Aim:  
Write a program in C/C++ to construct Predictive Parser Table.

Algorithm:

INPUT: Grammar G. OUTPUT: Parsing table M. METHOD:

1. For each production **A ->alpha** of the grammar, do the following:  
a. For each terminal a in FIRST(A), add A to M[A, a] .  
b. If is in FIRST() , then for each terminal b in FOLLOW(A) , add A to M [A,

b] . If is in FIRST() and $ is in FOLLOW(A), add A to M[A, $] as well.  
2. If, after performing the above, there is no production at all in M[A, a], then set M[A,

a] to error (which we normally represent by an empty entry in the table). 3. End.

gram = {

"E":["E+T","T"],

"T":["T\*F","F"],

"F":["(E)","i"],

# "S":["CC"],

# "C":["eC","d"],

}

def removeDirectLR(gramA, A):

"""gramA is dictonary"""

temp = gramA[A]

tempCr = []

tempInCr = []

for i in temp:

if i[0] == A:

#tempInCr.append(i[1:])

tempInCr.append(i[1:]+[A+"'"])

else:

#tempCr.append(i)

tempCr.append(i+[A+"'"])

tempInCr.append(["e"])

gramA[A] = tempCr

gramA[A+"'"] = tempInCr

return gramA

def checkForIndirect(gramA, a, ai):

if ai not in gramA:

return False

if a == ai:

return True

for i in gramA[ai]:

if i[0] == ai:

return False

if i[0] in gramA:

return checkForIndirect(gramA, a, i[0])

return False

def rep(gramA, A):

temp = gramA[A]

newTemp = []

for i in temp:

if checkForIndirect(gramA, A, i[0]):

t = []

for k in gramA[i[0]]:

t=[]

t+=k

t+=i[1:]

newTemp.append(t)

else:

newTemp.append(i)

gramA[A] = newTemp

return gramA

def rem(gram):

c = 1

conv = {}

gramA = {}

revconv = {}

for j in gram:

conv[j] = "A"+str(c)

gramA["A"+str(c)] = []

c+=1

for i in gram:

for j in gram[i]:

temp = []

for k in j:

if k in conv:

temp.append(conv[k])

else:

temp.append(k)

gramA[conv[i]].append(temp)

#print(gramA)

for i in range(c-1,0,-1):

ai = "A"+str(i)

for j in range(0,i):

aj = gramA[ai][0][0]

if ai!=aj :

if aj in gramA and checkForIndirect(gramA,ai,aj):

gramA = rep(gramA, ai)

for i in range(1,c):

ai = "A"+str(i)

for j in gramA[ai]:

if ai==j[0]:

gramA = removeDirectLR(gramA, ai)

break

op = {}

for i in gramA:

a = str(i)

for j in conv:

a = a.replace(conv[j],j)

revconv[i] = a

for i in gramA:

l = []

for j in gramA[i]:

k = []

for m in j:

if m in revconv:

k.append(m.replace(m,revconv[m]))

else:

k.append(m)

l.append(k)

op[revconv[i]] = l

return op

result = rem(gram)

terminals = []

for i in result:

for j in result[i]:

for k in j:

if k not in result:

terminals+=[k]

terminals = list(set(terminals))

#print(terminals)

def first(gram, term):

a = []

if term not in gram:

return [term]

for i in gram[term]:

if i[0] not in gram:

a.append(i[0])

elif i[0] in gram:

a += first(gram, i[0])

return a

firsts = {}

for i in result:

firsts[i] = first(result,i)

# print(f'First({i}):',firsts[i])

def follow(gram, term):

a = []

for rule in gram:

for i in gram[rule]:

if term in i:

temp = i

indx = i.index(term)

if indx+1!=len(i):

if i[-1] in firsts:

a+=firsts[i[-1]]

else:

a+=[i[-1]]

else:

a+=["e"]

if rule != term and "e" in a:

a+= follow(gram,rule)

return a

follows = {}

for i in result:

follows[i] = list(set(follow(result,i)))

if "e" in follows[i]:

follows[i].pop(follows[i].index("e"))

follows[i]+=["$"]

# print(f'Follow({i}):',follows[i])

resMod = {}

for i in result:

l = []

for j in result[i]:

temp = ""

for k in j:

temp+=k

l.append(temp)

resMod[i] = l

# create predictive parsing table

tterm = list(terminals)

tterm.pop(tterm.index("e"))

tterm+=["d"]

pptable = {}

for i in result:

for j in tterm:

if j in firsts[i]:

pptable[(i,j)]=resMod[i[0]][0]

else:

pptable[(i,j)]=""

if "e" in firsts[i]:

for j in tterm:

if j in follows[i]:

pptable[(i,j)]="e"

pptable[("F","i")] = "i"

toprint = f'{"": <10}'

for i in tterm:

toprint+= f'|{i: <10}'

print(toprint)

for i in result:

toprint = f'{i: <10}'

for j in tterm:

if pptable[(i,j)]!="":

toprint+=f'|{i+"->"+pptable[(i,j)]: <10}'

else:

toprint+=f'|{pptable[(i,j)]: <10}'

print(f'{"-":-<76}')

print(toprint)