**PRACTICAL NO 5**

**AIM : To implement LL parser**

**Introduction:**

A context-free grammar (CFG) is a set of recursive rewriting rules (or productions) used to generate patterns of strings. It consists of the following components:

* a set of terminal symbols
* a set of nonterminal symbols
* a set of productions (rules for replacing (or rewriting) nonterminal symbols)
* a start symbol

To generate a string of terminal symbols from a CFG, we:

* Begin with a string consisting of the start symbol;
* Apply one of the productions with the start symbol on the lefthand size, replacing the start symbol with the righthand side of the production;
* Repeat this process until all nonterminals have been replaced by terminal symbols.

**Input:**

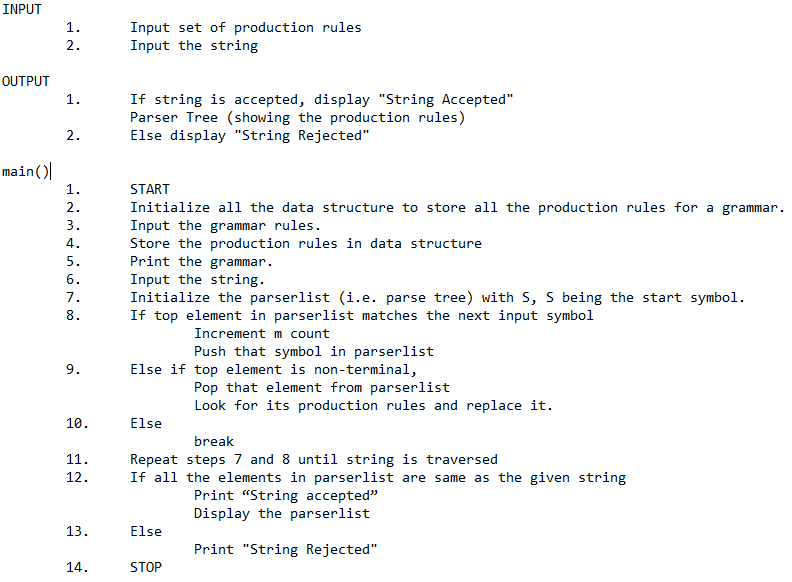
1. Input set of productions rules.
2. Input the string.

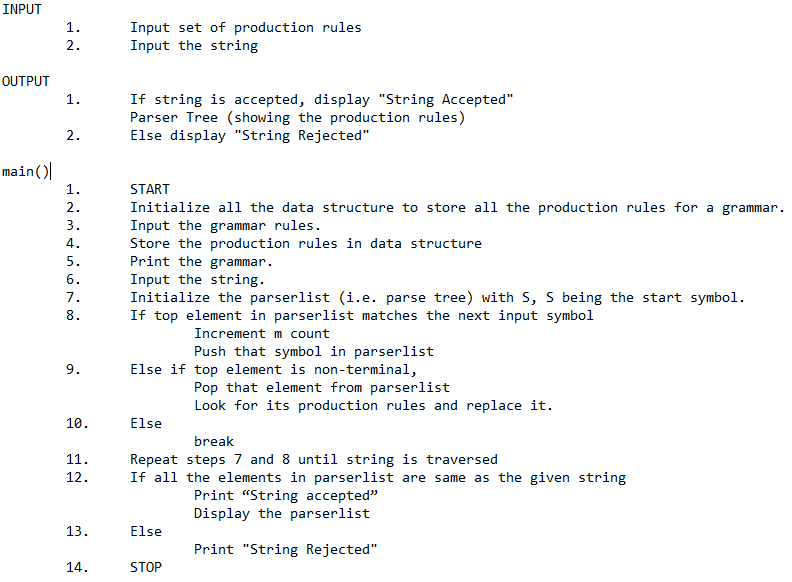
**Expected Output:**

The program should output the CFG and print

1. FIRST and FOLLOW of each non-terminal
2. Parsing Table
3. String is acceptable by the parser
   1. Parser Tree (showing the production rules used)
4. String is not acceptable by the parser.

**Algorithm 5.1: To check whether a string is accepted by CFG or not**





**Flowchart:**

Fig 5.2 Flowchart for algorithm 5.1

**Source Code:**

def findter():

global temp

global n

for k in range(n):

if temp[i]==prod[k][lhs][0]:

for t in range(int(nlist[k])):

templist = list(temp)

temp2list = []

temp2list = templist[i+1:]

templist[i:] = ""

rhslist = []

rhslist = list(prod[k][rhs][t])

templist[i:] = rhslist[:]

for ii in temp2list:

templist.append(ii)

temp = "".join(templist)

if string[i] == temp[i]:

return;

elif string[i]!=temp[i] and temp[i].isupper():

break;

break

if temp[i].isupper():

if temp not in outputlist:

parserlist.append(temp)

findter()

string=""

temp=""

lhs, rhs= 0, 1

n,z,x=0,0,0

i = 0

prod = []

nlist = []

outputlist = []

no = int(input("Enter number of production rules: "))

print("\nEnter production rules: \n")

for on in range(no) :

line = input()

listtemp = line.split()

listrhs = []

listrhs.append(listtemp[rhs])

listt = []

listt.append(listtemp[lhs])

listt.append(listrhs)

if n>0 and listt[lhs] == prod[n-1][lhs]:

prod[n-1][rhs].append(listt[rhs][0])

nlist[n-1] = str(int(nlist[n-1]) + 1)

else:

prod.append(listt)

nlist.append(str(1))

n=n+1

print("The grammar is: ")

for j in prod:

print(j[0], " -> ", " | ".join(j[1]))

while(1):

string = input("\nEnter any string (0 for exit): ")

if(string == "0"):

exit(1)

for j in range(int(nlist[0])):

parserlist = []

parserlist.append("S")

temp = prod[0][rhs][j]

m=0

for i in range(len(string)):

if i<len(temp) and string[i] == temp[i]:

m=m+1

if temp not in outputlist:

parserlist.append(temp)

elif i<len(temp) and string[i]!=temp[i] and temp[i].isupper():

findter()

if string[i]==temp[i]:

m=m+1

if temp not in outputlist:

parserlist.append(temp)

elif i<len(temp) and string[i]!=temp[i] and (ord(temp[i])<65 or ord(temp[i])>90):

break

if m==len(string) and len(string)==len(temp):

print("\nString Accepted\n")

print("We used LMD Top-Down approach\n")

print('{:>10}'.format("S =>") + '{:>5}'.format(parserlist[0]))

for rules in range(len(parserlist)-1):

print('{:>10}'.format(" =>") + '{:>5}'.format(parserlist[rules+1]))

break

if j == (int(nlist[0])-1):

print("String not Accepted")

**Output:**

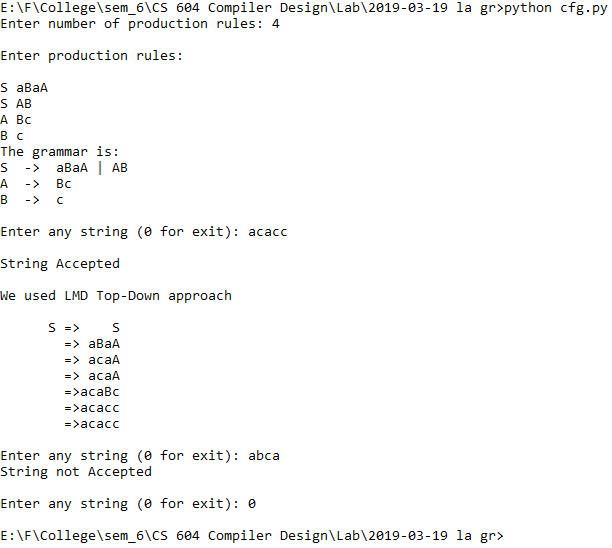


Fig 5.3 Output for checking whether string is accepted by CFG or not

**Frequently asked Questions:**

1. **What is a CFG?**

A context-free grammar (CFG) is a set of recursive rewriting rules (or productions) used to generate patterns of strings.

1. **What are the components of CFG?**

A CFG consists of the following components:

* a set of terminal symbols
* a set of nonterminal symbols
* a set of productions
* a start symbol

1. **Give one example of a CFG.**

S → AB

S → ASB

A → a

B → b