# LAB ASSIGNMENT-2

Submitted for

# COMPILER CONSTRUCTION (UCS802)

Submitted by

**Aryan Shanker Saxena**

**102103613**

## 4 COE 22

Submitted to

## Mr. Rajesh



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

**PYTHON CODE:**

**from** collections **import** deque

**from** typing **import** List, Dict, Tuple, Set

*class* Action:

*def* \_\_init\_\_(*self*, *type*: *str*, *state*: *int*):

        self.type **=** *type*

        self.state **=** state

# Creating ACTION and GOTO tables

action\_table: Dict[*int*, Dict[*str*, Action]] **=** {

    0: {"id": Action("shift", 5), "(": Action("shift", 4)},

    1: {"+": Action("shift", 6), "$": Action("accept", 0)},

    2: {"+": Action("reduce", 2), "\*": Action("shift", 7), ")": Action("reduce", 2), "$": Action("reduce", 2)},

    3: {"+": Action("reduce", 4), "\*": Action("reduce", 4), ")": Action("reduce", 4), "$": Action("reduce", 4)},

    4: {"id": Action("shift", 5), "(": Action("shift", 4)},

    5: {"+": Action("reduce", 6), "\*": Action("reduce", 6), ")": Action("reduce", 6), "$": Action("reduce", 6)},

    6: {"id": Action("shift", 5), "(": Action("shift", 4)},

    7: {"id": Action("shift", 5), "(": Action("shift", 4)},

    8: {"+": Action("shift", 6), ")": Action("shift", 11)},

    9: {"+": Action("reduce", 1), "\*": Action("shift", 7), ")": Action("reduce", 1), "$": Action("reduce", 1)},

    10: {"+": Action("reduce", 3), "\*": Action("reduce", 3), ")": Action("reduce", 3), "$": Action("reduce", 3)},

    11: {"+": Action("reduce", 5), "\*": Action("reduce", 5), ")": Action("reduce", 5), "$": Action("reduce", 5)},

}

goto\_table: Dict[*int*, Dict[*str*, *int*]] **=** {

    0: {"E": 1, "T": 2, "F": 3},

    4: {"E": 8, "T": 2, "F": 3},

    6: {"T": 9, "F": 3},

    7: {"F": 10},

}

productions: List[Tuple[*str*, *int*]] **=** [

    ("E'", 1),  # Dummy production to initialize the parse

    ("E", 3),   # E -> E + T

    ("E", 1),   # E -> T

    ("T", 3),   # T -> T \* F

    ("T", 1),   # T -> F

    ("F", 3),   # F -> ( E )

    ("F", 1)    # F -> id

]

# FIRST and FOLLOW sets for non-terminals

first\_set: Dict[*str*, Set[*str*]] **=** {

    "E": {"id", "("},

    "T": {"id", "("},

    "F": {"id", "("},

}

follow\_set: Dict[*str*, Set[*str*]] **=** {

    "E": {")", "+", "$"},

    "T": {"+", "\*", ")", "$"},

    "F": {"\*", "+", ")", "$"},

}

*def* print\_first\_and\_follow\_sets():

    print("FIRST Sets:")

**for** non\_terminal, symbols **in** first\_set.items():

        print(*f*"FIRST({non\_terminal}) = {{ {' '.join(symbols)} }}")

    print("\nFOLLOW Sets:")

**for** non\_terminal, symbols **in** follow\_set.items():

        print(*f*"FOLLOW({non\_terminal}) = {{ {' '.join(symbols)} }}")

*def* print\_action\_and\_goto\_tables():

    terminals **=** ["id", "+", "\*", "(", ")", "$"]

    non\_terminals **=** ["E", "T", "F"]

    print("\nACTION Table:")

    print(*f*"{'State'*:<8*}", *end***=**"")

**for** term **in** terminals:

        print(*f*"{term*:<8*}", *end***=**"")

    print("\n" **+** "-" **\*** (8 **+** len(terminals) **\*** 8))

**for** state **in** range(12):

        print(*f*"{state*:<8*}", *end***=**"")

**for** term **in** terminals:

            action **=** action\_table.get(state, {}).get(term)

**if** action:

**if** action.type **==** "shift":

                    print(*f*"s{action.state*:<7*}", *end***=**"")

**elif** action.type **==** "reduce":

                    print(*f*"r{action.state*:<7*}", *end***=**"")

**elif** action.type **==** "accept":

                    print(*f*"acc{''*:<6*}", *end***=**"")

**else**:

                print(" " **\*** 8, *end***=**"")

        print()

    print("\nGOTO Table:")

    print(*f*"{'State'*:<8*}", *end***=**"")

**for** non\_term **in** non\_terminals:

        print(*f*"{non\_term*:<8*}", *end***=**"")

    print("\n" **+** "-" **\*** (8 **+** len(non\_terminals) **\*** 8))

**for** state **in** range(12):

        print(*f*"{state*:<8*}", *end***=**"")

**for** non\_term **in** non\_terminals:

            next\_state **=** goto\_table.get(state, {}).get(non\_term, " ")

            print(*f*"{next\_state*:<8*}", *end***=**"")

        print()

*def* slr\_parser(*tokens*: List[*str*]):

    state\_stack **=** deque([0])

    symbol\_stack **=** deque()

    i **=** 0

**while** True:

        state **=** state\_stack[**-**1]

        token **=** tokens[i]

**if** token **not** **in** action\_table.get(state, {}):

            print("Status: Rejected")

**return**

        action **=** action\_table[state][token]

**if** action.type **==** "shift":

            state\_stack.append(action.state)

            symbol\_stack.append(token)

            i **+=** 1

**elif** action.type **==** "reduce":

            production\_idx **=** action.state

            prod\_len **=** productions[production\_idx][1]

            non\_terminal **=** productions[production\_idx][0]

**for** \_ **in** range(prod\_len):

                state\_stack.pop()

                symbol\_stack.pop()

            next\_state **=** goto\_table[state\_stack[**-**1]][non\_terminal]

            state\_stack.append(next\_state)

            symbol\_stack.append(non\_terminal)

**elif** action.type **==** "accept":

            print("Status: Accepted")

**return**

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    print\_first\_and\_follow\_sets()

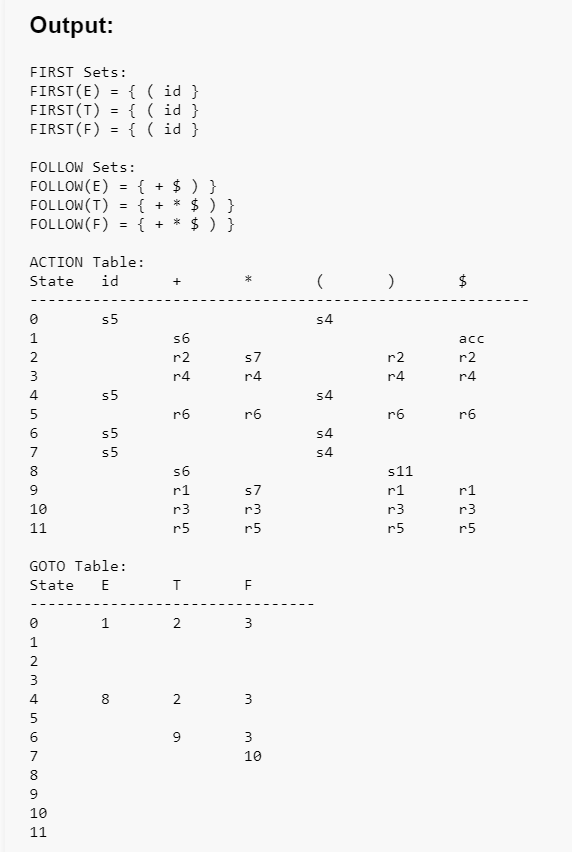
    print\_action\_and\_goto\_tables()

    print("\nThe input string to parse: id + id \* id")

    input\_tokens **=** ["id", "+", "id", "\*", "id", "$"]

    slr\_parser(input\_tokens)

**OUTPUT:**



For id \* id + id:



For id \* id + F:

