LAB ASSIGNMENT-2

Submitted for

COMPILER CONSTRUCTION (UCS802)

Submitted by

Shivam Khurana

102103754

4 COE 27

Submitted to

Dr. Ashutosh Aggarwal

Computer Science and Engineering Department

Thapar Institute of Engineering and Technology, Patiala

QUESTION:

Design a SLR parser for the grammar given below:

E→E+T/T

T→T*F/F F→(E)/id

JAVA CODE:

```
import java.util.*;

class Action {
   String type;
   int state;

   Action(String type, int state) {
      this.type = type;
      this.state = state;
   }
}

public class Compiler {
   static Map<Integer, Map<String, Action>> actionTable = new HashMap<>();
   static Map<Integer, Map<String, Integer>> gotoTable = new HashMap<>();
   static Map<String, Set<String>> firstSet = new HashMap<>();
   static Map<String, Set<String>> followSet = new HashMap<>();
   static Map<Integer, Pair<String, Integer>> productions = new HashMap<>();

   static class Pair<T, U> {
      T first;
      U second;

      Pair(T first, U second) {
         this.first = first;
         this.second = second;
      }
   }

   static {
      actionTable.put(0, new HashMap<>() {{ put("id", new Action("shift", 5)); put("(", new
Action("shift", 4)); }});
      actionTable.put(1, new HashMap<>() {{ put("+", new Action("shift", 6)); put("$", new
Action("accept", 0)); }});
      actionTable.put(2, new HashMap<>() {{ put("+", new Action("reduce", 2)); put("*", new
Action("shift", 7)); put(")", new Action("reduce", 2)); put("$", new Action("reduce", 2)); }});
```

```
        actionTable.put(3, new HashMap<>() {{ put("+", new Action("reduce", 4)); put("*", new
Action("reduce", 4)); put(")", new Action("reduce", 4)); put("$", new Action("reduce", 4)); }});
        actionTable.put(4, new HashMap<>() {{ put("id", new Action("shift", 5)); put("(", new
Action("shift", 4)); }});
        actionTable.put(5, new HashMap<>() {{ put("+", new Action("reduce", 6)); put("*", new
Action("reduce", 6)); put(")", new Action("reduce", 6)); put("$", new Action("reduce", 6)); }});
        actionTable.put(6, new HashMap<>() {{ put("id", new Action("shift", 5)); put("(", new
Action("shift", 4)); }});
        actionTable.put(7, new HashMap<>() {{ put("id", new Action("shift", 5)); put("(", new
Action("shift", 4)); }});
        actionTable.put(8, new HashMap<>() {{ put("+", new Action("shift", 6)); put(")", new
Action("shift", 11)); }});
        actionTable.put(9, new HashMap<>() {{ put("+", new Action("reduce", 1)); put("*", new
Action("shift", 7)); put(")", new Action("reduce", 1)); put("$", new Action("reduce", 1)); }});
        actionTable.put(10, new HashMap<>() {{ put("+", new Action("reduce", 3)); put("*", new
Action("reduce", 3)); put(")", new Action("reduce", 3)); put("$", new Action("reduce", 3)); }});
        actionTable.put(11, new HashMap<>() {{ put("+", new Action("reduce", 5)); put("*", new
Action("reduce", 5)); put(")", new Action("reduce", 5)); put("$", new Action("reduce", 5)); }});

        gotoTable.put(0, new HashMap<>() {{ put("E", 1); put("T", 2); put("F", 3); }});
        gotoTable.put(4, new HashMap<>() {{ put("E", 8); put("T", 2); put("F", 3); }});
        gotoTable.put(6, new HashMap<>() {{ put("T", 9); put("F", 3); }});
        gotoTable.put(7, new HashMap<>() {{ put("F", 10); }});

        firstSet.put("E", new HashSet<>() {{ add("id"); add("("); }});
        firstSet.put("T", new HashSet<>() {{ add("id"); add("("); }});
        firstSet.put("F", new HashSet<>() {{ add("id"); add("("); }});

        followSet.put("E", new HashSet<>() {{ add(")"); add("+"); add("$"); }});
        followSet.put("T", new HashSet<>() {{ add("+"); add("*"); add(")"); add("$"); }});
        followSet.put("F", new HashSet<>() {{ add("*"); add("+"); add(")"); add("$"); }});

        productions.put(0, new Pair<>("E'", 1));
        productions.put(1, new Pair<>("E", 3));
        productions.put(2, new Pair<>("E", 1));
        productions.put(3, new Pair<>("T", 3));
        productions.put(4, new Pair<>("T", 1));
        productions.put(5, new Pair<>("F", 3));
        productions.put(6, new Pair<>("F", 1));
    }

    public static void main(String[] args) {
        printFirstAndFollowSets();
        printActionAndGotoTables();

        System.out.println("\nThe input string to parse: id + id * F");
        String[] inputTokens = {"id", "+", "id", "*", "F", "$"};
        slrParser(inputTokens);
    }

    static void printFirstAndFollowSets() {
```

```
        System.out.println("FIRST Sets:");
        for (Map.Entry<String, Set<String>> entry : firstSet.entrySet()) {
            System.out.printf("FIRST(%s) = { %s }%n", entry.getKey(), String.join(" ", entry.getValue()));
        }

        System.out.println("\nFOLLOW Sets:");
        for (Map.Entry<String, Set<String>> entry : followSet.entrySet()) {
            System.out.printf("FOLLOW(%s) = { %s }%n", entry.getKey(), String.join(" ",
entry.getValue()));
        }
    }

    static void printActionAndGotoTables() {
        String[] terminals = {"id", "+", "*", "(", ")", "$"};
        String[] nonTerminals = {"E", "T", "F"};

        System.out.println("ACTION Table:");
        System.out.print(String.format("%-10s", "State"));
        for (String term : terminals) {
            System.out.print(String.format("%-15s", term));
        }
        System.out.println();
        System.out.println("-".repeat(10 + terminals.length * 15));

        for (int state : actionTable.keySet()) {
            System.out.printf(String.format("%-10d", state));
            for (String term : terminals) {
                Action action = actionTable.get(state).get(term);
                if (action != null) {
                    if (action.type.equals("shift")) {
                        System.out.printf(String.format("%-15s", "s" + action.state));
                    } else if (action.type.equals("reduce")) {
                        System.out.printf(String.format("%-15s", "r" + action.state));
                    } else if (action.type.equals("accept")) {
                        System.out.printf(String.format("%-15s", "acc"));
                    }
                } else {
                    System.out.print(String.format("%-15s", ""));
                }
            }
            System.out.println();
        }

        System.out.println("\nGOTO Table:");
        System.out.print(String.format("%-10s", "State"));
        for (String nonTerm : nonTerminals) {
            System.out.print(String.format("%-15s", nonTerm));
        }
        System.out.println();
        System.out.println("-".repeat(10 + nonTerminals.length * 15));
```

```
        for (int state : gotoTable.keySet()) {
            System.out.printf(String.format("%-10d", state));
            for (String nonTerm : nonTerminals) {
                Integer nextState = gotoTable.get(state).get(nonTerm);
                if (nextState != null) {
                    System.out.printf(String.format("%-15d", nextState));
                } else {
                    System.out.print(String.format("%-15s", ""));
                }
            }
            System.out.println();
        }
    }

    static void slrParser(String[] tokens) {
        Deque<Integer> stateStack = new ArrayDeque<>();
        Deque<String> symbolStack = new ArrayDeque<>();

        stateStack.push(0);
        int i = 0;
        while (true) {
            int state = stateStack.peekLast();
            String token = tokens[i];

            if (!actionTable.get(state).containsKey(token)) {
                System.out.println("Status: Rejected");
                return;
            }

            Action action = actionTable.get(state).get(token);
            if (action.type.equals("shift")) {
                stateStack.push(action.state);
                symbolStack.push(token);
                i++;
            } else if (action.type.equals("reduce")) {
                Pair<String, Integer> production = productions.get(action.state);
                String productionRule = production.first;
                int popCount = production.second;
                for (int j = 0; j < popCount; j++) {
                    stateStack.pop();
                    symbolStack.pop();
                }

                String nonTerminal = productionRule;
                Integer nextState = gotoTable.get(stateStack.peekLast()).get(nonTerminal);
                stateStack.push(nextState);
                symbolStack.push(nonTerminal);
            } else if (action.type.equals("accept")) {
                System.out.println("Status: Accepted");
                return;
            }
```

```
    }
  }
}
```

OUTPUT:

```
FIRST Sets:
FIRST(T) = { ( id }
FIRST(E) = { ( id }
FIRST(F) = { ( id }

FOLLOW Sets:
FOLLOW(T) = { $ ) * + }
FOLLOW(E) = { $ ) + }
FOLLOW(F) = { $ ) * + }
```

```
ACTION Table:
State     id            +             *             (             )             $
-------------------------------------------------------------------------------------------
0         s5                                        s4
1                       s6                                                    acc
2                       r2            s7                      r2            r2
3                       r4            r4                      r4            r4
4         s5                                        s4
5                       r6            r6                      r6            r6
6         s5                                        s4
7         s5                                        s4
8                       s6                                    s11
9                       r1            s7                      r1            r1
10                      r3            r3                      r3            r3
11                      r5            r5                      r5            r5
```

```
GOTO Table:
State     E             T             F
-----------------------------------------------------
0         1             2             3
4         8             2             3
6                       9             3
7                                     10
```

```
The input string to parse: id + id * F
Status: Accepted
```

Output when the input is id + id * (:

```
The input string to parse: id + id * (
Status: Rejected
```