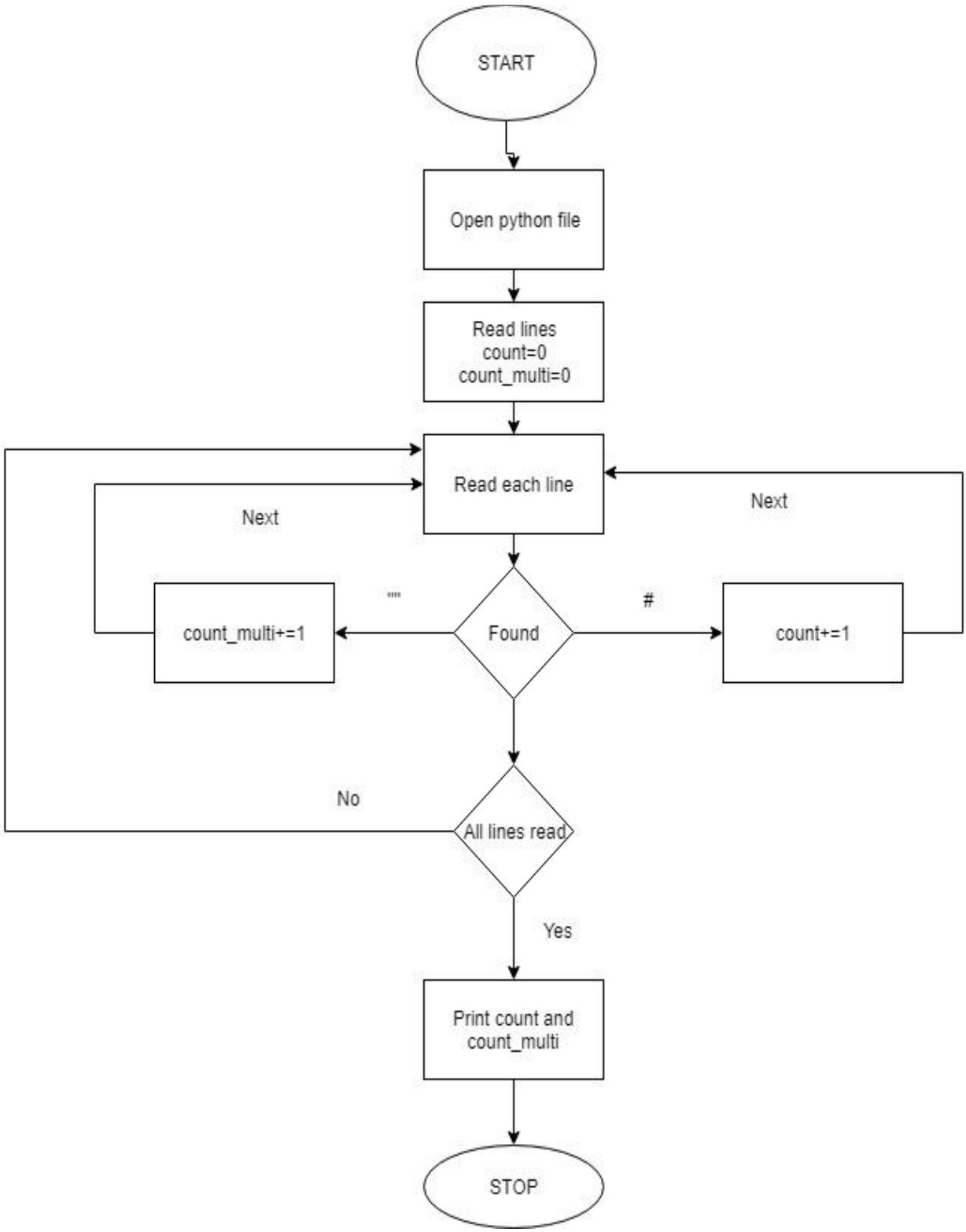
Expected Output:

The program that the file contains will have two types of comments, the single line or multiline comments. When the program will be executed it will check the source code for both these types of comments and the output of this program will contain single and multiline comments present in the program. The final output of this program will show the number of these types of comments.

Algorithm:

1. START
2. Open the python file which contains the source code to be evaluated. The path of this file is provided as an input to the code.
3. Set count (for single line comments) and count_multi (for multiple comments) equal to zero. Further on encounter of these types of comments both of these counters will be incremented.
4. For each line of the program.
   a. if we find " # " increment count by 1 and read the next line.
   b. Else if we find " ''' " increment count_multi by 1 and read the next line.
   c. Else read the next line.
5. When all lines are read print count and count_multi which indicate the number of single line comments and the number of multiple line comments respectively.
6. STOP

Flowchart


START
Open python file
Read lines
count=0
count_multi=0
Read each line
Next
Next
count_multi+=1
""
Found
#
count+=1
No
All lines read
Yes
Print count and
count_multi
STOP

Source code and output:

jupyter comment Last Checkpoint: 5 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Python program to find the number of single and multiline comments in the given program

In [8]:
```
from tkinter import*
from tkinter import filedialog
```

In [9]:
```
mgui=Tk()

mgui.geometry('450x450+200+200')
mgui.title('Count Comment')
```

Out[9]: ''

Code segment to calculate the number of comments

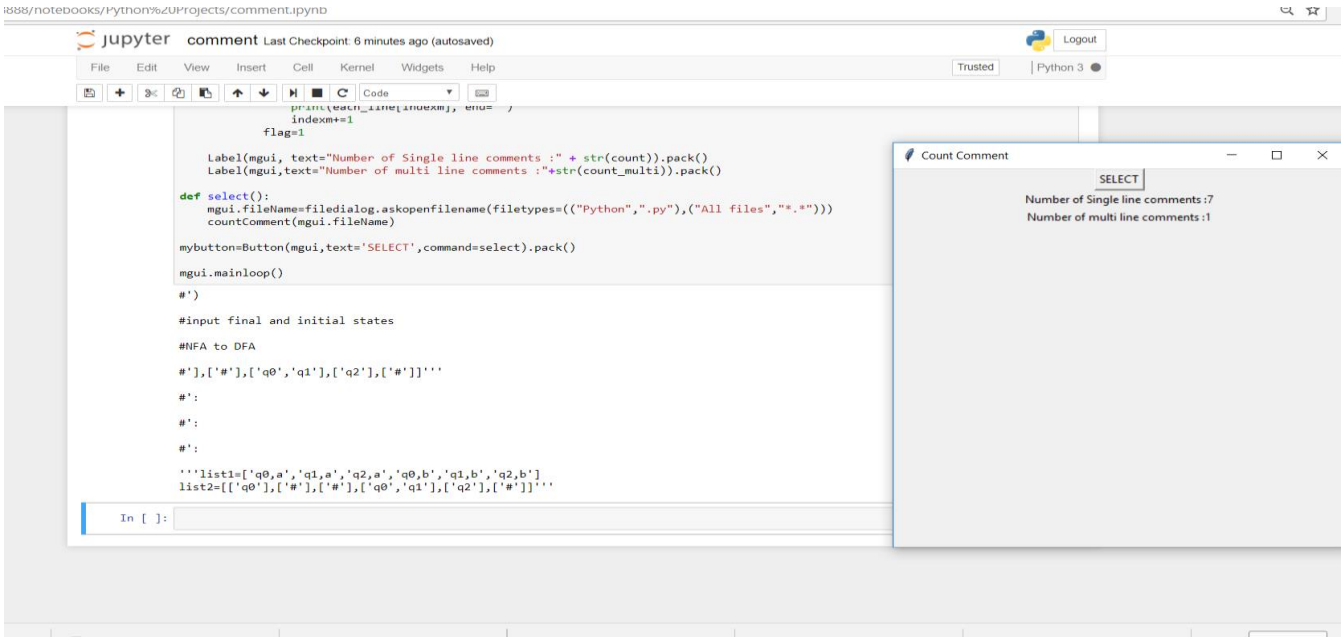
In [*]:
```
def countComment(filename):
    file=open(filename)
    lines=file.readlines()
    count=0
    count_multi=0
    temps=''
    comment="'''"
    flag=0
    for each_line in lines:

        if '#' in each_line:
            count+=1
            index=each_line.find('#')
            while index<len(each_line):
                print(each_line[index],end='')
```

jupyter comment Last Checkpoint: 6 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
            while index<len(each_line):
                print(each_line[index],end='')
                index+=1
            print("")
    str_print=''
    for each_line in lines:
        if flag==1:
            if comment not in each_line:
                print(each_line, end='')
                continue;
            elif comment in each_line:
                print(each_line, end='')
                flag=0
            continue;

        if comment in each_line and flag==0:
            count_multi+=1
            indexm=each_line.find(comment)
            while indexm < len(each_line):
                str_print+=each_line
                print(each_line[indexm], end='')
                indexm+=1
            flag=1

    Label(mgui, text="Number of Single line comments :" + str(count)).pack()
    Label(mgui,text="Number of multi line comments :"+str(count_multi)).pack()

def select():
    mgui.fileName=filedialog.askopenfilename(filetypes=(("Python",".py"),("All files","*.*")))
    countComment(mgui.fileName)

mybutton=Button(mgui,text='SELECT',command=select).pack()

mgui.mainloop()
```

```
#')

#input final and initial states

#NFA to DFA
```



Frequently Asked Questions(FAQs)

Q1. Which part in the code is skipped during the process of compilation and execution?

Ans. Comments are skipped during compilation and execution.

Q2. Which symbol is used to write single line comment in python?

Ans. '#' is used to specify single line comments in python.

Q3.Can we use a single # symbol to denote multi line comments?

Ans. No.

Q4. Comment on the case sensitivity of comments.

Ans. Comments used in code are not case sensitive.

Q5. Is it necessary to have comments for proper execution of the code?

Ans. Comments are not necessary but highly preferable.

Practical-4 Feburary 13, 2018

Aim:

Write a program to check for the lexemes from the code written in the file and classify it under various tokens.

Input:

Input to this program is provided by the user in the form of the path of file containing the code from which the lexemes are to be classified under various token categories.

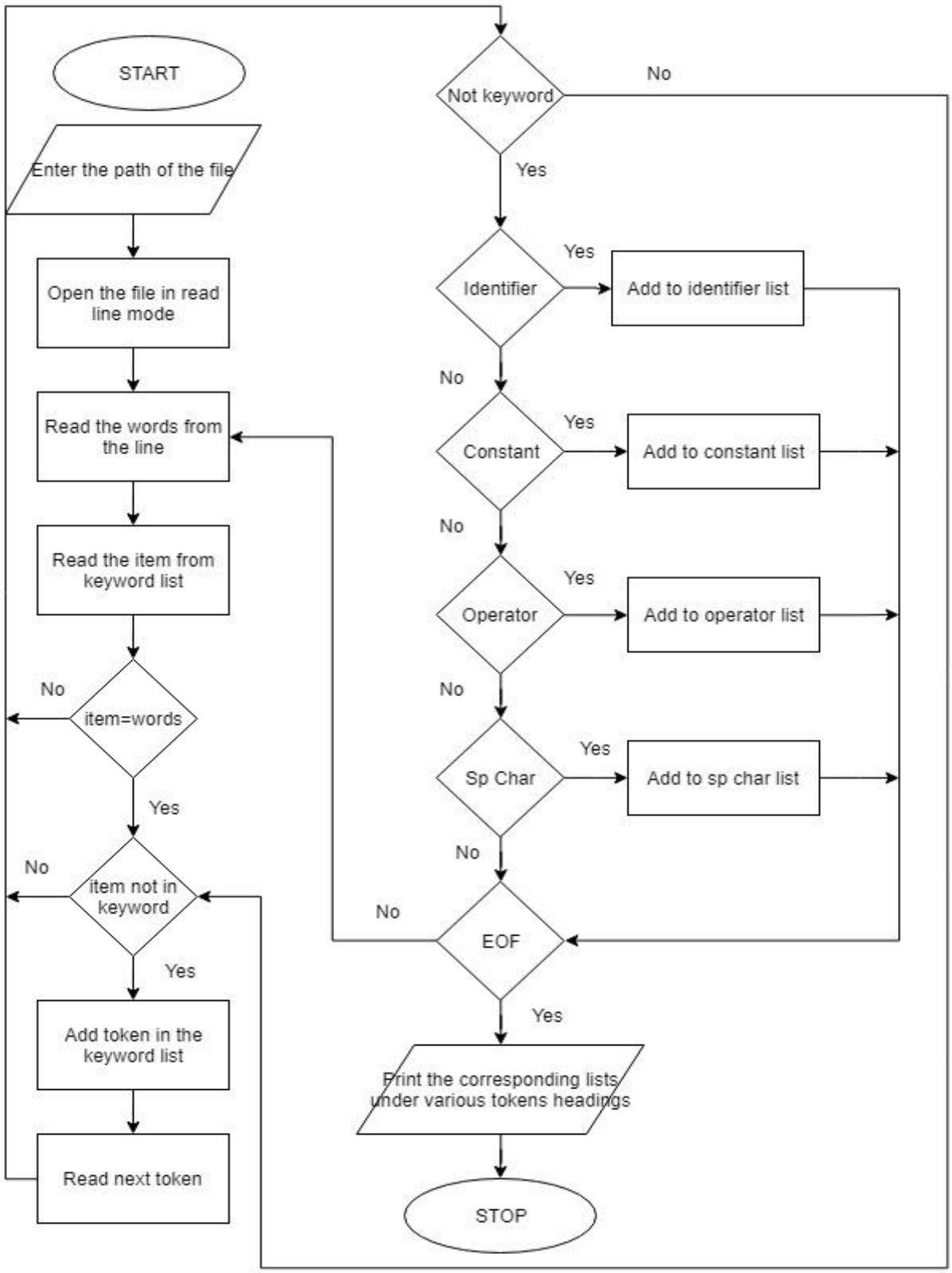
Expected Output:

On execution this program will open the file using the path specified and checks for the lexemes under various token categories and print the lexemes found as an output of the program.

Algorithm:

1. START
2. Define the lists of tokens for the given programing language.
3. Open the file containing the code in readline mode.
4. Repeat thru STEP5 for each line in file.
5. Repeat thru STEP9 for each word in line.
6. For each item in keyword list
    b. If word = item and word is not in keyword_found
    c. then add the word to list keyword_found
7. If word not in keyword_found then proceed
    d. If word[0]=' _' or word[0] is letter
    e. then add the word in list identifier_found
8. for each letter in word if letter is digit continue else break
    f. add the word in list constant_found
9. If word is operator then add the word in list operator_found
    g. Else add it to the spchar_found
10. Print all the lists
11. STOP

Flowchart:


START
Enter the path of the file
Open the file in read line mode
Read the words from the line
Read the item from keyword list
item=words
No
Yes
item not in keyword
No
Yes
Add token in the keyword list
Read next token
Not keyword
No
Yes
Identifier
Yes
Add to identifier list
No
Constant
Yes
Add to constant list
No
Operator
Yes
Add to operator list
No
Sp Char
Yes
Add to sp char list
No
EOF
No
Yes
Print the corresponding lists under various tokens headings
STOP

Source code and output:

jupyter lexemefile Last Checkpoint: 02/26/2018 (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [1]:
```
import numpy as np
```

In [2]:
```
keywords=["void","int","float","char","double","main"]
char=["A","B","C","D","E","F","G","H","I","J","K","L","M","N","O","P","Q","R","S","T","U","V","W","X","Y","Z","a","b","c","d","e"
digits=["0","1","2","3","4","5","6","7","8","9"]
operators=["+","-","/","*","=","==","(",")"]
spchar=["&","|","@","#","!","{","}",":",";"]
keyword_found=[]
identifier_found=[]
constant_found=[]
operator_found=[]
spchar_found=[]
```

In [3]:
```
file_name="code.txt"
j=0
file_text=open(file_name).readlines()
for lines in file_text:
    words=lines.split()
    for keyitem in keywords:
        for wordsitem in words:
            if keyitem == wordsitem:
                if wordsitem not in keyword_found:
                    keyword_found.append(wordsitem)
    for identi in words:
        if identi not in keyword_found:
            if identi[0] == "_" or identi[0] in char:
                if identi not in identifier_found:
                    identifier_found.append(identi)
    for const in words:
            for i in const:
                if i not in char and i not in operators and i not in spchar and i in digits:
                    if const not in constant_found:
                        constant_found.append(const)
                else:
                    break
    for opt in words:
        if opt not in keyword_found and opt not in identifier_found and opt not in constant_found:
            if opt in operators:
```

jupyter lexemefile Last Checkpoint: 02/26/2018 (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

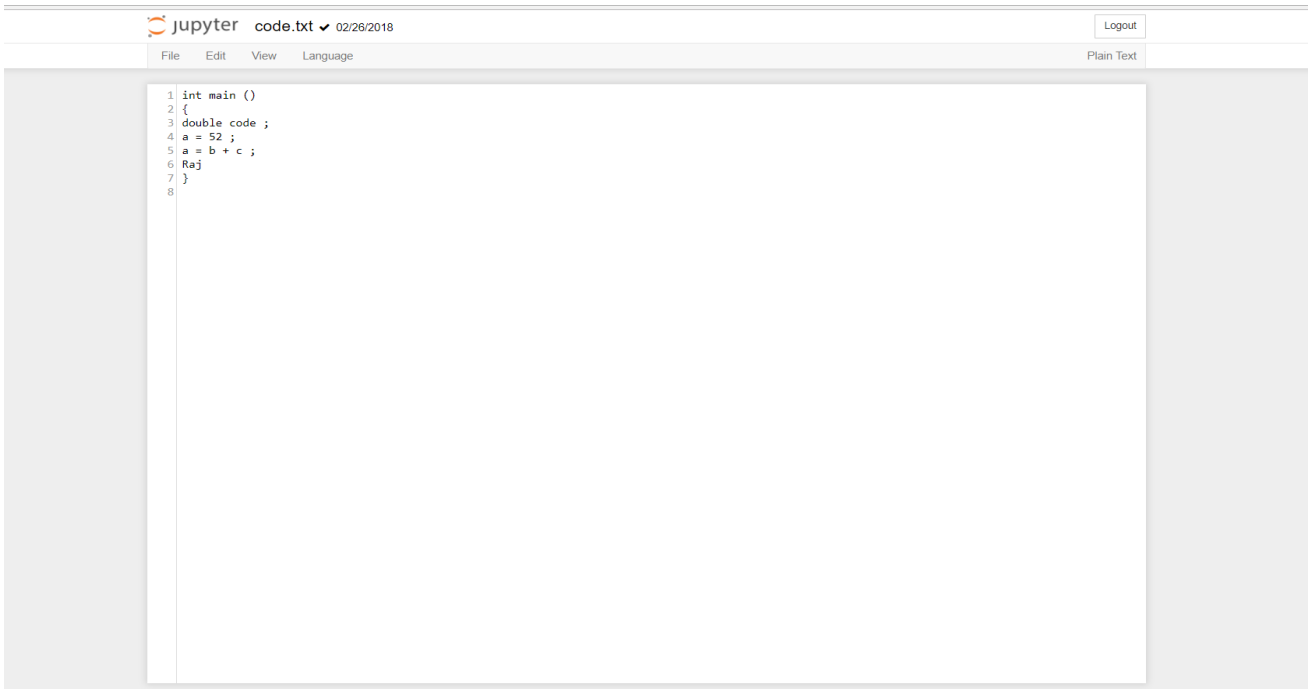
```
            for i in const:
                if i not in char and i not in operators and i not in spchar and i in digits:
                    if const not in constant_found:
                        constant_found.append(const)
                else:
                    break
    for opt in words:
        if opt not in keyword_found and opt not in identifier_found and opt not in constant_found:
            if opt in operators:
                if opt not in operator_found:
                    operator_found.append(opt)
            else:
                if opt not in spchar_found:
                    spchar_found.append(opt)
```

In [5]:
```
print("Keywords Present  ", end=' ')
print(' '.join(keyword_found))
print("Identifiers Present   ", end=' ')
print(' '.join(identifier_found))
print("Constants Present   ", end=' ')
print(' '.join(constant_found))
print("Operators Present   ", end=' ')
print(' '.join(operator_found))
print("Special Characters Present   ", end=' ')
print(' '.join(spchar_found))
```

```
Keywords Present   int main double
Identifiers Present    code a b c Raj
Constants Present    52
Operators Present    = +
Special Characters Present    () { ; }
```

In [ ]:

Text file used is shown on the next page.



Frequently Asked Questions(FAQs)

Q1. What is the output of lexical analysis phase?

Ans. Stream of tokens.

Q2. Lexical Analyzer belongs to synthesis phase of compiler? (True/False)

Ans. False.

Q3.What does instance of token known as?

Ans. Lexeme.

Q4. The output of lexical analysis phase is further fed to which phase of compiler?

Ans. Syntax and semantic analyzer.

Q5. What are keywords?

Ans. Keywords are the tokens whose definition is known to the compiler.

Practical-5

Feburary20, 2018

Aim:

Write a program to check for the lexemes from the code entered by the user and classify it under various token categories.

Input:

The input to this program is the source code entered line by line by the user during the execution of the program.

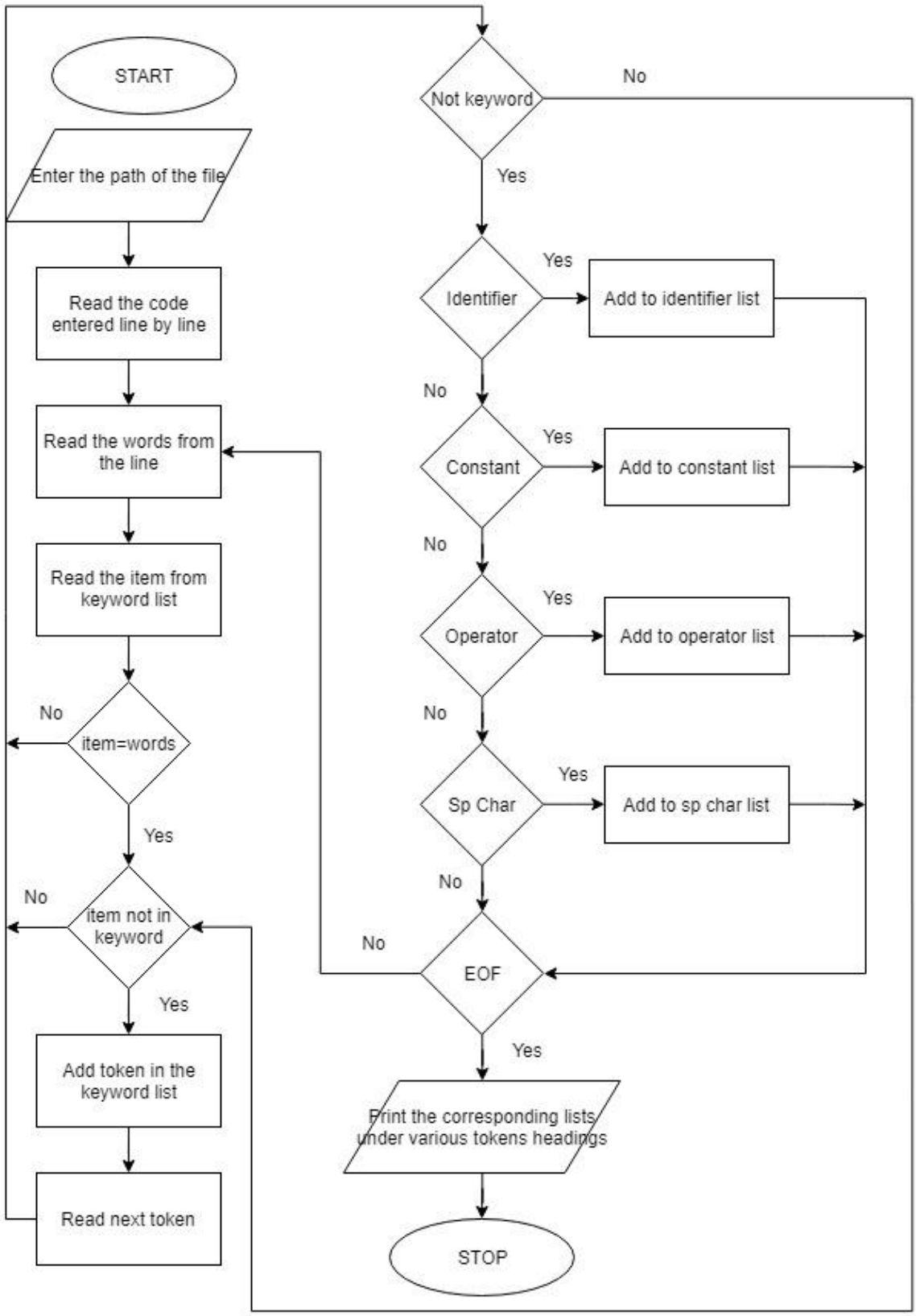
Expected Output:

When the user enters the program this code will identify and classify the lexes under various token categories and finally print these lexemes.

Algorithm:

1. START
2. Define the lists of tokens for the given programing language.
3. Ask the user to enter the code line by line
4. Repeat thru STEP5 for each line in file.
5. Repeat thru STEP9 for each word in line.
6. For each item in keyword list
    a. If word = item and word is not in keyword_found
    b. then add the word to list keyword_found
7. If word not in keyword_found then proceed
8. If word[0]=' _' or word[0] is letter
    a. then add the word in list identifier_found
9. for each letter in word if letter is digit continue else break
10. add the word in list constant_found
11. If word is operator then add the word in list operator_found
12. Else add it to the spchar_found
13. Print all the lists
14. STOP

Flowchart:


START
Enter the path of the file
Read the code entered line by line
Read the words from the line
Read the item from keyword list
item=words
No
Yes
item not in keyword
No
Yes
Add token in the keyword list
Read next token
Not keyword
No
Yes
Identifier
Yes
Add to identifier list
No
Constant
Yes
Add to constant list
No
Operator
Yes
Add to operator list
No
Sp Char
Yes
Add to sp char list
No
EOF
No
Yes
Print the corresponding lists under various tokens headings
STOP

Source code and output:

jupyter lexeme Last Checkpoint: 02/26/2018 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

In [6]:
```
import numpy as np
```

In [21]:
```
keywords=["void","int","float","char","double","main"]
char=["A","B","C","D","E","F","G","H","I","J","K","L","M","N","O","P","Q","R","S","T","U","V","W","X","Y","Z","a","b","c","d","e"
digits=["0","1","2","3","4","5","6","7","8","9"]
operators=["+","-","/","*","=","==","(",")"]
spchar=["&","|","@","#","!","{","}",":",";"]
keyword_found=[]
identifier_found=[]
constant_found=[]
operator_found=[]
spchar_found=[]
```

In [17]:
```
print("Please enter the code line by line:")
codelines = []

while True:
    choice=input("Is there a new line to be entered")
    if choice == 'y' or choice == 'Y':
        line = input("Enter the next line here")
        if line:
            codelines.append(line)
            text = '\n'.join(lines)
        else:
            break
    else:
        break

print(codelines)
```

```
Please enter the code line by line:
Is there a new line to be enteredy
Enter the next line hereint main ( )
Is there a new line to be enteredy
Enter the next line herea = 50 ;
Is there a new line to be enteredy
Enter the next line herea = b + c ;
Is there a new line to be enteredy
Enter the next line hereRaj
```

jupyter lexeme Last Checkpoint: 02/26/2018 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

In [22]:
```
for lines in codelines:
    for words in lines:
        for keyitem in keywords:
                if keyitem == words:
                    if words not in keyword_found:
                        keyword_found.append(words)
        if words not in keyword_found:
            if words[0] == "_" or words[0] in char:
                if words not in identifier_found:
                    identifier_found.append(words)
            for i in words:
                if i not in char and i not in operators and i not in spchar and i in digits:
                    if words not in constant_found:
                        constant_found.append(words)
                else:
                    break
        if words not in keyword_found and words not in identifier_found and words not in constant_found:
            if words in operators:
                if words not in operator_found:
                    operator_found.append(words)
            else:
                if words not in spchar_found:
                    spchar_found.append(words)
```

In [23]:
```
print("Keywords Present", end=' ')
print(' '.join(keyword_found))
print("Identifiers Present", end=' ')
print(' '.join(identifier_found))
print("Constants Present", end=' ')
print(' '.join(constant_found))
print("Operators Present", end=' ')
print(' '.join(operator_found))
print("Special Characters Present", end=' ')
print(' '.join(spchar_found))
```

```
Keywords Present
Identifiers Present i n t m a b c R j
Constants Present 5 0
Operators Present ( ) = +
Special Characters Present   ; }
```

Frequently Asked Questions(FAQs)

Q1. What does lexical analyzer do?

Ans. If the lexical analyzer finds a token invalid, it generates an error. The lexical analyzer works closely with the syntax analyzer. It reads character streams from the source code, checks for legal tokens, and passes the data to the syntax analyzer when it demands.

Q2. Differentiate between token and lexeme?

Ans.

- Lexemes are the words derived from the character input stream.
- Tokens are lexemes mapped into a token-name and an attribute-value..

Q3.Lexical analysis phase has nothing to do with Symbol Table. (True/False)

Ans. False.

Q4. The output of lexical analyzer phase is syntax tree of tokens. (True/False)

Ans. False.

Q5. Given an example each of keyword and a constant?

Ans. Keyword- int, Constant- 13.

Practical-6

Feburary 27, 2018

Aim:

Write a program to convert a given NFA to DFA.

Input:

The input to this program will be the complete information about the NFA (i.e. Non Deterministic Finite Automata), which includes the following:

- Number of states in the NFA and number of input symbols.
- Transition of each state on occurrence of each input symbol
- Initial and final states.

Expected Output:

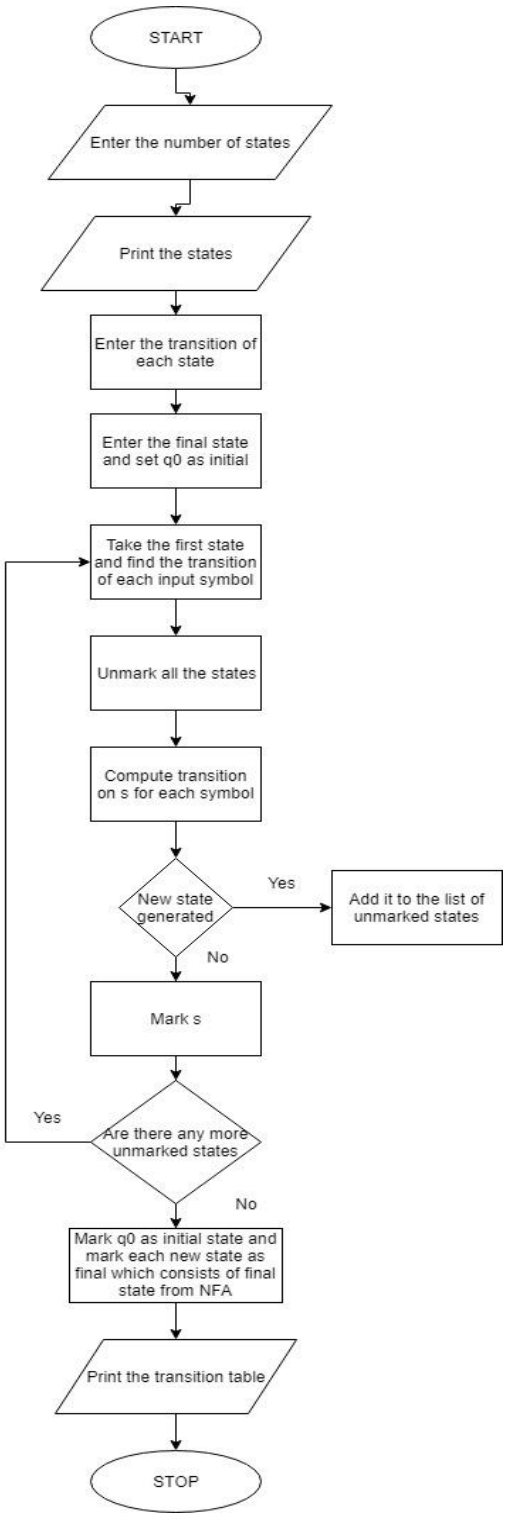
The expected output of this program is the Deterministic Finite Automata (DFA) equivalent to the given NFA that the user have provided the code as an input. The output of this program will have all the states in the DFA listed followed by the transition table of DFA created.

Algorithm:

a) START
b) Enter the number of states in the NFA.3. Print the states created in the NFA.
c) Enter transition of each state on occurrence of each input symbol.
d) Enter the final state and set q0 as initial state.
e) Take the first state i.e. q0 and check for all the transitions of each input symbol and by taking the union of all the states mark it as new state.
f) Unmark all the states.
g) for each unmarked state s
    i) Compute the transition of s on each input symbol. For each new state add it to the unmarked list.
    ii) Mark s
h) Mark q0 as initial state and mark all the new states that have final state of NFA as the new final state of DFA.
i) Print the transition table of newly generated DFA.

j) STOP

Flowchart:



Source Code and Output:

jupyter CDlab6 Last Checkpoint: 17 hours ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [4]:
```
print(" Transition for NFA That you want to convert Using states(q0,q1,....) and alphabet (a.b) :")
print("Enter Number Of States :")
nstates=int(input())
```

```
 Transition for NFA That you want to convert Using states(q0,q1,....) and alphabet (a.b) :
Enter Number Of States :
4
```

In [5]:
```
symbol=['a','b']
states_nfa=[]
for i in range(nstates):
    states_nfa.append('q'+str(i))
print("STATES")
print(states_nfa)
```

```
STATES
['q0', 'q1', 'q2', 'q3']
```

In [6]:
```
list1_nfa=[]
for i in range(len(symbol)):
    for j in range(nstates):
        list1_nfa.append(states_nfa[j]+","+symbol[i])
list2_nfa=[]
for i in range(len(list1_nfa)):
    print("Enter No. of transition for "+list1_nfa[i]+":")
    n=int(input())
    temp=[]
    if n==0:
        temp.append('#')
    for j in range(n):
```

jupyter CDlab6 Last Checkpoint: 17 hours ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
    for j in range(n):
        print("Enter state :")
        temp.append(input())
    list2_nfa.append(temp)
```

```
Enter No. of transition for q0,a:
1
Enter state :
q0
Enter No. of transition for q1,a:
0
Enter No. of transition for q2,a:
0
Enter No. of transition for q3,a:
0
Enter No. of transition for q0,b:
2
Enter state :
q0
Enter state :
q1
Enter No. of transition for q1,b:
1
Enter state :
q2
Enter No. of transition for q2,b:
1
Enter state :
q3
Enter No. of transition for q3,b:
0
```

jupyter CDlab6 Last Checkpoint: 17 hours ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

In [7]:
```
initial='q0'
print("Enter final state: ")
final=input()
```

```
Enter final state:
q3
```

In [8]:
```
#NFA to DFA
list1=list1_nfa.copy()
list2=list2_nfa.copy()

states=[]
symbol=['a','b']

listnfa1=[]
listnfa2=[]
checklist=[['q0']]
states=[['q0']]

for i in range(len(list1)):
    if 'q0' in list1[i]:
        listnfa1.append(list1[i])
        listnfa2.append(list2[i])
```

In [9]:
```
while True:
    for i in range(len(listnfa2)):
        if listnfa2[i] not in checklist:
            templist=[]
            templist=listnfa2[i].copy()
            print(templist)
            nfa1=[]
            nfa2=[]
```

jupyter CDlab6 Last Checkpoint: 17 hours ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

```
            nfa2=[]
            nfa3=[]
            nfa4=[]
            for j in range(len(templist)):
                for i in range(len(list1)):
                    var=str(templist[j]+",a")
                    var2=str(templist[j]+",b")
                    if var in list1[i]:
                        for k in range(len(list2[i])):
                            nfa1.append(list2[i][k])
                    elif var2 in list1[i]:
                        for k in range(len(list2[i])):
                            nfa3.append((list2[i][k]))
            for d in nfa1:
                if d not in nfa2 and d!='#':
                    nfa2.append(d)
            nfa1.sort()
            nfa2.sort()
            for d in nfa3:
                if d not in nfa4 and d!='#':
                    nfa4.append(d)
            nfa3.sort()
            nfa4.sort()
            listnfa1.append(str(templist)+",a")
            listnfa2.append(nfa2)
            listnfa1.append(str(templist) + ",b")
            listnfa2.append(nfa4)
            checklist.append(templist)
            if templist not in states:
                states.append(templist)
    tempchecklist=[]
    for d in listnfa2:
        if d not in tempchecklist and d != '#':
```

jupyter CDlab6 Last Checkpoint: 17 hours ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Code

```
    for d in listnfa2:
        if d not in tempchecklist and d != '#':
            tempchecklist.append(d)
    checklist.sort()
    tempchecklist.sort()
    if checklist==tempchecklist:
        break
print("********** Transition Table **********")
print("States"+" "*23,end='')
symbol.sort()
for i in range(len(symbol)):
    print(symbol[i]+" "*(26-(len(str(symbol[i])))),end='')
print("")
count=0
for i in range(len(states)):
    if i==0:
        print("->"+str(states[i])+" "*(23-(len(str(states[i]))))),end='')
        print("  "+str(listnfa2[count])+" "*(20-(len(str(listnfa2[count]))))),end='')
        print("  "+str(listnfa2[count+1]),end='')
        print("")
        count+=2
    elif final in states[i]:
        print("**" + str(states[i]) + " " * (23 - (len(str(states[i])))), end='')
        print("  " + str(listnfa2[count]) + " " * (20 - (len(str(listnfa2[count])))), end='')
        print("  " + str(listnfa2[count + 1]), end='')
        print("")
        count += 2
    else:
        print("  "+str(states[i]) + " " * (23 - (len(str(states[i])))), end='')
        print("  "+ str(listnfa2[count]) + " " * (20 - (len(str(listnfa2[count])))), end='')
        print("  "+ str(listnfa2[count + 1]), end='')
        print("")
        count += 2
```

jupyter CDlab6 Last Checkpoint: 17 hours ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Code

```
        print("")
        count+=2
    elif final in states[i]:
        print("**" + str(states[i]) + " " * (23 - (len(str(states[i])))), end='')
        print("  " + str(listnfa2[count]) + " " * (20 - (len(str(listnfa2[count])))), end='')
        print("  " + str(listnfa2[count + 1]), end='')
        print("")
        count += 2
    else:
        print("  "+str(states[i]) + " " * (23 - (len(str(states[i])))), end='')
        print("  "+ str(listnfa2[count]) + " " * (20 - (len(str(listnfa2[count])))), end='')
        print("  "+ str(listnfa2[count + 1]), end='')
        print("")
        count += 2
```

```
['q0', 'q1']
['q0', 'q1', 'q2']
['q0', 'q1', 'q2', 'q3']
********** Transition Table **********
States                        a                        b
->['q0']                   ['q0']                ['q0', 'q1']
  ['q0', 'q1']             ['q0']                ['q0', 'q1', 'q2']
  ['q0', 'q1', 'q2']       ['q0']                ['q0', 'q1', 'q2', 'q3']
**['q0', 'q1', 'q2', 'q3']  ['q0']                 ['q0', 'q1', 'q2', 'q3']
```

In [ ]:

Frequently Asked Questions(FAQs)

Q1 Fill in the blank.

.NFA _____ has multiple transitions for same input symbol.

Ans. may or may not

Q2. What is the full form of NFA.

Ans. Non Deterministic Finite Automata.

Q3.What do you mean by equivalence of NFA and DFA?

Ans. Converting a given NFA to a DFA is known as equivalence of NFA and DFA.

Q4. NFA can have zero transition on a state for a given input symbol. (True/False)

Ans. True.

Q5. NFA can never be converted to a DFA. (True/False)

Ans. False.

Practical-7 March 06, 2018

Aim:

Write a program to derive a string from the rules of given grammar by the means of left most and right most derivation (i.e. LMD and RMD).

Input:

The input to this program will be the complete grammar along with the string to be generated, which includes the following:

- Non terminals present in the grammar.
- Terminals present in the grammar.
- Start symbol for string derivation.
- Production rules of the grammar.

Expected Output: The expected output of this program consists of two subparts, one is the string derivation using the Left Most Derivation (LMD) and another is the string derivation using the Right Most Derivation (RMD) .

Algorithm:

1. START
2. Input grammar from a txt file or manually take input from user and string.
3. Create reference table for the grammar.
    a. Repeat through step c for each production rule.
    b. Calculate FIRST of right hand side of production rule and if right hand is empty (null) calculate FOLLOW of left hand side of production and store it in index.
    c. Place the rule in table with row = left hand side of production and column = all the terminals in the FIRST list.
4. Derivation of string using LMD.
    a. Place start symbol in a variable stack and string in variable str with '$' at the end of string.
    b. Initialize i=0,j=0 and generation=[]
    c. Repeat through d until str[j]!='$'

d. If stack[i]=str[j]
   i. i =i+1
   ii. j =j+1

   Else

   i. Replace stack[i] with right hand side of production rule present in table at location table(stack[i],str[j])
   ii. generation.append(stack)

5. print generation list.
6. Derivation of string using RMD
   a. Place start symbol in a variable stack and store string in variable str in reverse order with '$' at the end of string.
   b. Initialize i=0,j=len(stack)-1 and generation=[]
   c. Repeat through d until stack!=str
   d. If no entry for (stack[j],str[i]) in table and str[i] is terminal
      i. j = j-1

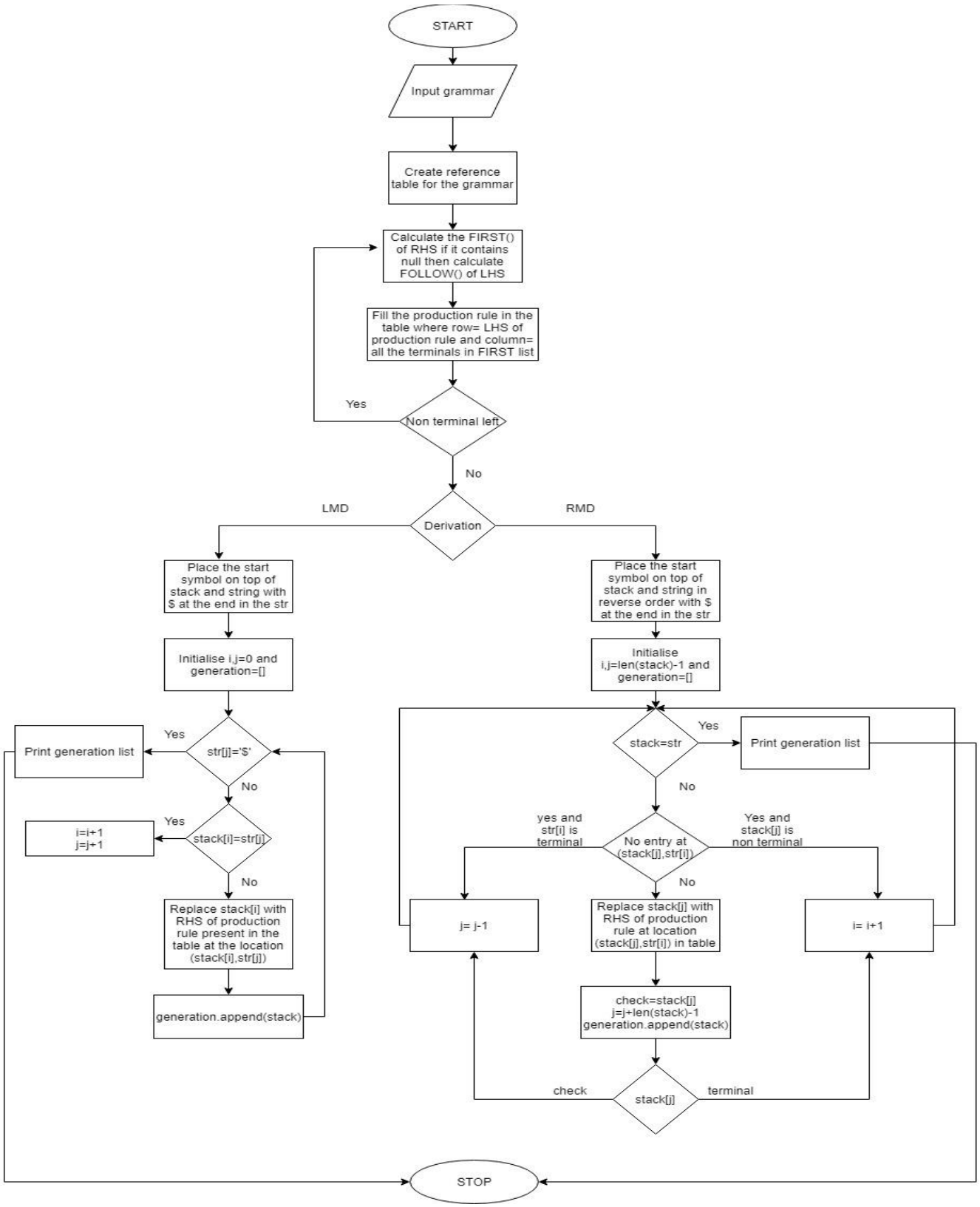
      Else if no entry for (stack[j],str[i]) in table and stack[j] is not terminal

      i. i = i+1

      Else if there is entry for (stack[j],str[i]) in table

      i. Replace stack[j] with right hand side of production rule present at location (stack[j],str[i])
      ii. Check=stack[j]
      iii. j = j + len(stack) – 1
      iv. generation.append(stack)
      v. If satck[j]== check then i = i + 1
      vi. If stack[j] is terminal then j = j – 1
7. Output generation list
8. STOP

Flowchart:


START
Input grammar
Create reference table for the grammar
Calculate the FIRST() of RHS if it contains null then calculate FOLLOW() of LHS
Fill the production rule in the table where row= LHS of production rule and column= all the terminals in FIRST list
Yes
Non terminal left
No
LMD
Derivation
RMD
Place the start symbol on top of stack and string with $ at the end in the str
Place the start symbol on top of stack and string in reverse order with $ at the end in the str
Initialise i,j=0 and generation=[]
Initialise i,j=len(stack)-1 and generation=[]
Yes
Print generation list
str[j]='$'
No
Yes
i=i+1
j=j+1
stack[i]=str[j]
No
Replace stack[i] with RHS of production rule present in the table at the location (stack[i],str[j])
generation.append(stack)
stack=str
Yes
Print generation list
No
yes and str[i] is terminal
No entry at (stack[j],str[i])
Yes and stack[j] is non terminal
No
j= j-1
Replace stack[j] with RHS of production rule at location (stack[j],str[i]) in table
i= i+1
check=stack[j]
j=j+len(stack)-1
generation.append(stack)
check
stack[j]
terminal
STOP

Source Code and Output:

jupyter CD string generation Last Checkpoint: 9 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

Program of string derivation using LMD and RMD

In [3]:
```
nonterminals=[]
terminals=[]
productions=[]
```

In [4]:
```
productions=['E->TA','A->+TA','A->~','T->FB','B->*FB','B->~','F->(E)','F->d']
nonterminals=['E','A','T','B','F']
terminals=['+','*','d','(',')']
startsymbol='E'

lhs=[]
rhs=[]
for rule in productions:
    lhs.append(rule.split('->')[0])
    rhs.append(rule.split('->')[1])
print(lhs)
print(rhs)
print("Non terminals  ")
print(nonterminals)
print("Terminals")
print(terminals)
derivation=[]
print("Production rule")
print(productions)
```

jupyter CD string generation Last Checkpoint: 10 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

```
['E', 'A', 'A', 'T', 'B', 'B', 'F', 'F']
['TA', '+TA', '~', 'FB', '*FB', '~', '(E)', 'd']
Non terminals
['E', 'A', 'T', 'B', 'F']
Terminals
['+', '*', 'd', '(', ')']
Production rule
['E->TA', 'A->+TA', 'A->~', 'T->FB', 'B->*FB', 'B->~', 'F->(E)', 'F->d']
```

In [5]:
```
def splitStr(str):
    stringList=[]
    temp=''
    for i in range(len(str)):
        temp=temp+str[i]
        if temp in terminals:
            stringList.append(temp)
            temp=''
    return stringList
```

In [6]:
```
str='d+d+d*d'
splitStr(str)
```

Out[6]:
```
['d', '+', 'd', '+', 'd', '*', 'd']
```

In [7]:
```
def firstFun(nonterminal):
    firstn=[]
    if nonterminal[0] in terminals:
        return nonterminal
    for rule in productions:
        lh=rule.split('->')[0]
        rh=rule.split('->')[1]
```

jupyter CD string generation Last Checkpoint: 10 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

```
        lh=rule.split('->')[0]
        rh=rule.split('->')[1]
        if lh==nonterminal:
            temp=''
            for i in range(len(rh)):
                temp=temp+rh[i]
                if temp in terminals:
                    firstn.append(temp)
                    temp=''
                    break
                elif temp =='~':
                    firstn.append('~')
                elif temp in nonterminals:
                    n=firstFun(temp)
                    firstn.extend(n)
                    if '~' in n and isupper(rh):
                        continue

    return firstn
```

In [8]:
```
first={}
```

In [9]:
```
for each in nonterminals:
    first[each]=firstFun(each)
```

In [10]:
```
print(first)
```

```
{'E': ['(', 'd'], 'A': ['+', '~'], 'T': ['(', 'd'], 'B': ['*', '~'], 'F': ['(', 'd']}
```

In [11]:
```
def followFun(nonterminal):
```

jupyter CD string generation Last Checkpoint: 10 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

In [11]:
```
def followFun(nonterminal):
    follown=[]
    for rule in productions:
        if nonterminal in rule.split('->')[1]:
            if nonterminal==startsymbol:
                follown.append('$')
            lh=rule.split('->')[0]
            rh=rule.split('->')[1]
            index=rh.find(nonterminal)
            if index<len(rh)-1:
                m=firstFun(rh[index+1])
                follown.extend(m)
                if '~' in m:
                    n=followFun(lh)
                    follown.extend(n)
                break
            if index==len(rh)-1:
                n=followFun(lh)
                follown.extend(n)
                break
    if '~' in follown:
        follown.remove('~')
    return follown
```

In [12]:
```
follow={}
```

In [13]:
```
for each in nonterminals:
    follow[each]=followFun(each)
```

In [14]:
```
print(follow)
```

jupyter CD string generation Last Checkpoint: 10 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

In [14]:
```
print(follow)
```

```
{'E': ['$', ')'], 'A': ['$', ')'], 'T': ['+', '$', ')'], 'B': ['+', '$', ')'], 'F': ['*', '+', '$', ')']}
```

In [15]:
```
def firstFun2(l,str):
    temp=''
    firstn2=[]
    if str[0] in terminals:
        firstn2.append(str[0])
        return firstn2
    if str=='~':
        firstn2.extend(follow[l])
        return firstn2
    else:
        for i in range(len(str)):
            temp=temp+str[i]
            if temp in terminals:
                firstn2.append(temp)
                return firstn2
            if str[i] in nonterminals:
                m=first[str[i]]
                firstn2.extend(m)
                if '~' in m :
                    continue
                else:
                    break

    return firstn2
```

In [16]:
```
print(firstFun2('E','TA'))
```

jupyter CD string generation Last Checkpoint: 10 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

In [16]:
```
print(firstFun2('E','TA'))
```

```
['(', 'd']
```

In [17]:
```
table={}
```

In [18]:
```
for i in range(len(lhs)):
    positions=firstFun2(lhs[i],rhs[i])
    for j in range(len(positions)):
        k=(lhs[i],positions[j])
        table[k]=productions[i]
```

In [19]:
```
table
```

Out[19]:
```
{('A', '$'): 'A->~',
 ('A', ')'): 'A->~',
 ('A', '+'): 'A->+TA',
 ('B', '$'): 'B->~',
 ('B', ')'): 'B->~',
 ('B', '*'): 'B->*FB',
 ('B', '+'): 'B->~',
 ('E', '('): 'E->TA',
 ('E', 'd'): 'E->TA',
 ('F', '('): 'F->(E)',
 ('F', 'd'): 'F->d',
 ('T', '('): 'T->FB',
 ('T', 'd'): 'T->FB'}
```

In [20]:
```
str1=splitStr(str)
str1.append('$')
```

jupyter CD string generation Last Checkpoint: 10 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [21]:
```
stack='E'
steps=[]
generation=[]
i=0
j=0
str1
```

Out[21]:
```
['d', '+', 'd', '+', 'd', '*', 'd', '$']
```

In [26]:
```
def generate(str1,stack,i,j):
    try:
       while True:
        if '~' in stack:
            stack=stack.replace('~','')
        if len(stack)==len(str1)-1 and str1[j]=='$':
            break

        if stack[i]=='~':
            stack=stack.replace('~','')


        #print(str[j])
        if stack[i]==str1[j]:
            print('match',i,j)
            #steps.append(table[(temp,str[j])])
            #str.remove(str[j])
            if str1[j]!='$':
                j=j+1
            i=i+1
            print(i,j)
            print(steps)
```

jupyter CD string generation Last Checkpoint: 11 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
            print(steps)
            print(new)
            print(stack)
            if '~' in stack:
                stack=stack.replace('~','')
            generation.append(stack)
    except:
        return 1
```

In [27]:
```
flag =generate(str1,stack,i,j)
```

```
0 0
['E->TA', 'T->FB', 'F->d', 'B->~', 'A->+TA', 'T->FB', 'F->d', 'B->~', 'A->+TA', 'T->FB', 'F->d', 'B->*FB', 'F->d', 'B->~', 'A-
>~', 'E->TA', 'T->FB', 'F->d', 'B->~', 'A->+TA', 'T->FB', 'F->d', 'B->~', 'A->+TA', 'T->FB', 'F->d', 'B->*FB', 'F->d', 'B->~',
'A->~', 'E->TA']
TA
TA
0 0
['E->TA', 'T->FB', 'F->d', 'B->~', 'A->+TA', 'T->FB', 'F->d', 'B->~', 'A->+TA', 'T->FB', 'F->d', 'B->*FB', 'F->d', 'B->~', 'A-
>~', 'E->TA', 'T->FB', 'F->d', 'B->~', 'A->+TA', 'T->FB', 'F->d', 'B->~', 'A->+TA', 'T->FB', 'F->d', 'B->*FB', 'F->d', 'B->~',
'A->~', 'E->TA', 'T->FB']
FBA
FBA
0 0
['E->TA', 'T->FB', 'F->d', 'B->~', 'A->+TA', 'T->FB', 'F->d', 'B->~', 'A->+TA', 'T->FB', 'F->d', 'B->*FB', 'F->d', 'B->~', 'A-
>~', 'E->TA', 'T->FB', 'F->d', 'B->~', 'A->+TA', 'T->FB', 'F->d', 'B->~', 'A->+TA', 'T->FB', 'F->d', 'B->*FB', 'F->d', 'B->~',
'A->~', 'E->TA', 'T->FB', 'F->d']
dBA
dBA
match 0 0
1 1
```

jupyter CD string generation Last Checkpoint: 11 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [29]:
```
generation
```

Out[29]:
```
['TA',
 'FBA',
 'dBA',
 'dA',
 'd+TA',
 'd+FBA',
 'd+dBA',
 'd+dA',
 'd+d+TA',
 'd+d+FBA',
 'd+d+dBA',
 'd+d+d*FBA',
 'd+d+d*dBA',
 'd+d+d*dA',
 'd+d+d*d',
 'TA',
 'FBA',
 'dBA',
 'dA',
 'd+TA',
 'd+FBA',
 'd+dBA',
 'd+dA',
 'd+d+TA',
 'd+d+FBA',
 'd+d+dBA',
 'd+d+d*FBA',
 'd+d+d*dBA',
 'd+d+d*dA'
```

jupyter CD string generation Last Checkpoint: 11 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [30]:
```
print('********************String Generation using LMD******************')
if flag==1:
    print('String can not be generated ')
else:
    print(startsymbol,'=>',generation[0],end='')
    print(" "*16,"[",steps[0],"]")
    print('')
    for i in range(len(steps)):
        print(' '*(len(startsymbol)),'=>',generation[i]," "*(18-len(generation[i])),end='')
        print("[",steps[i],"]")
        print('')
```

```
********************String Generation using LMD******************
E => TA                 [ E->TA ]

  => TA                 [ E->TA ]

  => FBA                [ T->FB ]

  => dBA                [ F->d ]

  => dA                 [ B->~ ]

  => d+TA               [ A->+TA ]

  => d+FBA              [ T->FB ]

  => d+dBA              [ F->d ]

  => d+dA               [ B->~ ]

  => d+d+TA             [ A->+TA ]
```

jupyter CD string generation Last Checkpoint: 11 minutes ago (autosaved)

```
=> d+dA              [ B->~ ]
=> d+d+TA            [ A->+TA ]
=> d+d+FBA           [ T->FB ]
=> d+d+dBA           [ F->d ]
=> d+d+d*FBA         [ B->*FB ]
=> d+d+d*dBA         [ F->d ]
=> d+d+d*dA          [ B->~ ]
=> d+d+d*d           [ A->~ ]
=> TA                [ E->TA ]
=> FBA               [ T->FB ]
=> dBA               [ F->d ]
=> dA                [ B->~ ]
=> d+TA              [ A->+TA ]
=> d+FBA             [ T->FB ]
=> d+dBA             [ F->d ]
=> d+dA              [ B->~ ]
```

jupyter CD string generation Last Checkpoint: 12 minutes ago (autosaved)

```
In [31]: generation=[]
         stepsrmd=[]
         stack1=[]
         stack='E'
         str1
         steps=[]

In [32]: strrmd=str1[::-1]
         strrmd.remove('$')
         strrmd.append('$')
         i=0
         j=len(stack)-1
         check=''
         k=table.keys()
         str

Out[32]: 'd+d+d*d'

In [33]: def funrmd(st,s,i,j):
             try:
                 stack=st
                 #strmd=s[::-1]
                 while True:
                     if stack==str:
                         print('Return',stack)
                         return stack
                     if (stack[j],s[i]) not in k and stack[j] in terminals and s[i] in terminals:
                         print(i,j)
                         j-=1
                         print(i,j)
                     elif (stack[j],s[i]) not in k and stack[j] not in terminals:
```

jupyter CD string generation Last Checkpoint: 12 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
            elif (stack[j],s[i]) not in k and stack[j] not in terminals:
                print('if',i,j)
                i+=1
                print(i,j)
            elif (stack[j],s[i]) in k:
                print('elif',i,j)
                print(s[i])
                val=table[(stack[j],s[i])].split('->')[1]
                steps.append(table[(stack[j],s[i])])
                val=val[::-1]
                check=stack[j]
                print('check ',check)
                stack=stack[::-1]
                print('VAL reversed ',val)
                print('Stack reversed ',stack)
                new=stack.replace(check,val,1)
                print('new ',new)
                stack=new[::-1]
                j=j+len(val)-1
                generation.append(stack)
                if stack[j]==check:
                    i+=1
                print(steps)
                print(stack)
                print(i,j)
                if stack[j] in terminals:
                    j=j-1
                if '~' in stack:
                    stack=stack.replace('~','')
                    j=j-1
                    i=0
```

jupyter CD string generation Last Checkpoint: 12 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
            print('new',i,j)
        except:
            print('except')
            return 2
```

In [34]:

```
def generatermd(stack,strrmd,i,j):
    while True:
        if '~' in stack:
            stack=stack.replace('~','')

        #print(stack[j],i,j,strrmd[i])
        if stack[-1] in terminals:
            print('FUN')
            s=str1[::]
            flag=funrmd(stack,s,i,j)
            return

        elif (stack[j],strrmd[i]) not in k :
            print('if',i,j)
            i+=1
            print(i,j)
            #val=table[(stack[j],strrmd[i])].split('->')[1]
            #steps.append(table[(stack[j],strrmd[i])])

        elif (stack[j],strrmd[i]) in k:
            print(steps)
            print('elif',i,j)
            val=table[(stack[j],strrmd[i])].split('->')[1]
            steps.append(table[(stack[j],strrmd[i])])
            val=val[::-1]
            check=stack[j]
```

jupyter CD string generation Last Checkpoint: 12 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
            check=stack[j]
            print('check ',check)
            stack=stack[::-1]
            print('VAL reversed ',val)
            print('Stack reversed ',stack)
            new=stack.replace(check,val,1)
            print('new ',new)
            stack=new[::-1]
            generation.append(stack)
            j=len(stack)-1
            print(steps)
            print(stack)
            if stack[-1]==check:
                i+=1
            if '~' in stack:
                stack=stack.replace('~','')
                j=len(stack)-1
                i=0
        print(i,j)
```

In [35]:
```
generatermd(stack,strrmd,i,j)
```

```
[]
elif 0 0
check  E
VAL reversed  AT
Stack reversed  E
new  AT
['E->TA']
TA
0 1
if 0 1
```

jupyter CD string generation Last Checkpoint: 12 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [36]:
```
stack[-1]
```

Out[36]: 'E'

In [37]:
```
strrmd.remove('$')
```

In [38]:
```
strrmd.append('$')
```

In [39]:
```
strrmd
```

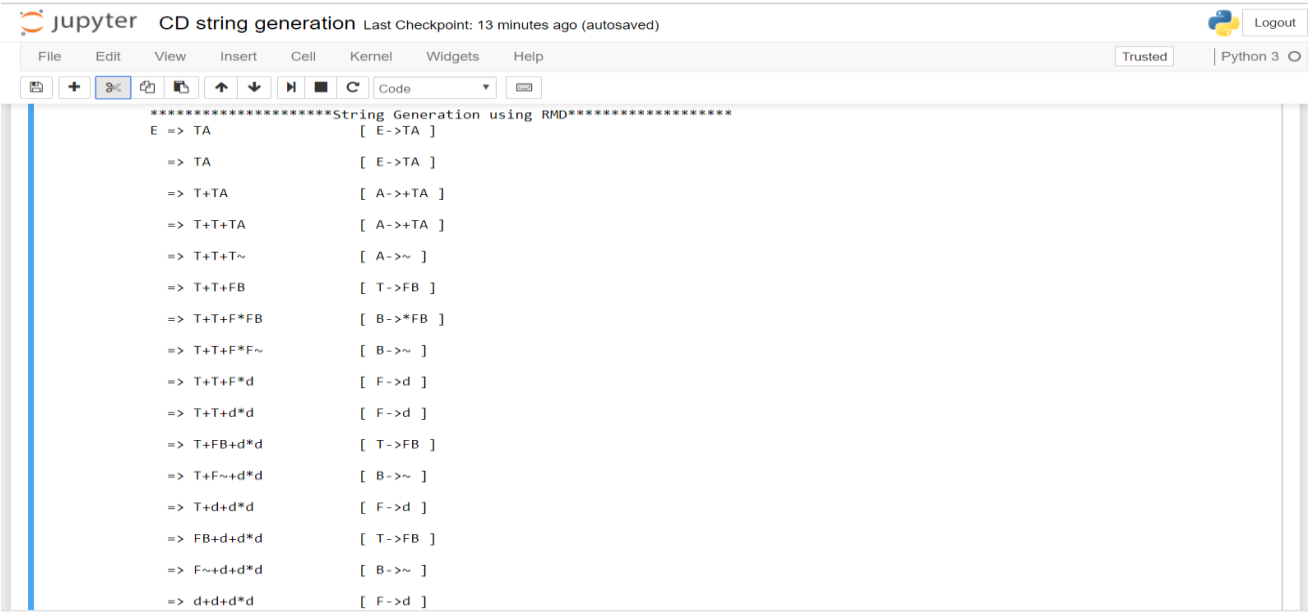
Out[39]: ['d', '*', 'd', '+', 'd', '+', 'd', '$']

In [40]:
```
str1
```

Out[40]: ['d', '+', 'd', '+', 'd', '*', 'd', '$']

In [41]:
```
print('********************String Generation using RMD******************')
if flag==2:
    print('String can not be generated ')
else:
    print(startsymbol,'=>',generation[0],end='')
    print(" "*16,"[",steps[0],"]")
    print('')
    for i in range(len(steps)):
        print(' '*(len(startsymbol)),'=>',generation[i]," "*(18-len(generation[i])),end='')
        print("[",steps[i],"]")
        print('')
```

```
********************String Generation using RMD******************
E => TA                 [ E->TA ]
```



Frequently Asked Questions(FAQs)

Q1. What is process of deriving string from the start symbol using the production rules of Grammar G?

Ans. String Derivation.

Q2. What is CFG?

Ans. A CFG is a set of recursive rewriting rules used to generate pattern of strings.

Q3.Any language that can be generated using CFG can also be generated using regular expressions? (True/False)

Ans. False.

Q4. Which derivation is used in bottom up parsing?

Ans. RMD

Q5. Which derivation is used in top down parsing?

Ans. LMD.

Practical-8 March20, 2018

Aim:

Write a program to implement operator precedence parser and show the step by step stack implementation of parsing.

Input:

The input to this program consists of the operator precedence grammar and the operator precedence relations from respective files. The user is also required to input the string on which the stack implementation is performed. Operator precedence parsing also requires the rule of operator precedence grammar.

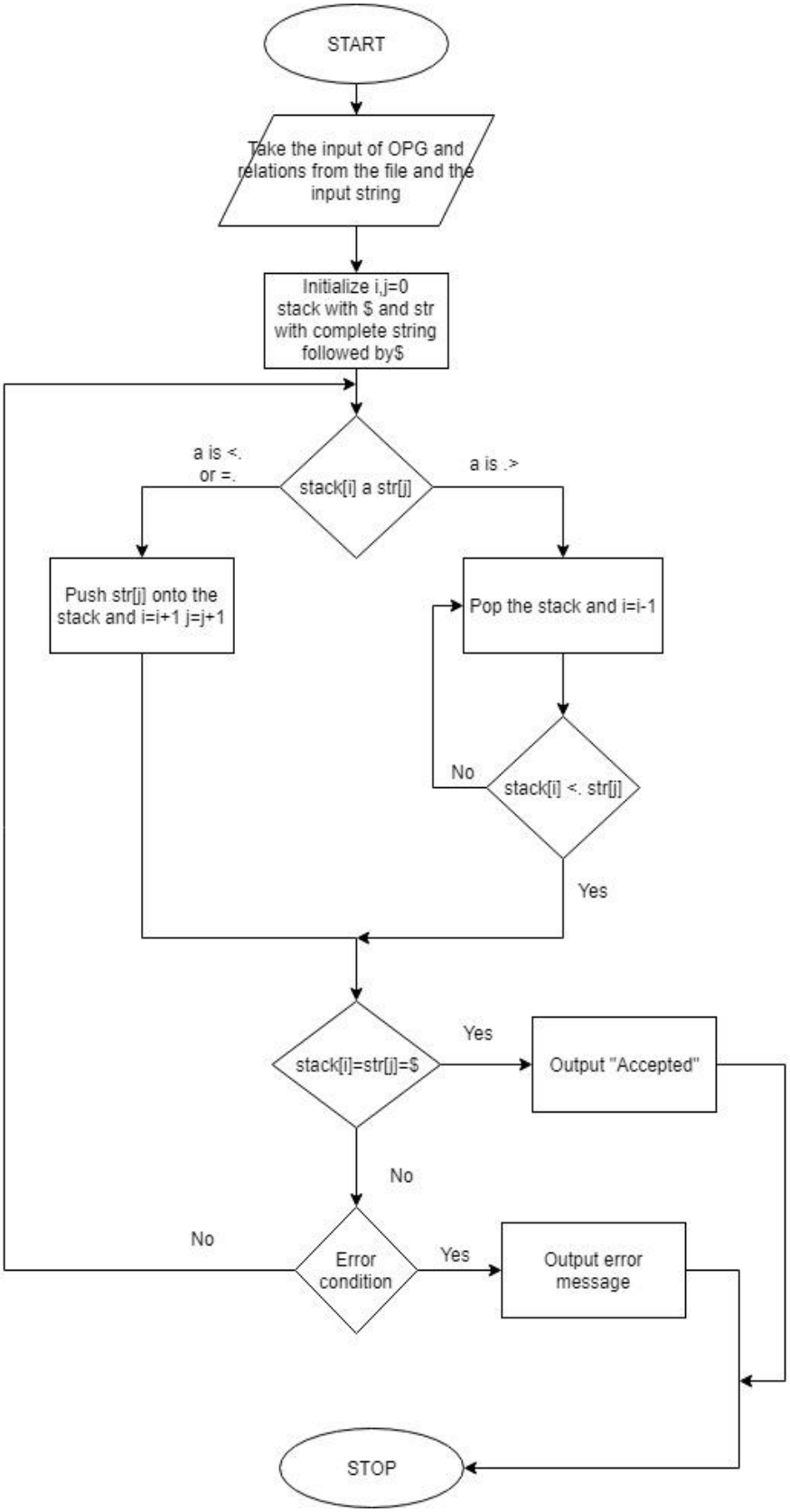
Expected Output:

When the program is executed the expected output of this program consists of the step by step stack implementation of the parsing of the string that the user has provided as the input to the program. For the purpose of parsing the code uses the rules and precedence relations presented in the file of whose the path is given as the input the program.

Algorithm:

1. START
2. Take the input of operator precedence grammar and operator precedence relations from the files and the input string.
3. Initialize i,j=0 put $ on the stack and string in the str followed by $
4. If stack[i]=str[j]=$ then accept and output “Accepted”
5. Else consider the operator precedence relations.
    a. If stack[i] <. str[j] or stack[i] =. str[j] then shift str[j] onto the stack and j=j+1 i=i+1
    b. Else if stack[i] .> str[j] then pop the stack and i=i-1 until stack terminal is related by <. to the terminal most recently popped
6. If stack[i]=$ but str[j]!=$ then through the error message.
7. STOP

Flowchart:


START
Take the input of OPG and relations from the file and the input string
Initialize i,j=0 stack with $ and str with complete string followed by$
stack[i] a str[j]
a is <. or =.
a is .>
Push str[j] onto the stack and i=i+1 j=j+1
Pop the stack and i=i-1
stack[i] <. str[j]
No
Yes
stack[i]=str[j]=$
Yes
Output "Accepted"
No
Error condition
No
Yes
Output error message
STOP

Source Code and Output:

jupyter grammar.txt ✔ Last Tuesday at 2:44 PM Logout

File Edit View Language Plain Text

```
E ->E+E
|E-E
|E*E
|E/E
|i
```

	A	B	C	D	E	F	G
1		+	-	*	/	i	$
2	+	>	>	<	<	<	>
3	-	>	>	<	<	<	>
4	*	>	>	>	>	<	>
5	/	>	>	>	>	<	>
6	i	>	>	>	>		>
7	$	<	<	<	<	<	
8							
9							
10							
11							
12							
13							

jupyter operatorprecedence Last Checkpoint: Last Tuesday at 2:40 PM (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Program to implement operator precedence parser and show step by step parsing

In [1]:
```
import sys
import shlex
import csv

#Write a program which will provide an implementation of operator precedance parser. Read an operator grammar and precedance tabl


def main():

    input_string = "i + i - i"
    input_ind = list(shlex.shlex(input_string))
    input_ind.append('$')

    master = {}
    master_list = []
    new_list = []
    non_terminals = []
    grammar = open('grammar.txt', 'r')

    for row2 in grammar:

        if '->' in row2:
            #new production
            if len(new_list) == 0:
```

jupyter operatorprecedence Last Checkpoint: Last Tuesday at 2:40 PM (autosaved)

File Edit View Insert Cell Kernel Widgets Help

```
                start_state = row2[0]
                non_terminals.append(row2[0])
                new_list = []
                new_list.append(row2.rstrip('\n'))
            else:
                master_list.append(new_list)
                del new_list
                new_list = []
                new_list.append(row2.rstrip('\n'))
                non_terminals.append(row2[0])

        elif '|' in row2:
            new_list.append(row2.rstrip('\n'))

    master_list.append(new_list)


    for x in range(len(master_list)):
        for y in range(len(master_list[x])):
            master_list[x][y] = [s.replace('|', '') for s in master_list[x][y]]
            master_list[x][y] = ''.join(master_list[x][y])
            master[master_list[x][y]] = non_terminals[x]

    for key, value in master.items():
        if '->' in key:
            length = len(key)
            for i in range(length):
                if key[i] == '-' and key[i + 1] == ">":
                    index =  i+2
                    break
            var_key = key
```

jupyter operatorprecedence Last Checkpoint: Last Tuesday at 2:40 PM (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
        new_key = key[index:]

    var = master[var_key]
    del master[var_key]
    master[new_key] = var

    order_table = []
    with open('order.csv', 'rU') as file2:
        order = csv.reader(file2)
        for row in order:
            order_table.append(row)

    operators = order_table[0]
    print(order_table)
    ###### Analysis
    stack = []

    stack.append('$')


    print("Stack", "\t\t\t\t\t\t", "Input", "\t\t\t\t", "Precedence relation", "\t\t", "Action")

    vlaag = 1
    while vlaag:
        if input_ind[0] == '$' and len(stack)==2:
            vlaag = 0

        length = len(input_ind)

        buffer_inp = input_ind[0]
        temp1 = operators.index(str(buffer_inp))
        print("stack",stack, stack[-1])
```

jupyter operatorprecedence Last Checkpoint: Last Tuesday at 2:40 PM (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
        print("stack",stack, stack[-1])
        if stack[-1] in non_terminals:
            buffer_stack = stack[-2]
        else:
            buffer_stack = stack[-1]
        temp2 = operators.index(str(buffer_stack))
        #print buffer_inp, buffer_stack

        precedence = order_table[temp2][temp1]

        if precedence == '<':
            action = 'shift'
        elif precedence == '>':
            action = 'reduce'

        print(stack, "\t\t", input_ind, "\t\t", precedence, "\t\t", action, "\n")

        if action == 'shift':
            stack.append(buffer_inp)
            input_ind.remove(buffer_inp)
        elif action == 'reduce':
            for key, value in master.items():
                var1 = ''.join(stack[-1:])
                var2 = ''.join(stack[-3:])
                if str(key) == str(buffer_stack):
                    stack[-1] = value
                    break
                elif key == var1 or stack[-3:]==list(var1):
                    stack[-3:] = value
                    break
                elif key == var2:
                    stack[-3:] = value
```

jupyter operatorprecedence Last Checkpoint: Last Tuesday at 2:40 PM (autosaved)

File Edit View Insert Cell Kernel Widgets Help

```
            stack.append(buffer_inp)
            input_ind.remove(buffer_inp)
        elif action == 'reduce':
            for key, value in master.items():
                var1 = ''.join(stack[-1:])
                var2 = ''.join(stack[-3:])
                if str(key) == str(buffer_stack):
                    stack[-1] = value
                    break
                elif key == var1 or stack[-3:]==list(var1):
                    stack[-3:] = value
                    break
                elif key == var2:
                    stack[-3:] = value
        del buffer_inp, temp1, buffer_stack, temp2, precedence

        if vlaag == 0:
            print("Accepted!!")

    return 2

if __name__ == "__main__":
    sys.exit(main())
```

```
[['', '+', '-', '*', '/', 'i', '$'], ['+', '>', '>', '<', '<', '<', '>'], ['-', '>', '>', '<', '<', '<', '>'], ['*', '>', '>',
'>', '>', '<', '>'], ['/', '>', '>', '>', '>', '<', '>'], ['i', '>', '>', '>', '>', '', '>'], ['$', '<', '<', '<', '<', '<',
'']]
Stack                                   Input                   Precedence relation         Action
stack ['$'] $
['$']           ['i', '+', 'i', '-', 'i', '$']          <           shift

stack ['$', 'i'] i
```

jupyter operatorprecedence Last Checkpoint: Last Tuesday at 2:40 PM (autosaved)

File Edit View Insert Cell Kernel Widgets Help

```
stack ['$', 'i'] i
['$', 'i']              ['+', 'i', '-', 'i', '$']           >           reduce

stack ['$', 'E'] E
['$', 'E']              ['+', 'i', '-', 'i', '$']           <           shift

stack ['$', 'E', '+'] +
['$', 'E', '+']                 ['i', '-', 'i', '$']        <           shift

stack ['$', 'E', '+', 'i'] i
['$', 'E', '+', 'i']            ['-', 'i', '$']             >           reduce

stack ['$', 'E', '+', 'E'] E
['$', 'E', '+', 'E']            ['-', 'i', '$']             >           reduce

stack ['$', 'E'] E
['$', 'E']              ['-', 'i', '$']             <           shift

stack ['$', 'E', '-'] -
['$', 'E', '-']                 ['i', '$']          <           shift

stack ['$', 'E', '-', 'i'] i
['$', 'E', '-', 'i']            ['$']           >           reduce

stack ['$', 'E', '-', 'E'] E
['$', 'E', '-', 'E']            ['$']           >           reduce

stack ['$', 'E'] E
['$', 'E']              ['$']                       reduce

Accepted!!
```

Frequently Asked Questions (FAQs)

Q1. OPG stands for?

Ans. Operator Precedence Grammar.

Q2. Which symbol is used when a yields precedence to b?

Ans. <.

Q3.What is the rule for having =. Precedence?

Ans. a=.b when for given production rule having RHS of the form éaBb§, where B is null or single non terminal and é,§ belongs to (VUT)*

Q4. Operator grammar can have production's right hand side as null. (True/False)

Ans. False

Q5. What are the two methods of calculating precedence table?

Ans. 1) Using the rules of association and precedence

2) Using OPG

Practical-9

March 27, 2018

Aim:

Write a program to implement predictive parser or LL parser and show the step by step stack implementation of parsing.

Input:

The input to this program will be the complete grammar along with the string to be generated, which includes the following:

- Non terminals and terminals present in the grammar.
- Start symbol and production rules for string derivation.

Expected Output:

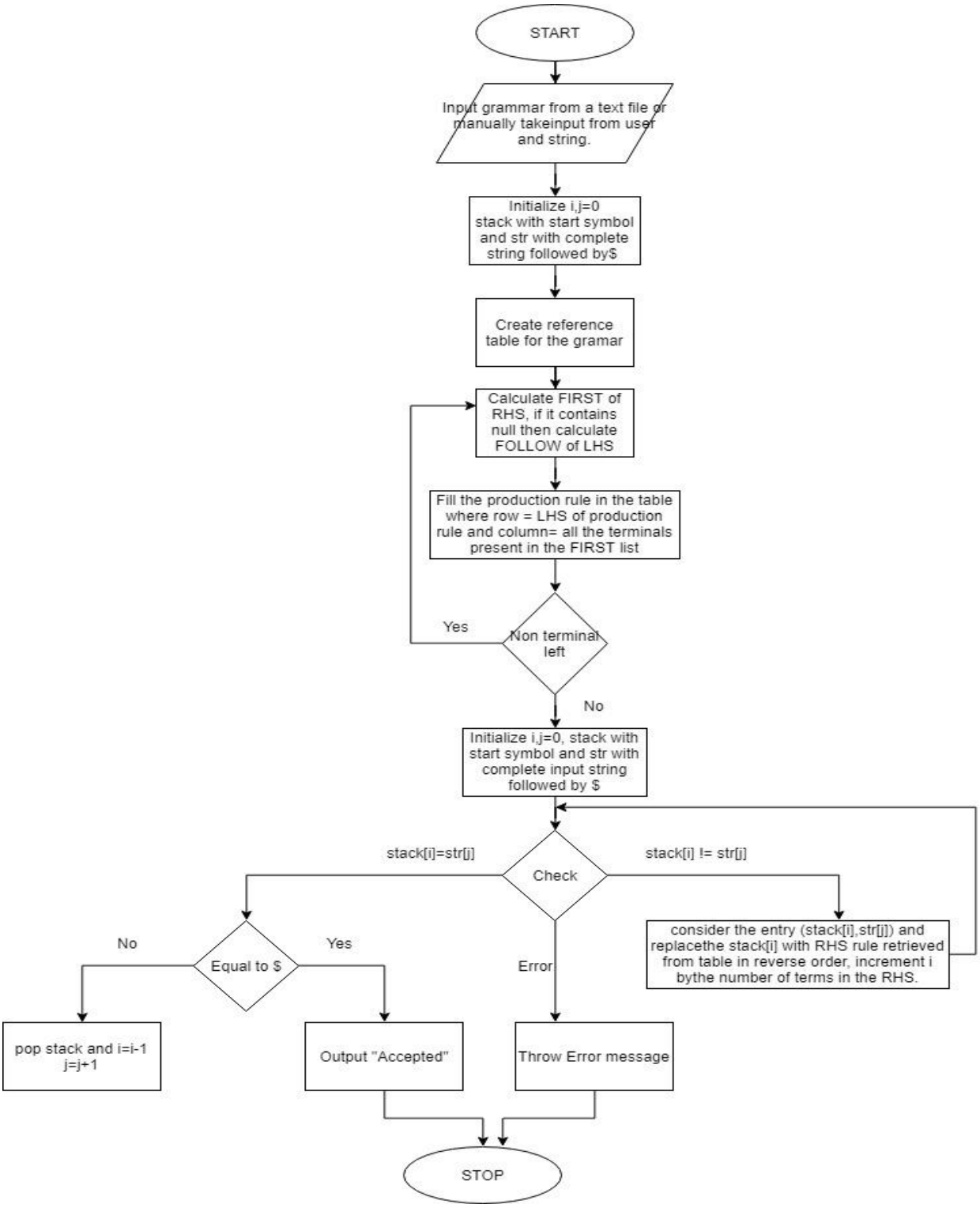
The expected output of this program consists of the step by step stack implementation of the parsing.

Algorithm:

1. START
2. Input grammar from a text file or manually take input from user and string.
3. Create reference table for the grammar.
    a. Repeat through step c for each production rule.
    b. Calculate FIRST of right hand side of production rule and if right hand is empty (null) calculate FOLLOW of left hand side of production and store it in index.
    c. Place the rule in table with row = left hand side of production and column = all the terminals in the FIRST list.
4. Initialize i,j=0 stack with start symbol and str with complete input string followed by $.
5. If stack[i]=str[j] then
    a. If there are equal to $ output “Accepted”
    b. Else pop stack and i=i-1 j=j+1
6. Else consider the entry (stack[i],str[j]) and replace the stack[i] with RHS rule retrieved from table in reverse order, increment i by the number of terms in the RHS.
7. If there is an error through an error message

8. STOP

Flowchart:



Source Code and Output:

jupyter predictive_parsing1 Last Checkpoint: 5 hours ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

Program to implement predictive parser and show the step by step stack implementation of parsing

In [2]:
```
nonterminals=[]
terminals=[]
productions=[]
```

In [2]:
```
productions=['E->TA','A->+TA','A->~','T->FB','B->*FB','B->~','F->(E)','F->d']
nonterminals=['E','A','T','B','F']
terminals=['+','*','d','(',')']
startsymbol='E'

lhs=[]
rhs=[]
for rule in productions:
    lhs.append(rule.split('->')[0])
    rhs.append(rule.split('->')[1])
print(lhs)
print(rhs)
print("Non terminals  ")
print(nonterminals)
print("Terminals")
print(terminals)
derivation=[]
print("Production rule")
print(productions)
```

jupyter predictive_parsing1 Last Checkpoint: 5 hours ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

```
['E', 'A', 'A', 'T', 'B', 'B', 'F', 'F']
['TA', '+TA', '~', 'FB', '*FB', '~', '(E)', 'd']
Non terminals
['E', 'A', 'T', 'B', 'F']
Terminals
['+', '*', 'd', '(', ')']
Production rule
['E->TA', 'A->+TA', 'A->~', 'T->FB', 'B->*FB', 'B->~', 'F->(E)', 'F->d']
```

In [3]:
```
def splitStr(str):
    stringList=[]
    temp=''
    for i in range(len(str)):
        temp=temp+str[i]
        if temp in terminals:
            stringList.append(temp)
            temp=''
    return stringList
```

In [89]:
```
str='d+d*d'
splitStr(str)
```

Out[89]:
```
['d', '+', 'd', '*', 'd']
```

In [90]:
```
def firstFun(nonterminal):
    firstn=[]
    if nonterminal[0] in terminals:
```

jupyter predictive_parsing1 Last Checkpoint: 5 hours ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

In [90]:
```
def firstFun(nonterminal):
    firstn=[]
    if nonterminal[0] in terminals:
        return nonterminal
    for rule in productions:
        lh=rule.split('->')[0]
        rh=rule.split('->')[1]
        if lh==nonterminal:
            temp=''
            for i in range(len(rh)):
                temp=temp+rh[i]
                if temp in terminals:
                    firstn.append(temp)
                    temp=''
                    break
                elif temp =='~':
                    firstn.append('~')
                elif temp in nonterminals:
                    n=firstFun(temp)
                    firstn.extend(n)
                    if '~' in n and isupper(rh):
                        continue

    return firstn
```

In [92]:
```
first={}
```

In [93]:
```
for each in nonterminals:
```

jupyter predictive_parsing1 Last Checkpoint: 5 hours ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

```
    first[each]=firstFun(each)
```

In [94]:
```
print(first)
```

```
{'E': ['(', 'd'], 'A': ['+', '~'], 'T': ['(', 'd'], 'B': ['*', '~'], 'F': ['(', 'd']}
```

In [95]:
```
def followFun(nonterminal):
    follown=[]
    for rule in productions:
        if nonterminal in rule.split('->')[1]:
            if nonterminal==startsymbol:
                follown.append('$')
            lh=rule.split('->')[0]
            rh=rule.split('->')[1]
            index=rh.find(nonterminal)
            if index<len(rh)-1:
                m=firstFun(rh[index+1])
                follown.extend(m)
                if '~' in m:
                    n=followFun(lh)
                    follown.extend(n)
                break
            if index==len(rh)-1:
                n=followFun(lh)
                follown.extend(n)
                break
    if '~' in follown:
        follown.remove('~')
    return follown
```

jupyter predictive_parsing1 Last Checkpoint: 5 hours ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

In [96]:
```
follow={}
```

In [97]:
```
for each in nonterminals:
    follow[each]=followFun(each)
```

In [98]:
```
print(follow)
```

```
{'E': ['$', ')'], 'A': ['$', ')'], 'T': ['+', '$', ')'], 'B': ['+', '$', ')'], 'F': ['*', '+', '$', ')']}
```

In [99]:
```
def firstFun2(l,str):
    temp=''
    firstn2=[]
    if str[0] in terminals:
        firstn2.append(str[0])
        return firstn2
    if str=='~':
        firstn2.extend(follow[l])
        return firstn2
    else:
        for i in range(len(str)):
            temp=temp+str[i]
            if temp in terminals:
                firstn2.append(temp)
                return firstn2
            if str[i] in nonterminals:
                m=first[str[i]]
                firstn2.extend(m)
                if '~' in m :
                    continue
                else:
```

jupyter predictive_parsing1 Last Checkpoint: 5 hours ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

```
                    break

    return firstn2
```

In [100]:
```
print(firstFun2('E','TA'))
```

```
['(', 'd']
```

In [101]:
```
table={}
```

In [102]:
```
for i in range(len(lhs)):
    positions=firstFun2(lhs[i],rhs[i])
    for j in range(len(positions)):
        k=(lhs[i],positions[j])
        table[k]=productions[i]
```

In [103]:
```
table
```

Out[103]:
```
{('A', '$'): 'A->~',
 ('A', ')'): 'A->~',
 ('A', '+'): 'A->+TA',
 ('B', '$'): 'B->~',
 ('B', ')'): 'B->~',
 ('B', '*'): 'B->*FB',
 ('B', '+'): 'B->~',
 ('E', '('): 'E->TA',
 ('E', 'd'): 'E->TA',
 ('F', '('): 'F->(E)',
 ('F', 'd'): 'F->d',
 ('T', '('): 'T->FB'
```

jupyter predictive_parsing1 Last Checkpoint: 5 hours ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

```
 ('T', '('): 'T->FB',
 ('T', 'd'): 'T->FB'}
```

In [104]:
```
str1=splitStr(str)
str1.append('$')
```

In [140]:
```
stack='$E'
steps=[]
generation=[]
stringtrack=[]
i=1
j=0
str1
```

Out[140]:
```
['d', '+', 'd', '*', 'd', '$']
```

In [124]:
```
if '~' in stack:
    stack=stack.replace('~','')
if stack[i]==str1[j] and stack[i]=='$':
    print('done')

if stack[i]=='~':
    stack=stack.replace('~','')


#print(str[j])
if stack[i]==str1[j] and stack[i]!='$':
    print('match',i,j)
    print('stack , str' , stack[i] ,str1[j])
    #steps.append(table[(temp,str[j])])
    #str.remove(str[j])
```

jupyter predictive_parsing1 Last Checkpoint: 5 hours ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

```
    stack=stack[:-1]
    j=j+1
    i=len(stack)-1
    #i=i+1
    print(i,j)
    print(steps)
    print(stack)
else:
    print(i,j,stack[i],str1[j])

    val=table[(stack[i],str1[j])].split('->')[1]
    #new=stack.replace(stack[i],table[(stack[i],str[j])]).split('->')[1]
    val=val[::-1]
    new=stack.replace(stack[i],val,1)
    steps.append(table[(stack[i],str1[j])])
    stack=new

    i= len(stack)-1
    print(steps)

    print(new)
    print(stack)

    if '~' in stack:
        stack=stack.replace('~','')
        i= len(stack)-1
    generation.append(stack)
    print(stack)
    print(i,j)
```

```
done
0 5 $ $
```

jupyter predictive_parsing1 Last Checkpoint: 5 hours ago (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

Code

In [141]:
```
def generate(str1,stack,i,j):
    try:
       while True:
        if '~' in stack:
            stack=stack.replace('~','')
        if stack[i]==str1[j] and stack[i]=='$':
            return 'Accepted'

        if stack[i]=='~':
            stack=stack.replace('~','')


        #print(str[j])
        if stack[i]==str1[j] and stack[i]!='$':
            print('match',i,j)
            print('stack , str' , stack[i] ,str1[j])
            #steps.append(table[(temp,str[j])])
            #str.remove(str[j])
            stack=stack[:-1]
            j=j+1
            i=len(stack)-1
            #i=i+1
            stringtrack.append(str[j:])
            print(i,j)
            print(steps)
            print(stack)
            steps.append(table[(stack[i],str1[j])])
            generation.append(stack)
        else:
            print(i,j,stack[i],str1[j])

            val=table[(stack[i],str1[j])].split('->')[1]
```

jupyter predictive_parsing1 Last Checkpoint: 5 hours ago (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

Code

```
            val=table[(stack[i],str1[j])].split('->')[1]
            #new=stack.replace(stack[i],table[(stack[i],str[j])]).split('->')[1]
            val=val[::-1]
            new=stack.replace(stack[i],val,1)
            steps.append(table[(stack[i],str1[j])])
            stack=new

            i= len(stack)-1
            print(steps)
            stringtrack.append(str[j:])
            print(new)
            print(stack)

            if '~' in stack:
                stack=stack.replace('~','')
                i= len(stack)-1
            generation.append(stack)
            print(stack)
    except:
        return 1
```

In [143]:
```
flag
```

Out[143]: 'Accepted'

In [148]:
```
generation
for i in range(len(stringtrack)):
    stringtrack[i]=stringtrack[i]+'$'
```

jupyter predictive_parsing1 Last Checkpoint: 5 hours ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

```
In [145]: steps

Out[145]: ['E->TA',
           'T->FB',
           'F->d',
           'B->~',
           'B->~',
           'A->+TA',
           'T->FB',
           'T->FB',
           'F->d',
           'B->*FB',
           'B->*FB',
           'F->d',
           'F->d',
           'B->~',
           'B->~',
           'A->~']

In [149]: print('********************Predictive parsing(LL parser) ******************')
          if flag==1:
              print(' Can not parse ')
          else:
              print('Stack',' '*(18- len('stack')),'Input',end='')
              print(" "*16,'Operation')
              print('')
              for i in range(len(stringtrack)):
                  print(generation[i]," "*(18-len(generation[i])),stringtrack[i],end='')
                  print(' '*(18-len(stringtrack[i])),"[",steps[i],"]")
                  print('')
```

jupyter predictive_parsing1 Last Checkpoint: 5 hours ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

```
********************Predictive parsing(LL parser) ******************
Stack              Input                Operation

$AT                d+d*d$             [ E->TA ]

$ABF               d+d*d$             [ T->FB ]

$ABd               d+d*d$             [ F->d ]

$AB                +d*d$              [ B->~ ]

$A                 +d*d$              [ B->~ ]

$AT+               +d*d$              [ A->+TA ]

$AT                d*d$               [ T->FB ]

$ABF               d*d$               [ T->FB ]

$ABd               d*d$               [ F->d ]

$AB                *d$                [ B->*FB ]

$ABF*              *d$                [ B->*FB ]

$ABF               d$                 [ F->d ]

$ABd               d$                 [ F->d ]

$AB                $                  [ B->~ ]

$A                 $                  [ B->~ ]

$                  $                  [ A->~ ]
```

Frequently Asked Questions (FAQs)

Q1.Another name for shift reduce parsing is

Ans. LL parsing.

Q2. What is LL parser?

Ans. In computer science, an LL parser is a top-down parser for a subset of context-free languages. It parses the input from Left to right, performing Leftmost derivation of the sentence. An LL parser is called an LL(k) parser if it uses k tokens of lookahead when parsing a sentence..

Q3. What are the different methods of top down parsing?

Ans. Backtracking, Predictive Parsing and Recursive Descent Parsing.

Q4.Differentiate between LL(0) and LR(0) parsers.

Ans. LL(0) parsers mean they process the token stream from Left to right, using Leftmost derivation with no look ahead. LR(0) parsers mean they process the token stream from Left to right, using Rightmost derivation with zero look ahead. It means that they build the parse tree from bottom to top, whereas LL(0) parsers build the parse tree from top to bottom.

Q5. What are the problems with top down parsing?

Ans. Left recursion, backtracking, choosing the right alternative, difficulty in error localization.

Practical-10

April 3, 2018

Aim:

Write a program to implement SLR parser.

Input:

The input to this program is the file containing the production rules of the grammar.

Expected Output:

The expected output of this program consists of the following:

- The rules of the grammar read from the file.
- Augmented grammar
- Canonical collection of LR(0) items.
- GOTO operations and reduction steps
- Step by step stack implementation of SLR parsing.
- Final message showing acceptance or rejection of string.

Algorithm:

(1) START

(2) Initialize all the lists required.

(3) Read the file and write the augmented grammar.

(4) Calculate I0 by taking the CLOSURE of first production rule of augmented grammar.

(5) Find the GOTO of I0 at each symbol that supports the form '.A'

(a) For each new set created add it to the list of canonical LR(0) item sets.

(6) For each new state created calculate GOTO and fill the GOTO table side by side.

(7) Every item set which have the RHS of production rule of the form 'S.' where S is the start symbol for grammar mark the entry in parsing table under $ as accept

(8) For each item set that has a production rule where dot symbol occurs at the end of RHS then using this item set as the row mark reduce operation under respective columns.

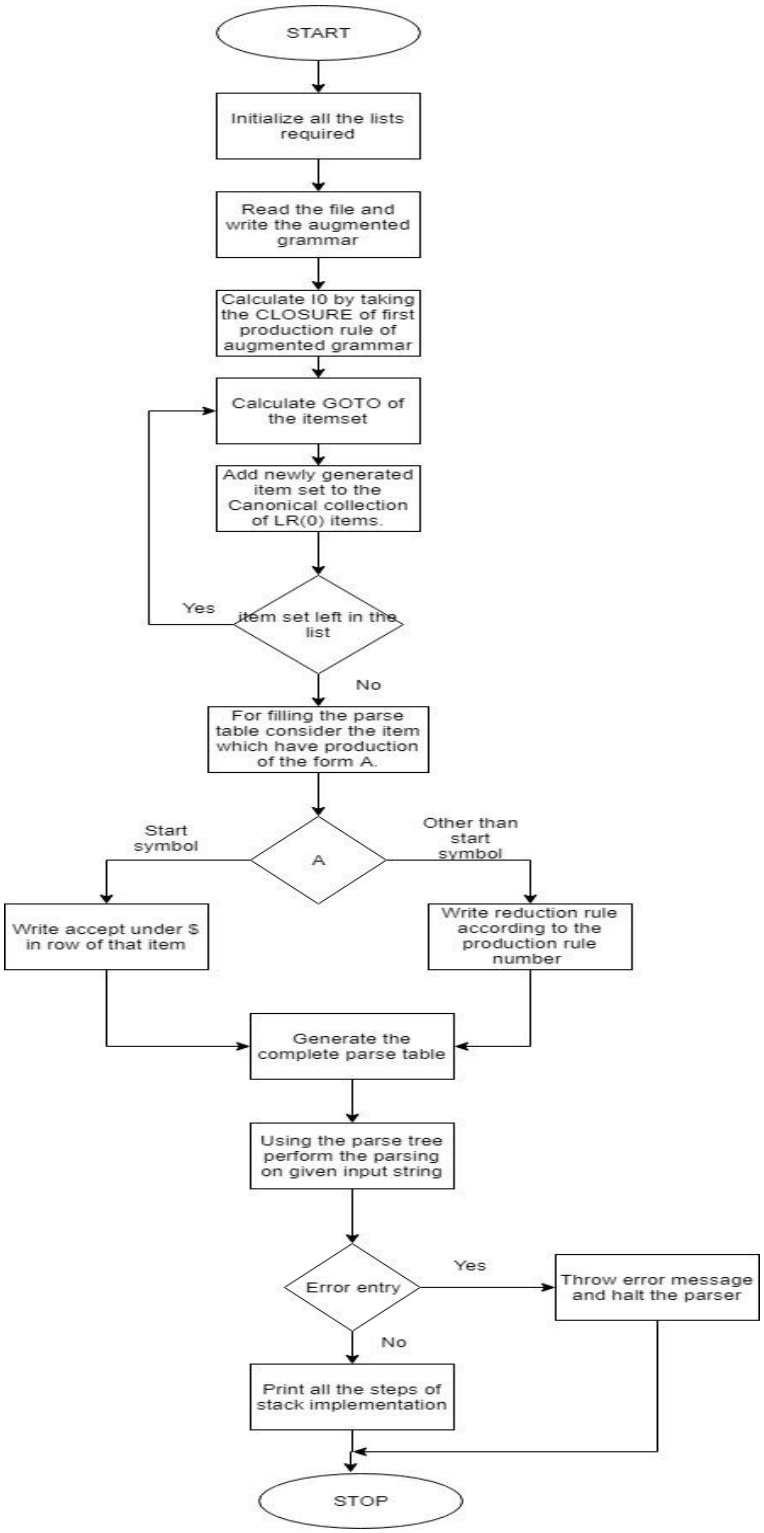
(9) Using the parsing table created in the previous step perform the parsing of string.

(a) According to the entries in the table perform the shift or reduce operations.

(b)If there is any error through an error message

(c)Once the parsing is done print the stack implementation and STOP

Flowchart:



Source Code and Output:

jupyter CDlab10 Last Checkpoint: 10 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted Py

Code

Program to implement SLR parser

In [1]:
```
import copy
```

In [2]:
```
grammar = []
new_grammar = []
terminals = []
non_terminals = []
I_n = {}
shift_list = []
reduction_list = []
action_list = []
rule_dict = {}
follow_dict = {}
SR = []
RR = []
```

In [3]:
```
def Conflict():
    global SR, RR, shift_list, reduction_list
    conflict = False
    # SR conflict if shift and Reduce occurs for same condition
    for S in shift_list:
        for R in reduction_list:
            if S[:2] == R[:2]:
```

jupyter CDlab10 Last Checkpoint: 10 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted Pyt

Code

```
                SR.append([S, R])
                conflict = True

    # RR conflict if 2 Reduce occurs for same condition
    for R1 in reduction_list:
        for R2 in reduction_list:
            if R1 == R2:
                continue

            if R1[:2] == R2[:2]:
                RR.append(R1)
                conflict = True

    return conflict
```

In [4]:
```
def read_grammar():
    global grammar, terminals, non_terminals, rule_dict

    file_name = input("Enter Grammar File Name:: ")
    try:
        grammar_file = open(file_name, "r")
    except:
        print("Cannot Find File Named", file_name)
        exit(0)
    # add garmmar
    for each_grammar in grammar_file:
        grammar.append(each_grammar.strip())

        # find terminals
```

jupyter CDlab10 Last Checkpoint: 10 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted

```
        if each_grammar[0] not in non_terminals:
            non_terminals.append(each_grammar[0])

    # find non terminals
    for each_grammar in grammar:
        for token in each_grammar.strip().replace(" ", "").replace("->", ""):
            if token not in non_terminals and token not in terminals:
                terminals.append(token)

    # generate dictionary of rules
    for l in range(1, len(grammar) + 1):
        rule_dict[l] = grammar[l - 1]
```

In [5]:
```
def augmented_grammar():
    global grammar, new_grammar
    read_grammar()
    # if non augmented grammar is given, augment it
    if "'" not in grammar[0]:
        grammar.insert(0, grammar[0][0] + "'" + "->" + grammar[0][0])

    # just add . infornt of each rule
    new_grammar = []
    for each_grammar in grammar:
        idx = each_grammar.index(">")
        # print each_grammar.split()
        each_grammar = each_grammar[:idx + 1] + "." + each_grammar[idx + 1:]
        new_grammar.append(each_grammar)
```

In [6]: def compute_I0():

jupyter CDlab10 Last Checkpoint: 10 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted

In [6]:
```
def compute_I0():
    global new_grammar, non_terminals, I_n
    augmented_grammar()

    grammar2add = []

    # add first rule to I(0)
    grammar2add.append(new_grammar[0])
    i = 0

    # check for terminals in new_grammar[0]
    for each in grammar2add:
        current_pos = each.index(".")
        current_variable = each[current_pos + 1]

        if current_variable in non_terminals:
            for each_grammar in new_grammar:
                if each_grammar[0] == current_variable and each_grammar not in grammar2add:
                    grammar2add.append(each_grammar)

        I_n[i] = grammar2add
```

In [7]:
```
def GOTO():
    global grammar, non_terminals, terminals, I_n, shift_list
    compute_I0()
    variables = non_terminals + terminals
    i = 0
    current_state = 0
    done = False
```

jupyter CDlab10 Last Checkpoint: 11 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted

Code

```
    while (not done):
        for each_variable in variables:
            grammar2add = []
            try:
                for each_rule in I_n[current_state]:
                    if each_rule[-1] == ".":
                        continue
                    dot_idx = each_rule.index(".")
                    if each_rule[dot_idx + 1] == each_variable:
                        rule = copy.deepcopy(each_rule)
                        rule = rule.replace(".", "")
                        rule = rule[:dot_idx + 1] + "." + rule[dot_idx + 1:]
                        grammar2add.append(rule)
                        for rule in grammar2add:
                            dot_idx = rule.index(".")
                            if rule[-1] == ".":
                                pass
                            else:
                                current_variable = rule[dot_idx + 1]
                                if current_variable in non_terminals:
                                    for each_grammar in new_grammar:
                                        if each_grammar[0] == current_variable and each_grammar[
                                            1] != "'" and each_grammar not in grammar2add:
                                            grammar2add.append(each_grammar)
            except:
                done = True
                break
            if grammar2add:
                if grammar2add not in I_n.values():
```

jupyter CDlab10 Last Checkpoint: 11 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted

Code

```
                    i += 1
                    I_n[i] = grammar2add
                for k, v in I_n.items():
                    if grammar2add == v:
                        idx = k
                shift_list.append([current_state, each_variable, idx])
        current_state += 1
```

In [8]:

```
def follow(var):
    global rule_dict, follow_dict, terminals
    value = []
    if var == rule_dict[1][0]:
        value.append("$")
    for rule in rule_dict.values():
        lhs, rhs = rule.split("->")
        if var == rule[-1]:
            for each in follow(rule[0]):
                if each not in value:
                    value.append(each)
        if var in rhs:
            idx = rhs.index(var)
            try:
                if rhs[idx + 1] in non_terminals and rhs[idx + 1] != var:
                    for each in follow(rhs[idx + 1]):
                        value.append(each)
                else:
                    value.append(rhs[idx + 1])
            except:
                pass
```

jupyter CDlab10 Last Checkpoint: 11 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted

```
    return value
```

In [9]:
```
def reduction():
    global I_n, rule_dict, reduction_list
    reduction_list.append([1, "$", "Accept"])
    for item in I_n.items():
        try:
            for each_production in item[1]:
                lhs, rhs = each_production.split(".")
                for rule in rule_dict.items():
                    if lhs == rule[1]:
                        f = follow(lhs[0])
                        for each_var in f:
                            reduction_list.append([item[0], each_var, "R" + str(rule[0])])
        except:
            pass
```

In [10]:
```
def test(string):
    global action_list, shift_list, reduction_list
    done = False
    stack = []
    stack.append(0)
    print("\n\nSTACK\t\tSTRING\t\tACTION")
    while not done:
        Reduce = False
        Shift = False
        # Check for reduction
        for r in reduction_list:
            # Reduce
```

jupyter CDlab10 Last Checkpoint: 11 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted

```
            # Reduce
            if r[0] == int(stack[-1]) and r[1] == string[0]:
                Reduce = True
                print
                ''.join(str(p) for p in stack), "\t\t", string, "\t\t", "Reduce", r[2]
                if r[2] == 'Accept':
                    return 1
                var = rule_dict[int(r[2][1])]
                lhs, rhs = var.split("->")
                for x in range(len(rhs)):
                    stack.pop()
                    stack.pop()
                var = lhs
                stack.append(var)
                for a in action_list:
                    if a[0] == int(stack[-2]) and a[1] == stack[-1]:
                        stack.append(str(a[2]))
                        break
        # Check for shift
        for g in shift_list:
            # Shift
            if g[0] == int(stack[-1]) and g[1] == string[0]:
                Shift = True
                print(''.join(str(p) for p in stack), "\t\t", string, "\t\t", "Shift", "S" + str(g[2]))
                stack.append(string[0])
                stack.append(str(g[2]))
                string = string[1:]
        if not Reduce and not Shift:
            print(''.join(str(p) for p in stack), "\t\t", string)
            return 0
```

jupyter CDlab10 Last Checkpoint: 12 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted

Code

In [11]:
```
def main():
    global I_n, shift_list, reduction_list, action_list, SR, RR
    GOTO()
    reduction()
    print("\n***************RULES*****************")
    for item in rule_dict.items():
        print(item)
    print("\n*********************AUGMENTED RULES********************")
    for item in new_grammar:
        print(item.replace(".", ""))
    print("\n**********************STATES**********************")
    for item in I_n.items():
        print( item)
    print("\n**********************GOTO OPERATIONS********************")
    for item in shift_list:
        print(item)
    print("\n**********************REDUCTION**********************")
    for item in reduction_list:
        print(item)
    if Conflict():
        if SR != []:
            print("SR conflict")
            for item in SR:
                print(item)
            print
        if RR != []:
            print("RR conflict")
            for item in RR:
                print(item)
```

jupyter CDlab10 Last Checkpoint: 12 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted

Code

```
            print
        exit(0)
    else:
        print("\nNO CONFLICT\n")
    print("Terminals:", terminals)
    print("NonTerminals:", non_terminals)
    print
    action_list.extend(shift_list)
    action_list.extend(reduction_list)
    string = input("\n\nEnter String:: ")
    try:
        if string[-1] != "$":
            string = string + "$"
    except:
        print("InputError")
        exit(0)
    print("\nTest String:", string)
    result = test(string)
    if result == 1:
        print( "***ACCEPTED***")
    elif result == 0:
        print("***NOT ACCEPTED***")
    return 0
if __name__ == '__main__':
    main()
```

```
Enter Grammar File Name:: slr_input.txt

***************RULES*****************
(1, 'E->E+T')
```

jupyter CDlab10 Last Checkpoint: 12 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted

```
Enter Grammar File Name:: slr_input.txt

***************RULES*****************
(1, 'E->E+T')
(2, 'E->T')
(3, 'T->T*F')
(4, 'T->F')
(5, 'F->(E)')
(6, 'F->i')

**********************AUGMENTED RULES********************
E'->E
E->E+T
E->T
T->T*F
T->F
F->(E)
F->i

**********************STATES**********************
(0, ["E'->.E", 'E->.E+T', 'E->.T', 'T->.T*F', 'T->.F', 'F->.(E)', 'F->.i'])
(1, ["E'->E.", 'E->E.+T'])
(2, ['E->T.', 'T->T.*F'])
(3, ['T->F.'])
(4, ['F->(.E)', 'E->.E+T', 'E->.T', 'T->.T*F', 'T->.F', 'F->.(E)', 'F->.i'])
(5, ['F->i.'])
(6, ['E->E+.T', 'T->.T*F', 'T->.F', 'F->.(E)', 'F->.i'])
(7, ['T->T*.F', 'F->.(E)', 'F->.i'])
(8, ['F->(E.)', 'E->E.+T'])
```

jupyter CDlab10 Last Checkpoint: 13 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted

```
(9, ['E->E+T.', 'T->T.*F'])
(10, ['T->T*F.'])
(11, ['F->(E).'])

**********************GOTO OPERATIONS********************
[0, 'E', 1]
[0, 'T', 2]
[0, 'F', 3]
[0, '(', 4]
[0, 'i', 5]
[1, '+', 6]
[2, '*', 7]
[4, 'E', 8]
[4, 'T', 2]
[4, 'F', 3]
[4, '(', 4]
[4, 'i', 5]
[6, 'T', 9]
[6, 'F', 3]
[6, '(', 4]
[6, 'i', 5]
[7, 'F', 10]
[7, '(', 4]
[7, 'i', 5]
[8, '+', 6]
[8, ')', 11]
[9, '*', 7]

**********************REDUCTION**********************
[1, '$', 'Accept']
```

jupyter CDlab10 Last Checkpoint: 13 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted

```
**********************REDUCTION**********************
[1, '$', 'Accept']
[2, '$', 'R2']
[2, '+', 'R2']
[2, ')', 'R2']
[3, '$', 'R4']
[3, '+', 'R4']
[3, ')', 'R4']
[3, '*', 'R4']
[5, '$', 'R6']
[5, '+', 'R6']
[5, ')', 'R6']
[5, '*', 'R6']
[9, '$', 'R1']
[9, '+', 'R1']
[9, ')', 'R1']
[10, '$', 'R3']
[10, '+', 'R3']
[10, ')', 'R3']
[10, '*', 'R3']
[11, '$', 'R5']
[11, '+', 'R5']
[11, ')', 'R5']
[11, '*', 'R5']

NO CONFLICT

Terminals: ['+', '*', '(', ')', 'i']
NonTerminals: ['E', 'T', 'F']
```

jupyter CDlab10 Last Checkpoint: 14 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted

```
[9, '+', 'R1']
[9, ')', 'R1']
[10, '$', 'R3']
[10, '+', 'R3']
[10, ')', 'R3']
[10, '*', 'R3']
[11, '$', 'R5']
[11, '+', 'R5']
[11, ')', 'R5']
[11, '*', 'R5']

NO CONFLICT

Terminals: ['+', '*', '(', ')', 'i']
NonTerminals: ['E', 'T', 'F']


Enter String:: i+i*i

Test String: i+i*i$


STACK           STRING          ACTION
0                i+i*i$                  Shift S5
0E1              +i*i$           Shift S6
0E1+6            i*i$            Shift S5
0E1+6T9                  *i$             Shift S7
0E1+6T9*7                i$              Shift S5
***ACCEPTED***
```

Frequently Asked Questions (FAQs)

Q1. What is SLR parser?

Ans. SLR stands for simple LR parser.

Q2. Which string derivation is used in SLR parser?

Ans. RMD.

Q3.SLR is bottom up parser. (True/False)

Ans. True.

Q4. Fill in the blank.

In SLR parsing canonical collection of _____ items are formed.

Ans. LR(0)

Q5. What is an augmented grammar?

Ans. Augmented grammar is the grammar having the augmented production rule, which is of the form S'-> S. Where S is the start symbol of original grammar G.

Practical-11

April 10, 2018

Aim:

Write a program to implement CLR parser.

Input:

The input to this program is the file containing the production rules of the grammar.

Expected Output:

The expected output of this program consists of the following:

- The rules of the grammar read from the file.
- Augmented grammar
- Canonical collection of LR(0) items.
- GOTO operations and reduction steps
- Step by step stack implementation of SLR parsing.
- Final message showing acceptance or rejection of string.

Algorithm:

1. START
2. Initialize all the lists
3. Take the input of production rules from the user one by one
4. Compute the augmented grammar
5. Calculate FIRST and FOLLOW of all the non-terminals present in the grammar.
6. Compute the item set along with the look ahead symbols
7. Calculate GOTO operation by taking CLOSURE of the production rule with LHS having any non-terminal after the dot symbol in the item set
8. If the result is any new item set then mark it as new set and calculate GOTO for it
9. Fill the CLR parsing table with shift, reduce and accept operations using these item sets generated
10. Perform parsing using this table
    a. If there is shift operation shift the symbol onto the stack

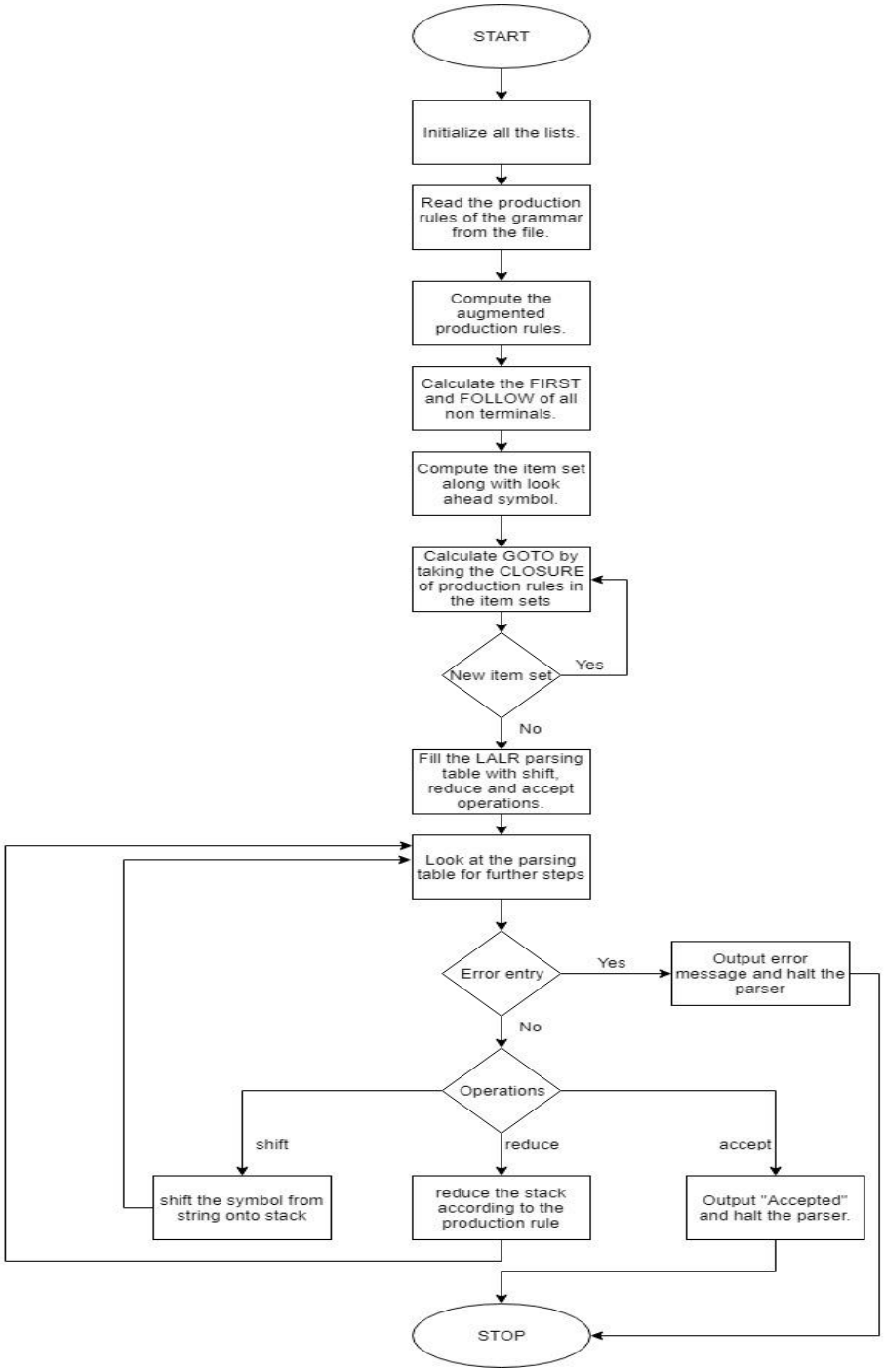
b. If there is reduce operation reduce the expression in the stack

c. If the accept entry is encountered then output "Accepted" and halt the parser

d. If an error entry is encountered then throw an error message and halt the parser

11. STOP

Flowchart:



Source Code and Output:

jupyter CDlab11 Last Checkpoint: 17 hours ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Program to implement CLR parser

In [1]:
```
import pandas as pd
import firstfollow
from firstfollow import production_list, nt_list as ntl, t_list as tl
```

In [2]:
```
nt_list, t_list = [], []
```

In [3]:
```
class State:
    _id = 0
    def __init__(self, closure):
        self.closure = closure
        self.no = State._id
        State._id += 1
class Item(str):
    def __new__(cls, item, lookahead=list()):
        self = str.__new__(cls, item)
        self.lookahead = lookahead
        return self
    def __str__(self):
        return super(Item, self).__str__() + ", " + '|'.join(self.lookahead)
```

In [4]:
```
def closure(items):
    def exists(newitem, items):
        for i in items:
            if i == newitem and sorted(set(i.lookahead)) == sorted(set(newitem.lookahead)):
                return True
        return False
    global production_list
    while True:
```

jupyter CDlab11 Last Checkpoint: 17 hours ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

```
    while True:
        flag = 0
        for i in items:
            if i.index('.') == len(i) - 1: continue
            Y = i.split('->')[1].split('.')[1][0]
            if i.index('.') + 1 < len(i) - 1:
                lastr = list(firstfollow.compute_first(i[i.index('.') + 2]) - set(chr(1013)))
            else:
                lastr = i.lookahead
            for prod in production_list:
                head, body = prod.split('->')
                if head != Y: continue
                newitem = Item(Y + '->.' + body, lastr)
                if not exists(newitem, items):
                    items.append(newitem)
                    flag = 1
        if flag == 0: break
    return items
```

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

In [6]:
```
def goto(items, symbol):
    global production_list
    initial = []
    for i in items:
        if i.index('.') == len(i) - 1: continue
        head, body = i.split('->')
        seen, unseen = body.split('.')
        if unseen[0] == symbol and len(unseen) >= 1:
            initial.append(Item(head + '->' + seen + unseen[0] + '.' + unseen[1:], i.lookahead))
    return closure(initial)
```

In [7]:
```
def calc_states():
```

jupyter CDlab11 Last Checkpoint: 17 hours ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [7]:
```
def calc_states():
    def contains(states, t):
        for s in states:
            if len(s) != len(t): continue
            if sorted(s) == sorted(t):
                for i in range(len(s)):
                    if s[i].lookahead != t[i].lookahead: break
                else:
                    return True
        return False
    global production_list, nt_list, t_list
    head, body = production_list[0].split('->')
    states = [closure([Item(head + '->.' + body, ['$'])])]
    while True:
        flag = 0
        for s in states:
            for e in nt_list + t_list:
                t = goto(s, e)
                if t == [] or contains(states, t): continue
                states.append(t)
                flag = 1
        if not flag: break
    return states
```

In [8]:
```
def make_table(states):
    global nt_list, t_list
    def getstateno(t):
        for s in states:
            if len(s.closure) != len(t): continue
            if sorted(s.closure) == sorted(t):
                for i in range(len(s.closure)):
                    if s.closure[i].lookahead != t[i].lookahead: break
                else:
```

jupyter CDlab11 Last Checkpoint: 17 hours ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
                else:
                    return s.no
        return -1
```

In [9]:
```
def getprodno(closure):
    closure = ''.join(closure).replace('.', '')
    return production_list.index(closure)
    SLR_Table = OrderedDict()
    for i in range(len(states)):
        states[i] = State(states[i])
    for s in states:
        SLR_Table[s.no] = OrderedDict()
        for item in s.closure:
            head, body = item.split('->')
            if body == '.':
                for term in item.lookahead:
                    if term not in SLR_Table[s.no].keys():
                        SLR_Table[s.no][term] = {'r' + str(getprodno(item))}
                    else:
                        SLR_Table[s.no][term] |= {'r' + str(getprodno(item))}
                continue
            nextsym = body.split('.')[1]
            if nextsym == '':
                if getprodno(item) == 0:
                    SLR_Table[s.no]['$'] = 'accept'
                else:
                    for term in item.lookahead:
                        if term not in SLR_Table[s.no].keys():
                            SLR_Table[s.no][term] = {'r' + str(getprodno(item))}
                        else:
                            SLR_Table[s.no][term] |= {'r' + str(getprodno(item))}
                continue
            nextsym = nextsym[0]
            t = goto(s.closure, nextsym)
```

jupyter CDlab11 Last Checkpoint: 17 hours ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Code

```
            if t != []:
                if nextsym in t_list:
                    if nextsym not in SLR_Table[s.no].keys():
                        SLR_Table[s.no][nextsym] = {'s' + str(getstateno(t))}
                    else:
                        SLR_Table[s.no][nextsym] |= {'s' + str(getstateno(t))}
                else:
                    SLR_Table[s.no][nextsym] = str(getstateno(t))
    return SLR_Table
```

In [10]:
```
def augment_grammar():
    for i in range(ord('Z'), ord('A') - 1, -1):
        if chr(i) not in nt_list:
            start_prod = production_list[0]
            production_list.insert(0, chr(i) + '->' + start_prod.split('->')[0])
            return
```

In [11]:
```
def main():
    global production_list, ntl, nt_list, tl, t_list
    firstfollow.main()
    print("\tFIRST AND FOLLOW OF NON-TERMINALS")
    for nt in ntl:
        firstfollow.compute_first(nt)
        firstfollow.compute_follow(nt)
        print(nt)
        print("\tFirst:\t", firstfollow.get_first(nt))
        print("\tFollow:\t", firstfollow.get_follow(nt), "\n")
    augment_grammar()
    nt_list = list(ntl.keys())
    t_list = list(tl.keys()) + ['$']
    print(nt_list)
    print(t_list)
    j = calc_states()
```

jupyter CDlab11 Last Checkpoint: 17 hours ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Code

```
    ctr = 0
    for s in j:
        print("Item{}:".format(ctr))
        for i in s:
            print("\t", i)
        ctr += 1
    table = make_table(j)
    print("\n\tCLR(1) TABLE\n")
    sr, rr = 0, 0
    new = {}
    for k, v in table.items():
        new[k] = dict(v)
    df = pd.DataFrame(new).T
    df.fillna('', inplace=True)
    print(df)
    for i, j in table.items():
        s, r = 0, 0
        for p in j.values():
            if p != 'accept' and len(p) > 1:
                p = list(p)
                if ('r' in p[0]):
                    r += 1
                else:
                    s += 1
                if ('r' in p[1]):
                    r += 1
                else:
                    s += 1
        if r > 0 and s > 0:
            sr += 1
        elif r > 0:
            rr += 1
    print("\n", sr, "s/r conflicts |", rr, "r/r conflicts")
    return
if __name__ == "__main__":
```

jupyter CDlab11 Last Checkpoint: 17 hours ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Code

```
    return
if __name__ == "__main__":
    main()
```

```
Enter the grammar productions (enter  end to stop)
S->CC
C->cC
C->b
end
        FIRST AND FOLLOW OF NON-TERMINALS
S
        First:   {'b', 'c'}
        Follow:  {'$'}

C
        First:   {'b', 'c'}
        Follow:  {'$', 'c', 'b'}

['S', 'C']
['c', 'b', '$']
Item0:
         Z->.S, $
         S->.CC, $
         C->.cC, b|c
         C->.b, b|c
Item1:
         Z->S., $
Item2:
         S->C.C, $
         C->.cC, $
         C->.b, $
Item3:
         C->c.C, b|c
         C->.cC, b|c
         C->.b, b|c
```

jupyter CDlab11 Last Checkpoint: 17 hours ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Code

```
['S', 'C']
['c', 'b', '$']
Item0:
         Z->.S, $
         S->.CC, $
         C->.cC, b|c
         C->.b, b|c
Item1:
         Z->S., $
Item2:
         S->C.C, $
         C->.cC, $
         C->.b, $
Item3:
         C->c.C, b|c
         C->.cC, b|c
         C->.b, b|c
Item4:
         C->b., b|c
Item5:
         S->CC., $
Item6:
         C->c.C, $
         C->.cC, $
         C->.b, $
Item7:
         C->b., $
Item8:
         C->cC., b|c
Item9:
         C->cC., $

        CLR(1) TABLE
```

jupyter CDlab11 Last Checkpoint: Last Friday at 11:11 PM (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Code

```
Item7:
        C->c.C, $
        C->.cC, $
        C->.d, $
Item8:
        C->d., $
Item9:
        C->cC., c|d
Item10:
        C->cC., $

        CLR(1) TABLE

            $   C   S      c      d
0               2   1   {s3}   {s4}
1   accept
2               5       {s6}   {s7}
3               8       {s3}   {s4}
4                       {r3}   {r3}
5     {r1}
6               9       {s6}   {s7}
7     {r3}
8                       {r2}   {r2}
9     {r2}

  0 s/r conflicts | 0 r/r conflicts
```

In [ ]:

Frequently Asked Questions (FAQs)

Q1. What is CLR parser?

Ans. CLR stands for canonical LR parser.

Q2. In CLR parsing canonical collection of _____ items are formed.

Ans. LR(1)

Q3.Item sets having different look ahead symbols are same in CLR. (True/False)

Ans. False.

Practical-12

April 10, 2018

Aim:

Write a program to implement LALR parser.

Input:

The input to this program is the file containing the production rules of the grammar.

Expected Output:

The expected output of this program consists of the following:

- The rules of the grammar read from the file.
- Canonical collection of LALR(1) items.
- FIRST and FOLLOW of symbols.
- LALR table.

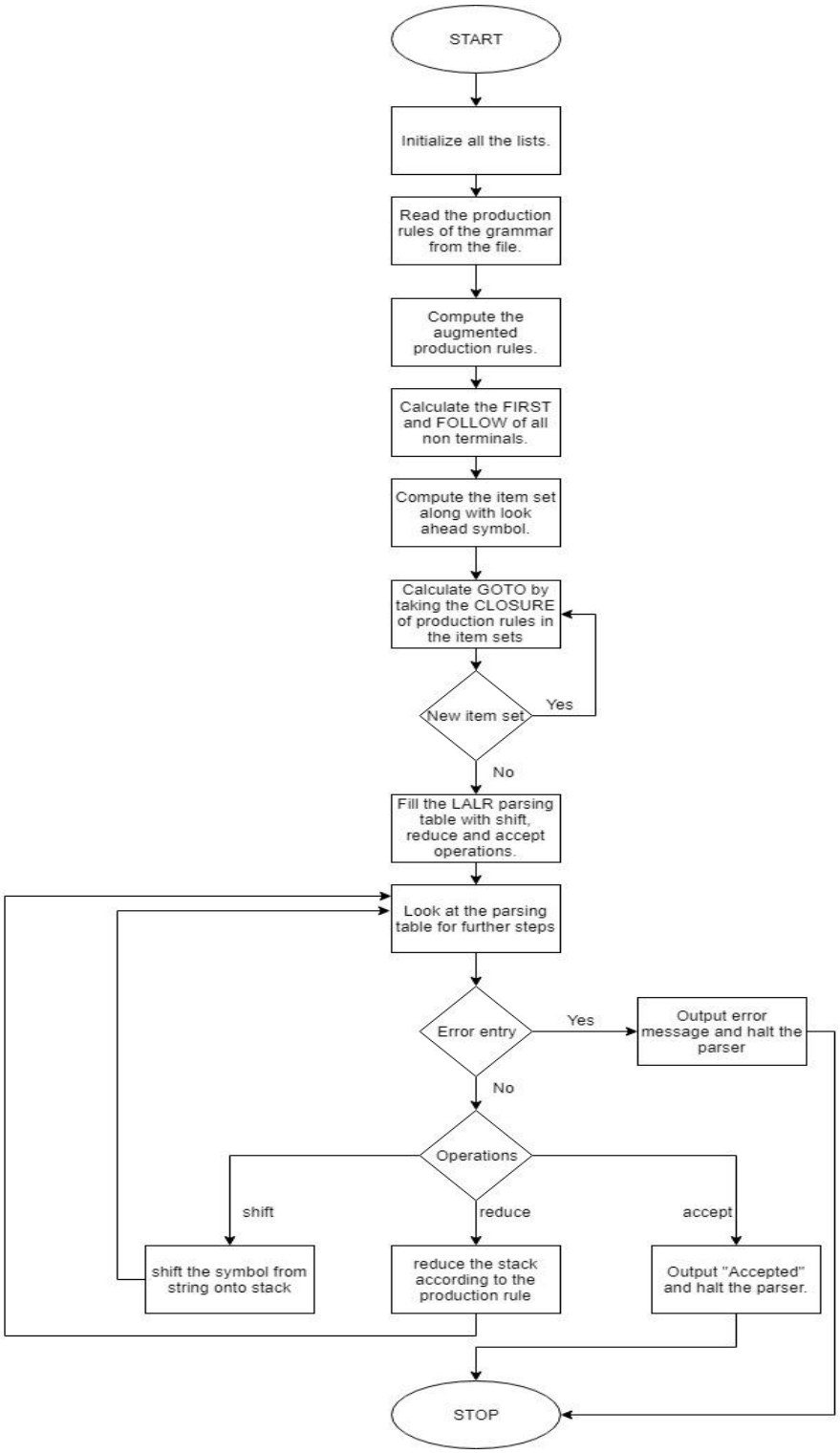
Algorithm:

1. START
2. Initialize all the lists.
3. Read the production rules of the grammar from the text file.
4. Compute the augmented production rules.
5. Calculate FIRST and FOLLOW of all the non-terminals present in the grammar.
6. Compute the item set along with the look ahead symbols.
7. Calculate GOTO operation by taking CLOSURE of the production rule with LHS having any non-terminal after the dot symbol in the item set.
8. If the result is any new item set then mark it as new set and calculate GOTO for it.
9. Fill the LALR parsing table with shift, reduce and accept operations using these item sets generated.
10. Perform the parsing using this table.
    i. If there is shift operation shift the symbol onto the stack.
    ii. If there is reduce operation reduce the expression in the stack.
    iii. If the accept entry is encountered then output “Accepted” and halt the parser.

iv. If an error entry is encountered then throw an error message and halt the parser.

11. STOP

Flowchart:



Source Code and Output:

jupyter CDlab12 Last Checkpoint: 11 hours ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Program to implement LALR parser

In [3]:
```
import pandas as pd
import firstfollow2
from firstfollow2 import production_list, nt_list as ntl, t_list as tl
from collections import OrderedDict
nt_list, t_list = [], []
```

In [4]:
```
class State:
    _id = 0
    def __init__(self, closure):
        self.closure = closure
        self.no = State._id
        State._id += 1
class Item(str):
    def __new__(cls, item, lookahead=list()):
        self = str.__new__(cls, item)
        self.lookahead = lookahead
        return self
    def __str__(self):
        return super(Item, self).__str__() + ", " + '|'.join(self.lookahead)
```

In [5]:
```
def closure(items):
    def exists(newitem, items):
        for i in items:
            if i == newitem and sorted(set(i.lookahead)) == sorted(set(newitem.lookahead)):
                return True
        return False
    global production_list
```

jupyter CDlab12 Last Checkpoint: 11 hours ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
    while True:
        flag = 0
        for i in items:
            if i.index('.') == len(i) - 1: continue
            Y = i.split('->')[1].split('.')[1][0]
            if i.index('.') + 1 < len(i) - 1:
                lastr = list(firstfollow2.compute_first(i[i.index('.') + 2]) - set(chr(1013)))
            else:
                lastr = i.lookahead
            for prod in production_list:
                head, body = prod.split('->')
                if head != Y: continue
                newitem = Item(Y + '->.' + body, lastr)
                if not exists(newitem, items):
                    items.append(newitem)
                    flag = 1
        if flag == 0: break
    return items
```

In [6]:
```
from collections import OrderedDict
def goto(items, symbol):
    global production_list
    initial = []
    for i in items:
        if i.index('.') == len(i) - 1: continue
        head, body = i.split('->')
        seen, unseen = body.split('.')
        if unseen[0] == symbol and len(unseen) >= 1:
            initial.append(Item(head + '->' + seen + unseen[0] + '.' + unseen[1:], i.lookahead))
    return closure(initial)
```

In [7]:
```
def calc_states():
    def contains(states, t):
        for s in states:
```

jupyter CDlab12 Last Checkpoint: 11 hours ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Markdown

In [7]:
```
def calc_states():
    def contains(states, t):
        for s in states:
            if len(s) != len(t): continue
            if sorted(s) == sorted(t):
                for i in range(len(s)):
                    if s[i].lookahead != t[i].lookahead: break
                else:
                    return True
        return False
    global production_list, nt_list, t_list
    head, body = production_list[0].split('->')
    states = [closure([Item(head + '->.' + body, ['$'])])]
    while True:
        flag = 0
        for s in states:
            for e in nt_list + t_list:
                t = goto(s, e)
                if t == [] or contains(states, t): continue
                states.append(t)
                flag = 1
        if not flag: break
    return states
```

In [8]:
```
def make_table(states):
    global nt_list, t_list
    def getstateno(t):
        for s in states:
            if len(s.closure) != len(t): continue
            if sorted(s.closure) == sorted(t):
                for i in range(len(s.closure)):
                    if s.closure[i].lookahead != t[i].lookahead: break
                else:
                    return s.no
```

jupyter CDlab12 Last Checkpoint: 11 hours ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Markdown

```
        return -1
    def getprodno(closure):
        closure = ''.join(closure).replace('.', '')
        return production_list.index(closure)
    SLR_Table = OrderedDict()
    for i in range(len(states)):
        states[i] = State(states[i])
    for s in states:
        SLR_Table[s.no] = OrderedDict()
        for item in s.closure:
            head, body = item.split('->')
            if body == '.':
                for term in item.lookahead:
                    if term not in SLR_Table[s.no].keys():
                        SLR_Table[s.no][term] = {'r' + str(getprodno(item))}
                    else:
                        SLR_Table[s.no][term] |= {'r' + str(getprodno(item))}
                continue
            nextsym = body.split('.')[1]
            if nextsym == '':
                if getprodno(item) == 0:
                    SLR_Table[s.no]['$'] = 'accept'
                else:
                    for term in item.lookahead:
                        if term not in SLR_Table[s.no].keys():
                            SLR_Table[s.no][term] = {'r' + str(getprodno(item))}
                        else:
                            SLR_Table[s.no][term] |= {'r' + str(getprodno(item))}
                continue
            nextsym = nextsym[0]
            t = goto(s.closure, nextsym)
            if t != []:
                if nextsym in t_list:
                    if nextsym not in SLR_Table[s.no].keys():
```

jupyter CDlab12 Last Checkpoint: 11 hours ago (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Markdown

```
                    SLR_Table[s.no][nextsym] = {'s' + str(getstateno(t))}
                else:
                    SLR_Table[s.no][nextsym] |= {'s' + str(getstateno(t))}
            else:
                SLR_Table[s.no][nextsym] = str(getstateno(t))
    return SLR_Table
```

In [9]:
```
def augment_grammar():
    for i in range(ord('Z'), ord('A') - 1, -1):
        if chr(i) not in nt_list:
            start_prod = production_list[0]
            production_list.insert(0, chr(i) + '->' + start_prod.split('->')[0])
            return
lalrdict={}
```

In [10]:
```
def main():
    global production_list, ntl, nt_list, tl, t_list,table
    firstfollow2.main()
    print("\tFIRST AND FOLLOW OF NON-TERMINALS")
    for nt in ntl:
        firstfollow2.compute_first(nt)
        firstfollow2.compute_follow(nt)
        print(nt)
        print("\tFirst:\t", firstfollow2.get_first(nt))
        print("\tFollow:\t", firstfollow2.get_follow(nt), "\n")
    augment_grammar()
    nt_list = list(ntl.keys())
    t_list = list(tl.keys()) + ['$']
    j = calc_states()
    ctr = 0
    for s in j:
        t=[]
        for i in s:
```

jupyter CDlab12 Last Checkpoint: 11 hours ago (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Markdown

```
            t.append(i)
        lalrdict[ctr]=t
        ctr += 1
    table = make_table(j)
    sr, rr = 0, 0
    new = {}
    for k, v in table.items():
        new[k] = dict(v)
    return
```

In [11]:
```
if __name__ == "__main__":
    main()
lalrdict2={}
item=[]
lookahead=[]
k=lalrdict.keys()
for i in k:
    it = []
    lo = []
    for j in range(len(lalrdict[i])):
        it.append(str(lalrdict[i][j]).split(',')[0])
        lo.append(str(lalrdict[i][j]).split(',')[1])
    item.append(it)
    lookahead.append(lo)
```

```
['S->CC', 'C->cC', 'C->b', 'end']
['S->CC', 'C->cC', 'C->b']
        FIRST AND FOLLOW OF NON-TERMINALS
S
        First:   {'b', 'c'}
        Follow:  {'$'}

C
        First:   {'b', 'c'}
```

jupyter CDlab12 Last Checkpoint: 11 hours ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [11]:
```
if __name__ == "__main__":
    main()
lalrdict2={}
item=[]
lookahead=[]
k=lalrdict.keys()
for i in k:
    it = []
    lo = []
    for j in range(len(lalrdict[i])):
        it.append(str(lalrdict[i][j]).split(',')[0])
        lo.append(str(lalrdict[i][j]).split(',')[1])
    item.append(it)
    lookahead.append(lo)
```

```
['S->CC', 'C->cC', 'C->b', 'end']
['S->CC', 'C->cC', 'C->b']
        FIRST AND FOLLOW OF NON-TERMINALS
S
        First:   {'b', 'c'}
        Follow:  {'$'}

C
        First:   {'b', 'c'}
        Follow:  {'$', 'b', 'c'}
```

In [12]:
```
def findmatch(l1,index1):
    for i in range((index1+1),len(l1)):
        var=[]
        for j in range(len(l1[i])):
            var.append(l1[i][j])
        if l1[index1]==l1[i]:
            return i
```

jupyter CDlab12 Last Checkpoint: 11 hours ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Notebook saved Trusted Python 3

In [12]:
```
def findmatch(l1,index1):
    for i in range((index1+1),len(l1)):
        var=[]
        for j in range(len(l1[i])):
            var.append(l1[i][j])
        if l1[index1]==l1[i]:
            return i
same=[]
save=[]
```

In [13]:
```
for i in range(len(item)):
    val=findmatch(item,i)
    if val in range(len(item)):
        k=str(i)+str(val)
        var = []
        for j in range(len(item[i])):
            var.append(item[i][j]+','+lookahead[i][j]+'|'+lookahead[val][j])
        lalrdict2[k]=var
        same.append(i)
        save.append(val)
    else:
        var1 = []
        for j in range(len(item[i])):
            var1.append(item[i][j]+','+lookahead[i][j])
        lalrdict2[i]=var1
```

In [14]:
```
for i in save:
    del lalrdict2[i]
new = {}
for k, v in table.items():
    new[k] = dict(v)
lalrtable=[]
for i in new:
```

jupyter CDlab12 Last Checkpoint: 11 hours ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Markdown

```
    lalrtable.append(str(i)+str(new[i]))
for j in range(len(same)):
    same1=str(same[j])
    same2=str(save[j])
    for i in range(len(lalrtable)):
        if same1 in lalrtable[i]:
            r=same1+same2
            lalrtable[i]=lalrtable[i].replace(same1,r)
        elif same2 in lalrtable[i]:
            r = same1 + same2
            lalrtable[i]=lalrtable[i].replace(same2,r)
table=[]
```

In [15]:

```
print("****************Items*****************")
for s in lalrdict2:
    print("Item{}:".format(s))
    for i in range(len(lalrdict2[s])):
        print("\t", lalrdict2[s][i])
for i in range(len(lalrtable)):
    if i in save:
        continue
    else: table.append(lalrtable[i])
print('***************LALR TABLE**************')
for i in range(len(table)):
    print(table[i])
```

```
****************Items*****************
Item0:
         Z->.S, $
         S->.CC, $
         C->.cC, b|c
         C->.b, b|c
Item1:
         Z->S., $
```

jupyter CDlab12 Last Checkpoint: 11 hours ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Markdown

```
    print(table[i])
```

```
****************Items*****************
Item0:
         Z->.S, $
         S->.CC, $
         C->.cC, b|c
         C->.b, b|c
Item1:
         Z->S., $
Item2:
         S->C.C, $
         C->.cC, $
         C->.b, $
Item36:
         C->c.C, b|c| $
         C->.cC, b|c| $
         C->.b, b|c| $
Item47:
         C->b., b|c| $
Item5:
         S->CC., $
Item89:
         C->cC., b|c| $
***************LALR TABLE**************
0{'S': '1', 'C': '2', 'c': {'s36'}, 'b': {'s47'}}
1{'$': 'accept'}
2{'C': '5', 'c': {'s36'}, 'b': {'s47'}}
36{'C': '89', 'c': {'s36'}, 'b': {'s47'}}
47{'b': {'r36'}, 'c': {'r36'}}
5{'$': {'r1'}}
89{'b': {'r2'}, 'c': {'r2'}}
```

In [ ]:

Frequently Asked Questions (FAQs)

Q1. What is LALR parser?

Ans. LALR stands for look ahead LR parser.

Q2. In LALR parsing canonical collection of _____ items are formed.

Ans. LALR(1)

Q3.Item sets having different look ahead symbols are same in LALR. (True/False)

Ans. True. They are merged to create a single item set.

Q4. What are the two functions used in creating canonical collection of items in LR parsers?

Ans. GOTO() and CLOSURE().

Q5. How many types of LR parsers are there? Name them.

Ans. There are basically three types of LR parsers, SLR, CLR and LALR.