Practical 3

Name: Rajatkumar Patel
Roll No.: 18BCE191

Aim

To find the First () and Follow () of a grammar.

Code

```
#include<bits/stdc++.h>
#define FAST ios_base::sync_with_stdio(false);cin.tie();cout.tie();
#define FILE_READ_IN freopen("input3.txt","r",stdin);
#define FILE_READ_OUT freopen("output.txt","w",stdout);
#define EPSILON '!'

using namespace std;

typedef long long ll;
map<char,set<char>> first,follow;
void append(set<char>& a,set<char>& b,bool excludeEpsilon=true){
   for(char c:b) {
       if(excludeEpsilon && c == EPSILON) continue;
       a.insert(c);
   }
}
void findFirst(map<char,vector<string>>& prods,set<char>
&terminals){
   set<char> done;
   for(auto i:prods){
       int cnt=0;
       for(string s:i.second){
           if(terminals.count(s[0]) || s[0]==EPSILON){
               // if terminal or epsilon
               first[i.first].insert(s[0]);
               cnt++;
           }
       }
```

```
        if(cnt == i.second.size()){
            // if all the productions are done
            done.insert(i.first);
        }
    }
    // runs until first for all the elements is computed
    while(done.size() < prods.size()){
        for(auto i:prods){

            if(done.count(i.first)) continue;
            int cnt=0;
            for(string s:i.second){
                int count=0;
                for(int j=0;j<s.length();j++){
                    if(done.count(s[j]) &&
first[s[j]].count(EPSILON)){
                        if(j+1 != s.length())

append(first[i.first],first[s[j]]),count++;
                        // if last element then add epsilon to
result
                        else
append(first[i.first],first[s[j]],false),count++;
                    }
                    else if(done.count(s[j])){
                        // append result in first of current element
                        append(first[i.first],first[s[j]]),count++;
                        break;
                    }
                    // if first is not computed for next element
then goto end
                    else goto end;
                }
                if(count > 0) cnt++;
            }
            end:
            if(cnt == prods[i.first].size()) {
                done.insert(i.first);
```

```
            }

        }
    }

}
// get first for the string and return true if first is already
computed for the input string
bool getFirst(map<char,vector<string>>& prods,set<char>
&terminals,char nonterminal,
char current,int start,set<char>& onStack,stack<char>&
st,set<char>& done){

    for(auto j:prods[nonterminal]){
        for(int k=start;k<j.length();k++){
            // if next symbol is terminal then take it and return
true
            if(terminals.count(j[k])){
                follow[current].insert(j[k]);
                return 1;
            }
            // if first contains epsilon
            if(first[j[k]].count(EPSILON)){
                // if next element is not the last element
               if(k+1 < j.length())
append(follow[current],first[j[k]]);
               else {
                   if(done.count(j[k]))
                        append(follow[current],first[j[k]]);
                    else {
                        st.push(j[k]);
                        onStack.insert(j[k]);
                        return 0;
                    }
                   // if follow is already computed for next element
or it is inside stack
                   // then append its result to the current element
```

```
                    if(done.count(nonterminal) ||
onStack.count(nonterminal)){
                        append(follow[current],follow[nonterminal]);
                    }
                    else {
                        // if current element is last element and
follow of lhs is not computed
                        // then push it on stack and return 0
                        st.push(nonterminal);
                        onStack.insert(nonterminal);
                        return 0;
                    }
                }
            }
            else {
                append(follow[current],first[j[k]]);
                return 1;
            }
        }
    }
    return 1;
}
// compute the follow for string
void findFollow(map<char,vector<string>>& prods,set<char>
&terminals){

    follow['S'].insert('$');    // insert $ in follow of start
symbol
    stack<char> st;
    st.push('S');

    set<char> done,onStack;
    onStack.insert('S');

    while(done.size()!=prods.size()){
        start:
        // if stack is empty, then push the remaining element in the
stack
```

```
        if(st.empty()){
            for(auto x:prods) {
                if(!done.count(x.first)){
                    st.push(x.first);
                    onStack.insert(x.first);
                    break;
                }
            }
        }
        while(!st.empty()){

            char top = st.top();
            int cnt=0;
            for(auto x:prods){
                for(string s:x.second){
                    for(int k=0;k<s.length();k++){
                        if(s[k]==st.top()){
                            cnt++;
                            // if next element is terminal
                            if(k+1 < s.length() &&
terminals.count(s[k+1])){
                                follow[st.top()].insert(s[k+1]);
                            }
                            // if next element is non terminal
                            else if(k+1 < s.length() &&
!terminals.count(s[k+1])){
                                bool res =
getFirst(prods,terminals,x.first,st.top(),k+1,onStack,st,done);
                                if(res) {

done.insert(st.top()),st.pop(),onStack.erase(top);
                                }
                                goto start;
                            }
                            // if no further element is left then
take follow of left hand side
                            else if(k+1 == s.length()){
```

```
                                if(done.count(x.first))
append(follow[st.top()],follow[x.first]);
                                else if(onStack.count(x.first)){

append(follow[st.top()],follow[x.first]);
                                }
                                else {
                                    onStack.insert(x.first);
                                    st.push(x.first);
                                    goto start;
                                }
                            }
                        }
                    }
                }
            }
            // if we are done with current element then pop it from
the stack
            if(st.top()==top)
done.insert(top),onStack.erase(top),st.pop();
        }
    }

}

void print(map<char,set<char>>& mp){
    for(auto i:mp){
        cout<<i.first<<"->{";
        for(char j:i.second){
            cout<<j<<",";
        }
        cout<<"}\n";
    }
    cout<<"----------------------------------------\n";
}

int main(){
  #ifndef ONLINE_JUDGE
```

```
   FILE_READ_IN
   FILE_READ_OUT
#endif

 // finding first and follow of the grammar

 /*
     S  ->  EX
     X ->  epsilon/+S
     E  ->  number | (S)

     S  ->  EX
     X ->  !/+S
     E  ->  @ | (S)

 */
int n; cin>>n;
map<char,vector<string>> prods;

for(int i=0;i<n;i++){
    char input_symbol; cin>>input_symbol;
    string production; cin>>production;
    prods[input_symbol].push_back(production);
}
int number_of_terminals;
cin>>number_of_terminals;
set<char> terminals;
for(int i=0;i<number_of_terminals;i++) {
    char terminal; cin>>terminal;
    terminals.insert(terminal);
}
cout<<"first: \n";
 findFirst(prods,terminals);
 findFollow(prods,terminals);
print(first);
cout<<"follow: \n";
print(follow);
```

```
  return 0;
}
```

Input

input3.txt

```
5
S EX
X !
X +S
E @
E (S)
4
+ @ ( )
```

** ! = epsilon and @ represents id

Output

output.txt

```
first:
E->{(,@,}
S->{(,@,}
X->{!,+,}
-----------------------------------
follow:
E->{$,),+,}
S->{$,),}
X->{$,),}
-----------------------------------
```

Conclusion

Computed first and follow sets which is one of the crucial information needed to compute the parse table for top-down parsers.