**Practical No. 05**

**Name: Amarsingh Kashyap**

**Roll No: 101**

**Branch: CSE 2nd shift**

**Aim: Construct the SLR parser for the given grammar**  
  
**Input:**

Grammar rules from a file or from a console entered by the user.  
**Implementation:**

Closure & go to function.  
**Output:**

SLR parser showing the states & transition.  
E -> E + T | T  
T-> T\*F | F  
F-> ( E ) | id

**CODE :**

**from** prettytable **import** PrettyTable  
**import** collections  
**from** tabulate **import** tabulate  
*# Closure & GOTO fuvtion*everyGOTO = []  
num = 0  
Iteration = list()  
epsilon = **"Є"***#Adding new elements to cluster***def** updateCluster(key, cluster, e):  
 *#print(key, cluster, e)* **if** key **in** cluster:  
 cluster[key].extend(e)  
 **else**:  
 cluster[key] = e  
 **return** cluster  
  
**def** getGOTO(State, next):  
 *#print("GOTO", State.id, next)* goto = {}  
 find = **"."**+next  
 cluster = []  
 myRules = []  
 rules = State.rule.copy()  
 lhs, rhs = list(rules.keys()), list(rules.values())  
 **for** k, v **in** zip(lhs, rhs):  
 myRules.append([k, v])  
  
 **for** rule **in** myRules:  
 *#print(rule)* **for** rhs **in** rule[1]:  
 rhs\_ = rhs.split(**" "**)  
 **if** find **in** rhs\_:  
 *#print(find,"->",rhs\_)* **if** find **in** rhs\_[:-1]:  
 *#print(find,"->",rhs\_)* at = rhs\_.index(find)  
 rhs\_[at] = find.replace(**"."**, **""**)  
 nextSym = at+1  
 looking = rhs\_[nextSym]  
 further = **"."**+looking  
 rhs\_[nextSym] = further  
 *#print(". at-", looking)* goto = updateCluster(rule[0], goto, [**" "**.join(rhs\_)])  
 *#goto[rule[0]] = [" ".join(rhs\_)]* **if** looking **in** non\_terminals:  
 cluster.append(looking)  
 **else**:  
 at = rhs\_.index(find)  
 rhs\_[at] = next+**"."** goto[rule[0]] = [**" "**.join(rhs\_)]  
 *#print(goto)* **for** reference **in** cluster:  
 clust = {}  
 getClosure(clust, reference, BASE[reference])  
 **for** c **in** clust:  
 **if** c **in** goto:  
 goto[c].extend(clust[c])  
 **else**:  
 goto[c] = clust[c]  
 *#print("GOTO:", goto)* **return** goto **if** goto **else None  
  
def** getNext(r):  
 r = r.split(**" "**)  
 DotAt = **""**.join([sym **for** sym **in** r **if "." in** sym])  
 **return** DotAt.replace(**"."**,**""**)  
  
**def** getClosure(clust, sym, rule):  
 *#print(clust, sym, rule)  
 '''added for adding new rule and checking if . is looking at any NT'''* added = []  
 processed = []  
 **if** sym **in** clust:  
 clust[sym].extend(rule)  
 **else**:  
 clust[sym] = rule  
 added.append(rule)  
  
 **for** new **in** added:  
 **for** each **in** new:  
 single = each.split(**" "**)  
 **for** e **in** single:  
 **if "." in** e[:-1]:  
 at = e.index(**"."**)  
 looking = e[at+1:]  
 **if** looking **in** non\_terminals **and** looking **not in** processed:  
  
 clust[looking] = BASE[looking]  
 processed.append(looking)  
 added.append(rules[looking])  
 *#print("returned->",clust)* **return** clust  
  
**class** State:  
 **def** \_\_init\_\_(self, r, id):  
 self.id = id  
 self.state = self.CreateState(r)  
 self.rule = r  
  
 **def** CreateState(self, rules):  
 state = []  
 **for** lhs **in** rules:  
 **for** rhs **in** rules[lhs]:  
 state.append(lhs+**"->"**+rhs)  
 **return** state  
  
 **def** getState(self):  
 print(self.id)  
 **for** r **in** self.state:  
 print(**"\t"**,r)  
  
**def** GetBase(rules):  
 base = rules.copy()  
 **return** base  
  
**def** PrintGrammar(rules):  
 **for** lhs **in** rules:  
 **for** rhs **in** rules[lhs]:  
 print(**'\t'**,lhs,**"->"**,rhs)  
  
**def** AugumentGrammar(productions):  
  
 productions.insert(0, start\_symbol + **"'->"** + start\_symbol)  
 print(productions)  
 rules = {}  
 **for** p **in** productions:  
 r = p.split(**"->"**)  
 rules[r[0]] = r[1].split(**'|'**)  
  
 **for** lhs **in** rules:  
 rhs = rules[lhs]  
 rules[lhs] = [**"."**+r.replace(epsilon, **""**) **for** r **in** rhs]  
 **return** rules  
  
  
terminals = list()  
non\_terminals = list()  
start\_symbol = **""**newSymbol = **""**productions = []  
BASE = dict()  
rules = dict()  
allS = []  
  
**def** TakeInput():  
 *# input terminal symbols* print(**"=============================================================================================="**)  
 print(**"Enter Terminals:"**)  
 **global** terminals  
 terminals = list(map(str, input().split()))  
  
 *# input non terminal symbols* print(**"=============================================================================================="**)  
 print(**"Enter Non-Terminals:"**)  
 **global** non\_terminals  
 non\_terminals = list(map(str, input().split()))  
  
 *# input start symbol* print(**"=============================================================================================="**)  
 **global** start\_symbol  
 start\_symbol = input(**"Enter the Starting Symbol: "**)  
 print(start\_symbol+**"'"**+**" should be for Augmentation!"**)  
 *# input all production rules* print(**"=============================================================================================="**)  
 no\_of\_productions = int(input(**"Enter no of Productions: "**))  
 print(**"=============================================================================================="**)  
 **global** productions  
 print(**"Enter the Production Rules:\n\t"**)  
 **for** \_ **in** range(no\_of\_productions):  
 productions.append(input().replace(**"#"**, epsilon))  
  
allS.extend(non\_terminals)  
allS.extend(terminals)  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 *#terminals = ["+", "\*", "(", ")", "id"]  
 #non\_terminals = ["E", "T", "F"]  
 #productions = ["E->E + T|T", "T->T \* F|F", "F->( E )|id"]* StateTable = PrettyTable([**"States"**])  
 TakeInput()  
 newSymbol = start\_symbol+**"'"** allS = non\_terminals+terminals  
 rules = AugumentGrammar(productions)  
 *#print(rules)* BASE = GetBase(rules)  
 print(**"=============================================================================================="**)  
 print(**"Augmented Grammar"**)  
 PrintGrammar(rules)  
 *# print(rules)* cluster = {}  
 **'''E'->.E'''** cluster = getClosure(cluster, newSymbol, rules[newSymbol])  
 I0 = State(cluster, **"I0"**)  
 entry = []  
 **for** e **in** I0.state:  
 entry.append(e + **"\n"**)  
 entry.insert(0, I0.id + **"\n"**)  
 StateTable.add\_row([**""**.join(entry)])  
  
 Iteration.append(I0)  
  
 **'''representation'''** tab = [[row] **for** row **in** I0.state]  
 tab = tabulate(tab, headers=[**"Closure("**+newSymbol+**"->."**+start\_symbol+**") = I0"**])  
 everyGOTO.append(tab)  
  
 **for** i **in** Iteration:  
 **for** each **in** allS:  
 cluster = getGOTO(i, each)  
 unique = **False  
 if** cluster:  
 newState = State(cluster.copy(), **None**)  
 **for** s **in** Iteration:  
 **if** collections.Counter(s.state) == collections.Counter(newState.state):  
 newState.id = s.id  
 **break  
 if** newState.id **is None**:  
 num = num+1  
 newState.id = **"I"**+str(num)  
 unique = **True** Header = **"GOTO("** + i.id + **","** + each + **") = "**+newState.id  
 tab = [[row] **for** row **in** newState.state]  
 tab = tabulate(tab, headers=[Header])  
  
 everyGOTO.append(tab)  
 **if** unique:  
 entry = []  
 **for** e **in** newState.state:  
 entry.append(e+**"\n"**)  
 entry.insert(0, newState.id+**"\n"**)  
 StateTable.add\_row([**""**.join(entry)])  
 Iteration.append(newState)  
  
 print(**"=============================================================================================="**)  
 **for** eachGOTO **in** everyGOTO:  
 print(eachGOTO)  
 print()  
 print(**"=============================================================================================="**)  
 print(StateTable)  
 print(**"=============================================================================================="**)

**OUTPUT :**



























