**PROGRAM – 8**

**AIM :** WAP to implement Predictive parser using First and Follow .

**PROGRAM :**

#include <bits/stdc++.h>

using namespace std ;

map<char , vector<string> > partedProds ;

set<char> ans ;

char ans\_ch ;

int ans\_i ;

map<char , vector<char> > firstOfProds ;

map<char , vector<char> > followOfProds ;

map<char , bool> containsNull ;

vector<char> nts ;

vector<char> ts ;

int p\_count = 0 ;

map<char , map<char , int> > fullParseTable ;

map<char , int> parseTable ;

vector<char> nullFirsts ;

map<string , int> prodNos ;

vector<string> partProd(string str) {

vector<string> parts ;

string temp = "" ;

nts . push\_back(str[0]) ;

for(int i = 3 ; i < str . length() ; i ++) {

if(str[i] == '|') {

if(temp == "$") {

string temp\_s = "" ;

temp\_s += str[0] ;

temp\_s += "->" ;

temp\_s += temp ;

if(prodNos . find(temp\_s) == prodNos . end())

prodNos[temp\_s] = ++ p\_count ;

containsNull[str[0]] = true ;

}

else {

string temp\_s = "" ;

temp\_s += str[0] ;

temp\_s += "->" ;

temp\_s += temp ;

if(prodNos . find(temp\_s) == prodNos . end())

prodNos[temp\_s] = ++ p\_count ;

parts . push\_back(temp) ;

}

temp = "" ;

} else {

int asc = (int)str[i] ;

if(asc < 65 || asc > 90) {

if(str[i] != '$')

ts . push\_back(str[i]) ;

}

temp = temp + str[i] ;

}

}

if(temp != "") {

if(temp == "$") {

string temp\_s = "" ;

temp\_s += str[0] ;

temp\_s += "->" ;

temp\_s += temp ;

if(prodNos . find(temp\_s) == prodNos . end())

prodNos[temp\_s] = ++ p\_count ;

containsNull[str[0]] = true ;

}

else {

string temp\_s = "" ;

temp\_s += str[0] ;

temp\_s += "->" ;

temp\_s += temp ;

if(prodNos . find(temp\_s) == prodNos . end())

prodNos[temp\_s] = ++ p\_count ;

parts . push\_back(temp) ;

}

}

return parts ;

}

void findFirst(char ch , int i , int j) {

vector<string> vec = partedProds[ch] ;

if(i == vec . size())

return ;

if(ch == vec[i][j]) {

cout << "\nProductions with left-recursion are not accepted ...\n" ;

exit(0) ;

}

int asc = (int)vec[i][j] ;

if(asc >= 65 && asc <= 90) { // Non-Terminals

char c = vec[i][j] ;

findFirst(c , 0 , 0) ;

while(containsNull[c]) {

if(j < vec[i] . length() - 1) {

c = vec[i][++ j] ;

int asc = (int)c ;

if(asc >= 65 && asc <= 90)

findFirst(c , 0 , 0) ;

else {

string temp\_s = "" ;

temp\_s += ans\_ch ;

temp\_s += "->" ;

vector<string> vec1 = partedProds[ans\_ch] ;

temp\_s += vec1[ans\_i] ;

parseTable[c] = prodNos[temp\_s] ;

ans . insert(c) ;

break ;

}

} else {

ans . insert('$') ;

nullFirsts . push\_back(ch) ;

break ;

}

}

} else { // Terminals

string temp\_s = "" ;

temp\_s += ans\_ch ;

temp\_s += "->" ;

vector<string> vec1 = partedProds[ans\_ch] ;

temp\_s += vec1[ans\_i] ;

parseTable[vec[i][j]] = prodNos[temp\_s] ;

ans . insert(vec[i][j]) ;

}

if(ch == ans\_ch)

ans\_i = i + 1 ;

findFirst(ch , i + 1 , 0) ;

}

void displayFirsts() {

cout << "\nDisplaying the firsts -\n" ;

for(map<char , vector<char> > :: iterator it = firstOfProds . begin() ; it != firstOfProds . end() ; it ++) {

cout << "First(" << (\*it) . first << ") : { " ;

vector<char> f = (\*it) . second ;

for(int j = 0 ; j < f . size() ; j ++) {

cout << f[j] ;

if(j != f . size() - 1)

cout << " , " ;

}

cout << " }\n" ;

}

}

void findFollow(char ch) {

if(followOfProds . find(ch) != followOfProds . end()) // Follow already calculated

return ;

set<char> ans1 ;

if(ch == 'S')

ans1 . insert('#') ;

for(map<char , vector<string> > :: iterator it = partedProds . begin() ; it != partedProds . end() ; it ++) {

char ch1 = (\*it) . first ;

vector<string> prods = (\*it) . second ;

for(int i = 0 ; i < prods . size() ; i ++) {

string prod = prods[i] ;

int pos = prod . find(ch) ;

if(pos != string::npos) { // ch found in production

if(pos == prod . length() - 1) {

if(ch != ch1) {

findFollow(ch1) ;

vector<char> follow\_ch1 = followOfProds[ch1] ;

for(int j = 0 ; j < follow\_ch1 . size() ; j ++) {

ans1 . insert(follow\_ch1[j]) ;

}

}

} else {

label :

int asc1 = (int)prod[pos + 1] ;

if(asc1 >= 65 && asc1 <= 90) { // Non-terminals

vector<char> first\_temp = firstOfProds[prod[pos + 1]] ;

for(int j = 0 ; j < first\_temp . size() ; j ++) {// Adding first to follow

if(first\_temp[j] != '$')

ans1 . insert(first\_temp[j]) ;

}

if(find(first\_temp . begin() , first\_temp . end() , '$') != first\_temp . end()) {// $ present

if(pos + 2 < prod . length()) {

pos ++ ;

goto label ;

} else {

if(ch != ch1) {

findFollow(ch1) ;

vector<char> follow\_ch1 = followOfProds[ch1] ;

for(int j = 0 ; j < follow\_ch1 . size() ; j ++) {

ans1 . insert(follow\_ch1[j]) ;

}

}

}

}

} else {

ans1 . insert(prod[pos + 1]) ;

}

}

}

}

}

vector<char> ans3 ;

for(set<char> :: iterator it = ans1 . begin() ; it != ans1 . end() ; it ++)

ans3 . push\_back(\*it) ;

followOfProds[ch] = ans3 ;

}

void displayFollows() {

cout << "\nDisplaying the follows -\n" ;

for(map<char , vector<char> > :: iterator it = followOfProds . begin() ; it != followOfProds . end() ; it ++) {

cout << "Follow(" << (\*it) . first << ") : { " ;

vector<char> f = (\*it) . second ;

for(int j = 0 ; j < f . size() ; j ++) {

cout << f[j] ;

if(j != f . size() - 1)

cout << " , " ;

}

cout << " }\n" ;

}

}

void printStkQ(stack<char> stk , queue<char> q) {

stack<char> temp\_stk ;

int n = stk . size() ;

for(int j = 0 ; j < n ; j ++) {

temp\_stk . push(stk . top()) ;

stk . pop() ;

}

for(int j = 0 ; j < n ; j ++) {

cout << temp\_stk . top() ;

stk . push(temp\_stk . top()) ;

temp\_stk . pop() ;

}

cout << "\t" ;

queue<char> temp\_q ;

n = q . size() ;

for(int j = 0 ; j < n ; j ++) {

cout << q . front() ;

temp\_q . push(q . front()) ;

q . pop() ;

}

q = temp\_q ;

}

void parseStr(string str) {

stack<char> stk ;

queue<char> q ;

stk . push('$') ;

stk . push('S') ;

cout << "\nStack\tString\tProduction no.\n\n" ;

for(int i = 0 ; i < str . length() ; i ++)

q . push(str[i]) ;

q . push('$') ;

while(! stk . empty() && ! q . empty()) {

char ch1 = stk . top() ;

char ch2 = q . front() ;

if(ch1 == ch2) {

printStkQ(stk , q) ;

stk . pop() ;

q . pop() ;

cout << "\t\_\n" ;

} else {

if(ch1 < 65 || ch1 > 90) {

cout << "Invalid string ...\n" ;

exit(0) ;

} else {

map<char , int> pr = fullParseTable[ch1] ;

if(pr . find(ch2) == pr . end()) { // No such production exists

cout << "Invalid string ...\n" ;

exit(0) ;

} else {

int prod\_num = pr[ch2] ;

string prod\_str ;

for(map<string , int> :: iterator it = prodNos . begin() ; it != prodNos . end() ; it ++) {

if((\*it) . second == prod\_num) {

prod\_str = (\*it) . first ;

break ;

}

}

printStkQ(stk , q) ;

stk . pop() ;

if(prod\_str[3] != '$') {

for(int z = prod\_str . length() - 1 ; z >= 3 ; z --)

stk . push(prod\_str[z]) ;

}

cout << "\t" << prod\_num << "\n" ;

}

}

}

}

cout << "Valid string ...\n" ;

}

int main() {

cout << "Enter the number of productions : " ;

int n ;

cin >> n ;

vector<string> prods(n) ;

for(int i = 0 ; i < n ; i ++) {

cin >> prods[i] ;

}

for(int i = 0 ; i < n ; i ++)

containsNull[prods[i][0]] = false ;

for(int i = 0 ; i < n ; i ++) {

vector<string> temp = partProd(prods[i]) ;

partedProds[prods[i][0]] = temp ;

}

ts . push\_back('#') ;

for(int i = 0 ; i < n ; i ++) {

char temp\_ch = prods[i][0] ;

ans\_ch = temp\_ch ;

ans\_i = 0 ;

findFirst(temp\_ch , 0 , 0) ;

fullParseTable[temp\_ch] = parseTable ;

parseTable . clear() ;

if(containsNull[temp\_ch]) {

nullFirsts . push\_back(temp\_ch) ;

ans . insert('$') ;

}

vector<char> ans2 ;

for(set<char> :: iterator it = ans . begin() ; it != ans . end() ; it ++)

ans2 . push\_back(\*it) ;

firstOfProds[temp\_ch] = ans2 ;

ans . clear() ;

}

for(int i = 0 ; i < n ; i ++)

findFollow(prods[i][0]) ;

for(int i = 0 ; i < nullFirsts . size() ; i ++) {

char temp\_ch = nullFirsts[i] ;

vector<char> ch\_follow = followOfProds[temp\_ch] ;

for(int k = 0 ; k < ch\_follow . size() ; k ++) {

map<char , int> temp\_p\_table = fullParseTable[temp\_ch] ;

string temp\_s = "" ;

temp\_s += temp\_ch ;

temp\_s += "->$" ;

temp\_p\_table[ch\_follow[k]] = prodNos[temp\_s] ;

fullParseTable[temp\_ch] = temp\_p\_table ;

}

}

displayFirsts() ;

displayFollows() ;

cout << "\nProduction numbers -\n" ;

for(map<string , int> :: iterator it = prodNos . begin() ; it != prodNos . end() ; it ++)

cout << (\*it) . second << " : " << (\*it) . first << "\n" ;

cout << "\nParsing table -\n" ;

cout << "\t" ;

for(int i = 0 ; i < ts . size() ; i ++) {

cout << ts[i] << "\t" ;

}

cout << "\n" ;

for(int i = 0 ; i < nts . size() ; i ++) {

char p\_ch = nts[i] ;

cout << p\_ch << "\t" ;

map<char , int> p\_table = fullParseTable[p\_ch] ;

for(int j = 0 ; j < ts . size() ; j ++) {

char p\_ch1 = ts[j] ;

if(p\_table . find(p\_ch1) != p\_table . end())

cout << p\_table[p\_ch1] << "\t" ;

else

cout << "\_\t" ;

}

cout << "\n" ;

}

cout << "\nInput string to parse -\n" ;

string parse\_str ;

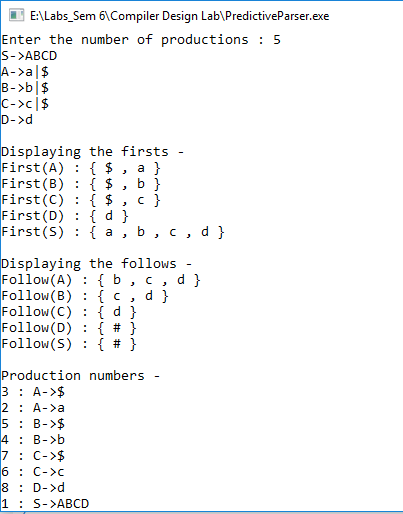
cin >> parse\_str ;

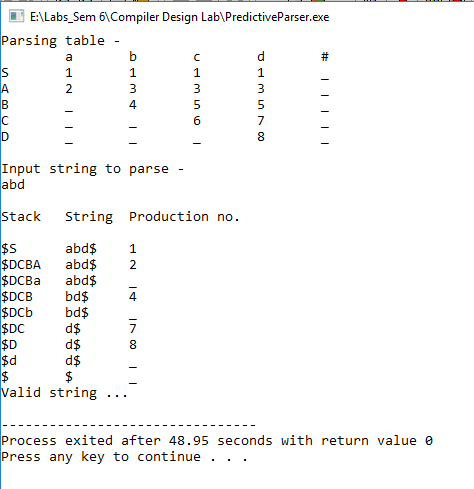
parseStr(parse\_str) ;

return 0 ;

}

**Output :**

****

****