SLR PARSER

```
#include <iostream>
#include<unordered_map>
#include<unordered_set>
#include<queue>
#include<stack>
#include<map>
#include <string.h>
#include <vector>
using namespace std;
vector < pair < char, string > > prod, sets[20];
int numStates, terminalcount;
vector < char > nt, term;
map < char, int > terminalIndexMap, nonTerminalIndexMap;
int table[20][20];
vector < int > transition_table[20][20];
queue < int > Q1;
vector < char > first[10], follow[10];
vector < char > ep;
int visited[10];

bool is_nonterm(char c) {
  if (c >= 'A' && c <= 'Z') {
    return 1;
  } else {
    return 0;
  }
}
void make_set(pair < char, string > b) {
  queue < pair < char, string > > Q;
  pair < char, string > x;
  if (find(sets[numStates].begin(), sets[numStates].end(), b) == sets[numStates].end())
{
    sets[numStates].push_back(b);
    Q.push(b);
  }
  while (!Q.empty()) {
    x = Q.front();
    Q.pop();
    int pos, f = 1;
    for (int i = 0; i < x.second.size(); i++) {
      if (x.second[i] == '.') {
          pos = i;
```

```
      break;
     }
    }
    if (x.second.size() == pos + 1) {
     f = 0;
    }
    for (int i = 0; f && i < prod.size(); i++) {
     if (prod[i].first == x.second[pos + 1]) {
      string d = ".";
      d = d + prod[i].second;
      pair < char, string > y = make_pair(prod[i].first, d);
      if (find(sets[numStates].begin(), sets[numStates].end(), y) ==
sets[numStates].end()) {
       sets[numStates].push_back(y);
       Q.push(y);
      }
     }
    }
  }
}
int check() {
  for (int i = 0; i < numStates; i++) {
   if (sets[i] == sets[numStates]) {
    return i;
   }
  }
  return -1;
}
void gotoFunction(char targetSymbol, int currentStateIndex) {
  for (int itemIndex = 0; itemIndex < sets[currentStateIndex].size(); itemIndex++) {
   int dotPosition = 0, foundSymbol = 0;

   for (int charIndex = 0; charIndex < sets[currentStateIndex]
[itemIndex].second.size(); charIndex++) {
    if (sets[currentStateIndex][itemIndex].second[charIndex] == '.' &&
      charIndex + 1 < sets[currentStateIndex][itemIndex].second.size() &&
      sets[currentStateIndex][itemIndex].second[charIndex + 1] == targetSymbol) {
     dotPosition = charIndex;
     foundSymbol = 1;
     break;
    }
   }
   if (foundSymbol) {
    string updatedProduction = "";
```

```
      for (int charIndex = 0; charIndex < dotPosition; charIndex++) {
        updatedProduction += sets[currentStateIndex][itemIndex].second[charIndex];
      }

      updatedProduction += sets[currentStateIndex][itemIndex].second[dotPosition +
1];
      updatedProduction += '.';

      for (int charIndex = dotPosition + 2; charIndex < sets[currentStateIndex]
[itemIndex].second.size(); charIndex++) {
        updatedProduction += sets[currentStateIndex][itemIndex].second[charIndex];
      }

      make_set(make_pair(sets[currentStateIndex][itemIndex].first,
updatedProduction));
    }
   }

   if (sets[numStates].empty()) {
    return;
   }

   int matchingStateIndex = check();

   if (matchingStateIndex == -1) {
    table[currentStateIndex][terminalIndexMap[targetSymbol]] = numStates;

    if (find(transition_table[currentStateIndex]
[terminalIndexMap[targetSymbol]].begin(), transition_table[currentStateIndex]
[terminalIndexMap[targetSymbol]].end(), numStates) ==
transition_table[currentStateIndex][terminalIndexMap[targetSymbol]].end()) {
      transition_table[currentStateIndex]
[terminalIndexMap[targetSymbol]].push_back(numStates);
    }

    Q1.push(numStates);
    numStates++;
   } else {
    table[currentStateIndex][terminalIndexMap[targetSymbol]] = matchingStateIndex;
    sets[numStates].clear();

    if (find(transition_table[currentStateIndex]
[terminalIndexMap[targetSymbol]].begin(), transition_table[currentStateIndex]
[terminalIndexMap[targetSymbol]].end(), matchingStateIndex) ==
transition_table[currentStateIndex][terminalIndexMap[targetSymbol]].end()) {
```

```
      transition_table[currentStateIndex]
[terminalIndexMap[targetSymbol]].push_back(matchingStateIndex);
    }
  }
}
void mark_epsilon() {
  for (int i = 0; i < prod.size(); i++) {
    if (prod[i].second.size() == 0) {
      ep.push_back(prod[i].first);
    }
  }
}
void first_util(char lhs, vector < char > & v) {
  if (visited[nonTerminalIndexMap[lhs]] == 1) {
    return;
  }
  visited[nonTerminalIndexMap[lhs]] = 1;
  for (int i = 0; i < prod.size(); i++) {
    if (lhs == prod[i].first && prod[i].second.size() > 0) {
      int max_l = prod[i].second.size(), j = 0;
      while (j < max_l) {
        if (!is_nonterm(prod[i].second[j])) {
          if (find(v.begin(), v.end(), prod[i].second[j]) == v.end()) {
            v.push_back(prod[i].second[j]);
          }
          break;
        } else {
          first_util(prod[i].second[j], v);
          if (
            find(ep.begin(), ep.end(), prod[i].second[j]) == ep.end()) {
            break;
          }
          j++;
        }
      }
      if (j == max_l) {
        ep.push_back(lhs);
      }
    }
  }
}
void find_first() {
  for (int i = 0; i < nt.size(); i++) {
    memset(visited, 0, 10 * sizeof(int));
    char c = nt[i];
```

```
    first_util(c, first[nonTerminalIndexMap[nt[i]]]);
  }
}
void follow_util(char lhs, vector < char > & v) {
  if (visited[nonTerminalIndexMap[lhs]] == 1) {
    return;
  }
  visited[nonTerminalIndexMap[lhs]] = 1;
  if (lhs == prod[0].first) {
    if (find(v.begin(), v.end(), '$') == v.end()) {
      v.push_back('$');
    }
  }
  for (int i = 0; i < prod.size(); i++) {
    int max_l = prod[i].second.size();
    for (int j = 0; j < prod[i].second.size(); j++) {
      if (prod[i].second[j] == lhs) {
        j++;
        while (j < max_l) {
          if (!is_nonterm(prod[i].second[j])) {

            if (find(v.begin(), v.end(), prod[i].second[j]) == v.end()) {
              v.push_back(prod[i].second[j]);
            }
            break;
          } else {
            int index = nonTerminalIndexMap[prod[i].second[j]];
            for (int l = 0; l < first[index].size(); l++) {
              if (find(v.begin(), v.end(), first[index][l]) == v.end()) {
                v.push_back(first[index][l]);
              }
            }
            if (
              find(ep.begin(), ep.end(), prod[i].second[j]) == ep.end()) {
              break;
            }
            j++;
          }
        }
        if (j == max_l) {
          follow_util(prod[i].first, v);
        }
      }
    }
  }
```

```
}
void find_follow() {
  for (int i = 0; i < nt.size(); i++) {
    memset(visited, 0, 10 * sizeof(int));
    char c = nt[i];
    follow_util(c, follow[nonTerminalIndexMap[nt[i]]]);
  }
}
int find_prod(pair < char, string > x) {
  for (int i = 0; i < prod.size(); i++) {
    if (prod[i] == x) {
      return (i + 1);
    }
  }
  return -1;
}
void print_stack(stack < char > s1, stack < int > states) {
  string h = "";
  int a[20], i = 0;
  while (!s1.empty()) {
    h = h + s1.top();
    s1.pop();
  }
  while (!states.empty()) {
    a[i++] = states.top();
    states.pop();
  }

  for (int j = h.size() - 1; j >= 0; j--) {
    cout << a[j + 1] << h[j];
  }
  cout << a[0];
}
void parse_inputString() {
  string input;
  cout << "\n\nEnter the string : \n";
  cin >> input;
  cout << "\n";
  stack < char > s1;
  stack < int > states;
  int ptr = 0;
  states.push(0);
  while (ptr < input.size()) {
    cout << "$";
    print_stack(s1, states);
```

```
    for (int i = 0; i < 20 - 2 * s1.size() - input.size() + ptr; i++) {
      cout << " ";
    }
    for (int i = ptr; i < input.size(); i++) {
      cout << input[i];
    }
    cout << "$\n";
    int x = states.top(), y = terminalIndexMap[input[ptr]];
    if (table[x][y] > 0) {
      s1.push(input[ptr]);
      ptr++;
      states.push(table[x][y]);
    } else if (table[x][y] < 0) {
      int pn = (-1) * table[x][y];
      pn--;
      if (s1.size() < prod[pn].second.size() &&
        states.size() < prod[pn].second.size()) {
        cout << "String Rejected\n";
        exit(0);
      }
      for (int i = 0; i < prod[pn].second.size(); i++) {
        s1.pop();
        states.pop();
      }
      s1.push(prod[pn].first);
      states.push(table[states.top()][terminalIndexMap[s1.top()]]);
    } else if (table[x][y] == 0) {
      cout << "String Rejected \n";
      exit(0);
    }
  }
  cout << "\nAccepted\n";
}

int main() {
  int n;
  char c;
  string s;
  n = 6;
  char lhs[] = {
    'E',
    'E',
    'T',
    'T',
    'F',
```

```
  'F'
};
string rhs[] = {
  "E+T",
  "T",
  "T*F",
  "F",
  "(E)",
  "i"
};
for (int i = 0; i < 6; i++) {
  c = lhs[i];
  s = rhs[i];
  if (s == "%") {
    s = "";
  }
  prod.push_back(make_pair(c, s));
}
terminalcount = 0;
for (int i = 0; i < prod.size(); i++) {
  int x = 1;
  if (find(nt.begin(), nt.end(), prod[i].first) == nt.end())

  {
    nt.push_back(prod[i].first);
  }
  for (int j = 0; j < prod[i].second.size(); j++) {
    if (!is_nonterm(prod[i].second[j]) &&
      find(term.begin(), term.end(), prod[i].second[j]) == term.end()) {
      term.push_back(prod[i].second[j]);
      terminalIndexMap[prod[i].second[j]] = terminalcount++;
      cout << prod[i].second[j] << " ";
    }
  }
}
term.push_back('$');
terminalIndexMap['$'] = terminalcount++;
for (int i = 0; i < nt.size(); i++) {
  terminalIndexMap[nt[i]] = terminalcount++;
  nonTerminalIndexMap[nt[i]] = i;
  cout << nt[i] << " ";
}
cout << "\n";
mark_epsilon();
find_first();
```

```
for (int i = 0; i < nt.size(); i++) {
  cout << "first(" << nt[i] << ") = { ";
  for (int j = 0; j < first[nonTerminalIndexMap[nt[i]]].size(); j++) {
    cout << first[nonTerminalIndexMap[nt[i]]][j] << " ";
  }
  cout << "}\n";
}
cout << endl;
find_follow();
for (int i = 0; i < nt.size(); i++) {
  cout << "follow(" << nt[i] << ") = { ";
  for (int j = 0; j < follow[nonTerminalIndexMap[nt[i]]].size(); j++) {
    cout << follow[nonTerminalIndexMap[nt[i]]][j] << " ";
  }
  cout << "}\n";
}
string h = ".";
h = h + prod[0].first;
numStates = 0;
make_set(make_pair('X', h));
numStates++;
memset(table, 0, sizeof(int) * 400);
Q1.push(0);
while (!Q1.empty()) {
  int ind = Q1.front();
  Q1.pop();
  char g;
  for (int i = 0; i < nt.size(); i++) {
    g = nt[i];
    gotoFunction(g, ind);
  }
  for (int i = 0; i < term.size(); i++) {
    g = term[i];
    gotoFunction(g, ind);
  }
}

for (int i = 0; i < numStates; i++) {
  for (int j = 0; j < sets[i].size(); j++) {
    int last_i = sets[i][j].second.size();
    if (sets[i][j].second[last_i - 1] == '.') {
      string rhs = sets[i][j].second.substr(0, last_i - 1);
      cout << sets[i][j].first << " -> " << rhs << "\n";
      int prod_num =
        find_prod(make_pair(sets[i][j].first, rhs));
```

```
      if (prod_num < 0) {
        table[i][terminalIndexMap['$']] = numStates;
        continue;
      }
      prod_num = (-1) * prod_num;
      int index = nonTerminalIndexMap[sets[i][j].first];
      for (int l = 0; l < follow[index].size(); l++) {
        table[i][terminalIndexMap[follow[index][l]]] = prod_num;
        int pos = terminalIndexMap[follow[index][l]];
        if (
          find(transition_table[i][pos].begin(), transition_table[i][pos].end(), prod_num)
== transition_table[i][pos].end()) {
          transition_table[i][pos].push_back(prod_num);
        }
      }

    }
  }
 }

 cout << "\n\nSets are :\n\n";
 for (int j = 0; j < numStates; j++) {
  cout << "I" << j << " { ";
  for (int i = 0; i < sets[j].size(); i++) {
   cout << sets[j][i].first << " -> " << sets[j][i].second << "";
  }
  cout << "}\n";
 }
 int multiple_ent = 0;
 for (int i = 0; i < numStates; i++) {
  for (int j = 0; j < terminalcount; j++) {
   if (transition_table[i][j].size() > 1) {
    multiple_ent = 1;
    break;
   }
  }
 }
 if (multiple_ent) {
  exit(0);
 } else {
  cout << "\n\n      ACTION              | GOTO    \n\n     ";
  for (int i = 0; i < term.size(); i++) {
   cout << term[i] << "    ";
  }
  for (int i = 0; i < nt.size(); i++) {
```

```
    cout << nt[i] << "   ";
   }
   cout << "\n";
   cout << "   .....\n";
   for (int i = 0; i < numStates; i++) {
    cout << i << " ";
    if (i < 10) {
     cout << " ";
    }
    cout << ": ";
    for (int j = 0; j < terminalcount; j++) {
     if (table[i][j] == 0) {
      cout << "  ";
     } else {
      if (table[i][j] > 0) cout << "s" << table[i][j];
      else
       cout << "r" << -1 * table[i][j];
     }
     if (abs(table[i][j]) < 10 && abs(table[i][j]) >= 0) {
      cout << " ";
     }
     cout << " ";
    }
    cout << "\n";
   }

  }
  parse_inputString();
  return 0;
}
```

OUTPUT:

```
X -> E
E -> T
T -> F
F -> i
E -> E+T
T -> T*F
F -> (E)


Sets are :

I0 { X -> .EE -> .E+TE -> .TT -> .T*FT -> .FF -> .(E)F -> .i}
I1 { X -> E.E -> E.+T}
I2 { E -> T.T -> T.*F}
I3 { T -> F.}
I4 { F -> (.E)E -> .E+TE -> .TT -> .T*FT -> .FF -> .(E)F -> .i}
I5 { F -> i.}
I6 { E -> E+.TT -> .T*FT -> .FF -> .(E)F -> .i}
I7 { T -> T*.FF -> .(E)F -> .i}
I8 { F -> (E.)E -> E.+T}
I9 { E -> E+T.T -> T.*F}
I10 { T -> T*F.}
I11 { F -> (E).}


       ACTION               |  GOTO

     +   *   (   )   i   $   E   T   F
   ......................................
0  :         s4      s5      s1  s2  s3
1  : s6                  s12
2  : r2  s7      r2      r2
3  : r4  r4      r4      r4
4  :         s4      s5      s8  s2  s3
5  : r6  r6      r6      r6
6  :         s4      s5          s9  s3
7  :         s4      s5              s10
8  : s6          s11
9  : r1  s7      r1      r1
10 : r3  r3      r3      r3
11 : r5  r5      r5      r5


Enter the string :
ii*i

$0                ii*i$
$0i5               i*i$
String Rejected
```

```
Enter the string :
(i)+(i)*i

$0           (i)+(i)*i$
$0(4          i)+(i)*i$
$0(4i5         )+(i)*i$
$0(4F3         )+(i)*i$
$0(4T2         )+(i)*i$
$0(4E8         )+(i)*i$
$0(4E8)11        +(i)*i$
$0F3            +(i)*i$
$0T2            +(i)*i$
$0E1            +(i)*i$
$0E1+6           (i)*i$
$0E1+6(4          i)*i$
$0E1+6(4i5         )*i$
$0E1+6(4F3         )*i$
$0E1+6(4T2         )*i$
$0E1+6(4E8         )*i$
$0E1+6(4E8)11        *i$
$0E1+6F3            *i$
$0E1+6T9            *i$
$0E1+6T9*7           i$

Accepted
```