## Pandit Deendayal Energy University, Gandhinagar

## School of Technology

**Department of Computer Science & Engineering**

**System Software & Compiler Design Lab (20CP302P)**



# Name: Sutariya Smit Dharmendrabhai

Enrolment No: **21BCP142**

Semester: **V**

Division: **3 (G5)**

Branch: **Computer Science Engineering**

**Practical: 6**

**Aim:** WAP to construct operator precedence parsing table for the given grammar and check the validity of the string.

**Code:**

from tabulate import tabulate

# S->x A y |x B y |x A z

# A->a S|q

# B->q

firstop = {}

lastop = {}

productions = []

prod\_dict = {}

table\_list = []

def add\_to\_firstop(nterm, symbol):

    if nterm not in firstop:

        firstop[nterm] = set()

    firstop[nterm].add(symbol)

def add\_to\_lastop(nterm, symbol):

    if nterm not in lastop:

        lastop[nterm] = set()

    lastop[nterm].add(symbol)

def replace\_err(table):

    for i in range(len(table)):

        for j in range(len(table[i])):

            if table[i][j] == ' ':

                table[i][j] = 'err'

    return table

def parse\_expression(str):

    stack = ['$']  # Initialize the stack with '$'

    string = str.split()

    input\_buffer = list(string) + ['$']  # Append '$' to the input string

    print(input\_buffer)

    index = 0  # Index to traverse the input buffer

    while len(stack) > 0:

        top\_stack = stack[-1]

        print(top\_stack)

        current\_input = input\_buffer[index]

        top\_stack\_index = terminals.index(top\_stack)

        current\_input\_index = terminals.index(current\_input)

        relation = terminal\_matrix[top\_stack\_index][current\_input\_index]

        if relation == '<' or relation == '=':

            stack.append(current\_input)

            index += 1

        elif relation == '>':

            popped = ''

            while relation != '<':

                popped = stack.pop()  # Pop elements from the stack until '<' relation is found

                top\_stack = stack[-1] if stack else None

                top\_stack\_index = terminals.index(top\_stack) if top\_stack else None

                relation = terminal\_matrix[top\_stack\_index][terminals.index(popped)]

        elif relation == 'acc':

            print("Input string is accepted.")

            return

        else:

            print("Input string is not accepted.")

            return

no\_of\_terminals = int(input("Enter no. of terminals: "))

terminals = []

print("Enter the terminals:")

for \_ in range(no\_of\_terminals):

    terminals.append(input())

no\_of\_non\_terminals = int(input("Enter no. of non-terminals: "))

non\_terminals = []

print("Enter the non-terminals:")

for \_ in range(no\_of\_non\_terminals):

    non\_terminals.append(input())

starting\_symbol = input("Enter the starting symbol: ")

no\_of\_productions = int(input("Enter no of productions: "))

print("Enter the productions:")

for \_ in range(no\_of\_productions):

    productions.append(input())

for nT in non\_terminals:

    prod\_dict[nT] = []

for production in productions:

    nonterm\_to\_prod = production.split("->")

    alternatives = nonterm\_to\_prod[1].split("|")

    for alternative in alternatives:

        prod\_dict[nonterm\_to\_prod[0]].append(alternative)

print("Populated prod\_dict:")

for non\_terminal, prods in prod\_dict.items():

    print(f"{non\_terminal} -> {prods}")

parsing\_string = input("Enter an expression to parse: ")

# Compute firstop for each non-terminal

for non\_terminal in non\_terminals:

    for production in prod\_dict[non\_terminal]:

        symbols = production.split()

        print(symbols)

        for symbol in symbols:

            if symbol in non\_terminals:

                add\_to\_firstop(non\_terminal, symbol)

            elif symbol in terminals:

                add\_to\_firstop(non\_terminal, symbol)

                break

# Compute lastop for each non-terminal

for non\_terminal in non\_terminals:

    for production in prod\_dict[non\_terminal]:

        symbols = production.split()

        for symbol in reversed(symbols):

            if symbol in non\_terminals:

                add\_to\_lastop(non\_terminal, symbol)

            elif symbol in terminals:

                add\_to\_lastop(non\_terminal, symbol)

                break

# Print the firstop and lastop sets

print("firstop:")

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

    print(f'firstop({non\_terminal}) = {{{", ".join(first\_set)}}}')

print("lastop:")

for non\_terminal, last\_set in lastop.items():

    print(f'lastop({non\_terminal}) = {{{", ".join(last\_set)}}}')

counter=0

while counter<no\_of\_productions:

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

        first\_set\_copy = first\_set.copy()  # Create a copy of the set to iterate over

        for symbol in first\_set\_copy:

            if symbol in non\_terminals:

                firstop[non\_terminal] |= firstop[symbol]

    counter+=1

# Remove non-terminals from lastop sets

counter=0

while counter<no\_of\_productions:

    for non\_terminal, last\_set in lastop.items():

        last\_set\_copy = last\_set.copy()  # Create a copy of the set to iterate over

        for symbol in last\_set\_copy:

            if symbol in non\_terminals:

                lastop[non\_terminal] |= lastop[symbol]

    counter+=1

# Remove non-terminals from firstop sets

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

    first\_set\_copy = first\_set.copy()  # Create a copy of the set to iterate over

    for symbol in first\_set\_copy:

        if symbol in non\_terminals:

            first\_set.remove(symbol)

# Remove non-terminals from lastop sets

for non\_terminal, last\_set in lastop.items():

    last\_set\_copy = last\_set.copy()  # Create a copy of the set to iterate over

    for symbol in last\_set\_copy:

        if symbol in non\_terminals:

            last\_set.remove(symbol)

# Print the modified firstop and lastop sets

print("Firstop:")

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

    print(f'Firstop({non\_terminal}) = {{{", ".join(first\_set)}}}')

print("Lastop:")

for non\_terminal, last\_set in lastop.items():

    print(f'Lastop({non\_terminal}) = {{{", ".join(last\_set)}}}')

terminals.append('$')

terminal\_matrix = [[' ' for \_ in range(len(terminals))] for \_ in range(len(terminals))]

# Rule 1: Whenever terminal a immediately precedes non-terminal B in any production, put a <·α where α is any terminal in the firstop+ list of B

for non\_terminal in non\_terminals:

    for productions in prod\_dict[non\_terminal]:

        production = productions.split()

        for i in range(len(production) - 1):

            if production[i] in terminals and production[i + 1] in non\_terminals:

                for alpha in firstop[production[i + 1]]:

                    row\_index = terminals.index(production[i])

                    col\_index = terminals.index(alpha)

                    terminal\_matrix[row\_index][col\_index] = '<'

# Rule 2: Whenever terminal b immediately follows non-terminal C in any production, put β ·>b where β is any terminal in the lastop+ list of C

for non\_terminal in non\_terminals:

    for productions in prod\_dict[non\_terminal]:

        production = productions.split()

        for i in range(1, len(production)):

            if production[i - 1] in non\_terminals and production[i] in terminals:

                for beta in lastop[production[i - 1]]:

                    row\_index = terminals.index(beta)

                    col\_index = terminals.index(production[i])

                    terminal\_matrix[row\_index][col\_index] = '>'

# Rule 3: Whenever a sequence aBc or ac occurs in any production, put a ≐ c

for non\_terminal in non\_terminals:

    for productions in prod\_dict[non\_terminal]:

        production = productions.split()

        for i in range(1, len(production) - 1):

            if production[i - 1] in terminals and production[i + 1] in terminals:

                row\_index = terminals.index(production[i - 1])

                col\_index = terminals.index(production[i + 1])

                terminal\_matrix[row\_index][col\_index] = '='

# Rule 4: Add relations $<· a and a ·> $ for all terminals in the firstop+ and lastop+ lists, respectively of S

for alpha in firstop[starting\_symbol]:

    col\_index = terminals.index(alpha)

    terminal\_matrix[-1][col\_index] = '<'

for beta in lastop[starting\_symbol]:

    row\_index = terminals.index(beta)

    terminal\_matrix[row\_index][-1] = '>'

dollar\_index = terminals.index('$')

terminal\_matrix[-1][dollar\_index] = 'acc'

terminal\_matrix = replace\_err(terminal\_matrix)

for i in range(len(terminals)):

    row = [terminals[i]]

    row.extend([terminal\_matrix[i][j] for j in range(len(terminals))])

    table\_list.append(row)

headers = [''] + terminals

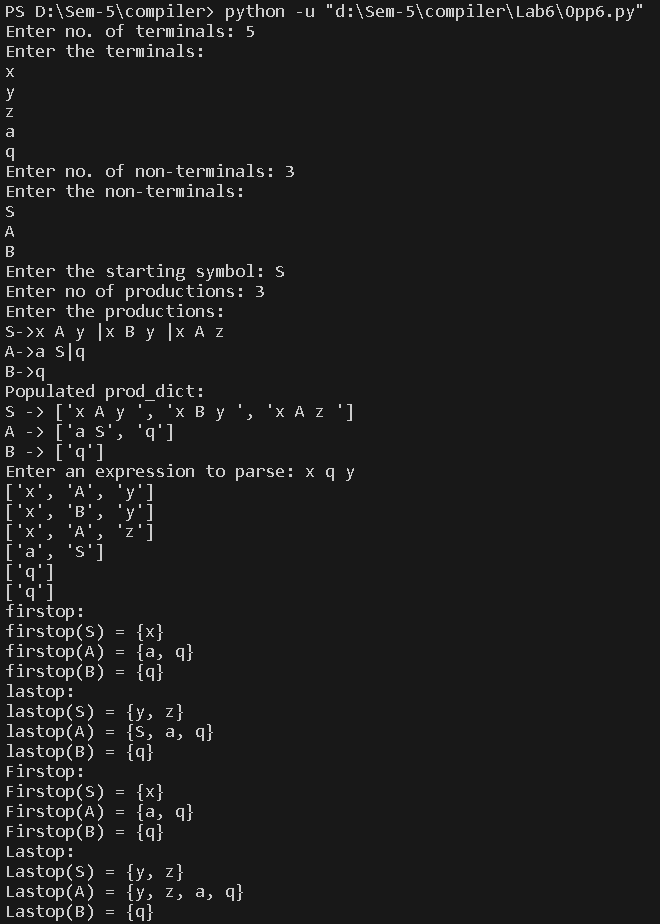
Operator\_Precedence\_table = tabulate(table\_list, headers, tablefmt="grid")

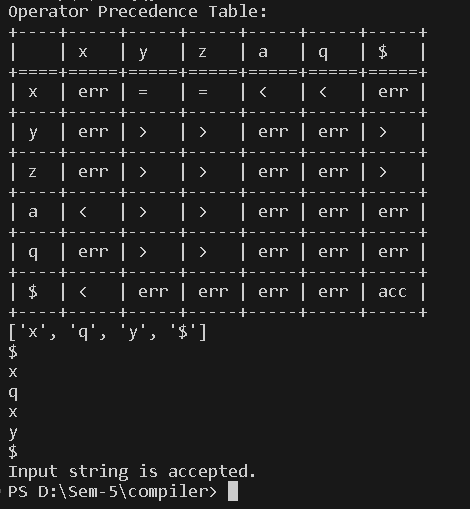
print("Operator Precedence Table:")

print(Operator\_Precedence\_table)

parse\_expression(parsing\_string)

**Output:**

****

****