**Compiler Design Lab**

**Practical-4**

**Name: Yash Patni**

**Roll: 71**

**Batch: A4**

**Aim:** (A) Write a program to validate a string using the parsing table generated in

practical 3. Consider valid &amp; invalid string &amp; print the stack &amp; buffer input tracing.

Input: LL (1) parsing table

Implementation: String parsing rules using stack

Output: Each step-in string parsing and whether the input string is valid or invalid.

**Code:**

from tabulate import tabulate

terminals = ['a', 'b', 'c', 'p']

non\_terminals = ['S', 'A', 'B', 'C']

starting\_symbol = 'S'

productions\_dict = {

    'S': ['A', 'BC'],

    'A': ['a', 'b'],

    'B': ['p', '#'],

    'C': ['c']

}

LL1\_table = {}

conflicts = False

def first(string):

    first\_ = set()

    if string in non\_terminals:

        alternatives = productions\_dict[string]

        for alternative in alternatives:

            first\_2 = first(alternative)

            first\_ |= first\_2

    elif string in terminals:

        first\_ = {string}

    elif string == '' or string == '#':

        first\_ = {'#'}

    else:

        first\_2 = first(string[0])

        if '#' in first\_2:

            i = 1

            while '#' in first\_2:

                first\_ |= (first\_2 - {'#'})

                if string[i:] in terminals:

                    first\_ |= {string[i:]}

                    break

                elif string[i:] == '':

                    first\_ |= {'#'}

                    break

                first\_2 = first(string[i:])

                first\_ |= (first\_2 - {'#'})

                i += 1

        else:

            first\_ |= first\_2

    return first\_

def follow(nT):

    follow\_ = set()

    prods = productions\_dict.items()

    if nT == starting\_symbol:

        follow\_ |= {'$'}

    for nt, rhs in prods:

        for alt in rhs:

            for char in alt:

                if char == nT:

                    following\_str = alt[alt.index(char) + 1:]

                    if following\_str == '':

                        if nt == nT:

                            continue

                        else:

                            follow\_ |= follow(nt)

                    else:

                        follow\_2 = first(following\_str)

                        if '#' in follow\_2:

                            follow\_ |= (follow\_2 - {'#'})

                            follow\_ |= follow(nt)

                        else:

                            follow\_ |= follow\_2

    return follow\_

FIRST = {non\_terminal: set() for non\_terminal in non\_terminals}

FOLLOW = {non\_terminal: set() for non\_terminal in non\_terminals}

for non\_terminal in non\_terminals:

    FIRST[non\_terminal] |= first(non\_terminal)

FOLLOW[starting\_symbol] |= {'$'}

for non\_terminal in non\_terminals:

    FOLLOW[non\_terminal] |= follow(non\_terminal)

for non\_terminal, alternatives in productions\_dict.items():

    for alternative in alternatives:

        first\_set\_alt = first(alternative)

        for terminal in first\_set\_alt - {'#'}:

            if (non\_terminal, terminal) not in LL1\_table:

                LL1\_table[(non\_terminal, terminal)] = alternative

            else:

                print(f"Conflict at ({non\_terminal}, {terminal})")

                conflicts = True

        if '#' in first\_set\_alt or '' in first\_set\_alt:  # If epsilon is in FIRST(α)

            for terminal in FOLLOW[non\_terminal]:

                if (non\_terminal, terminal) not in LL1\_table:

                    LL1\_table[(non\_terminal, terminal)] = alternative

                else:

                    print(f"Conflict at ({non\_terminal}, {terminal})")

                    conflicts = True

if not conflicts:

    print("No conflicts found.")

# Parsing function

def parse\_input(input\_str):

    stack = [starting\_symbol]

    buffer = list(input\_str)

    stack\_trace = []

    buffer\_trace = [buffer[:]]

    action\_trace = []

    accepted = False

    while stack:

        top\_of\_stack = stack[-1]

        if top\_of\_stack in terminals:

            if top\_of\_stack == buffer[0]:

                stack.pop()

                buffer.pop(0)

                action\_trace.append(f"Matched {top\_of\_stack}")

            else:

                break

        elif top\_of\_stack in non\_terminals:

            key = (top\_of\_stack, buffer[0])

            if key in LL1\_table:

                production = LL1\_table[key]

                stack.pop()

                if production != '#':

                    action\_trace.append(f"Applied {top\_of\_stack} -> {production}")

                    stack.extend(reversed(production))

            else:

                break

        else:

            break

        stack\_trace.append(''.join(stack))

        buffer\_trace.append(buffer[:])

        if not stack and not buffer:

            accepted = True

    return accepted, stack\_trace, buffer\_trace, action\_trace

# Validate strings

def validate\_string(input\_str):

    print("Parsing input string:", input\_str)

    accepted, stack\_trace, buffer\_trace, action\_trace = parse\_input(input\_str)

    if accepted:

        print("String Accepted.")

    else:

        print("String Rejected.")

    trace\_data = list(zip(stack\_trace, buffer\_trace, action\_trace))

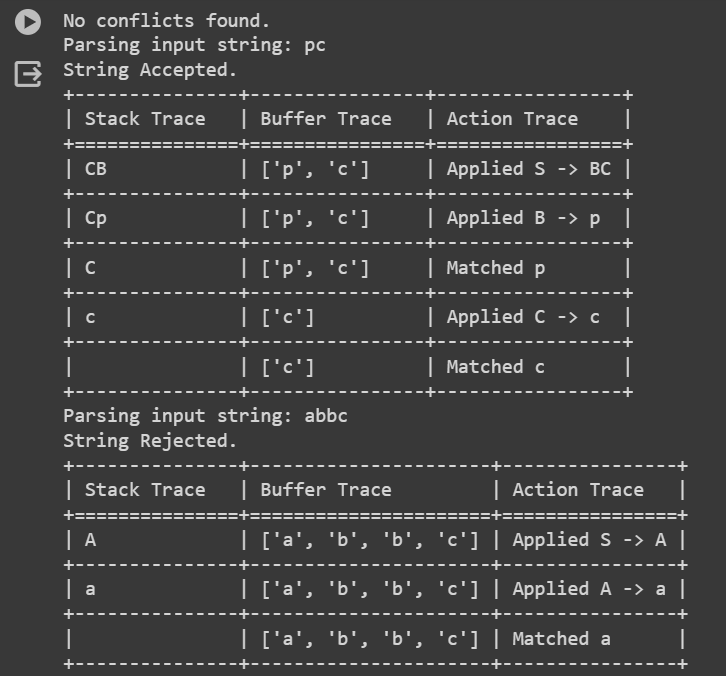
    print(tabulate(trace\_data, headers=['Stack Trace', 'Buffer Trace', 'Action Trace'], tablefmt='grid'))

# Test with some strings

validate\_string("pc")

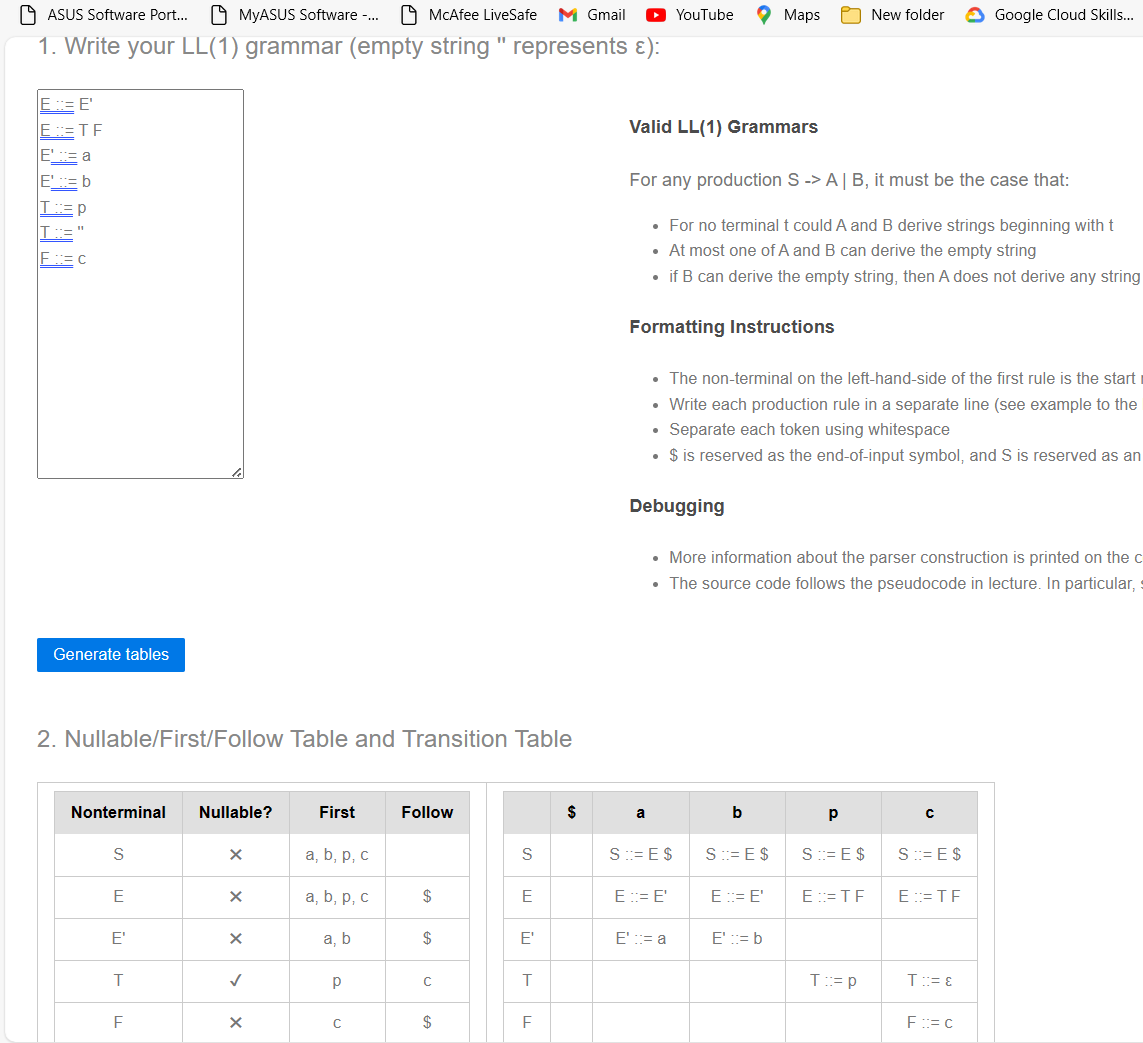
validate\_string("abbc")

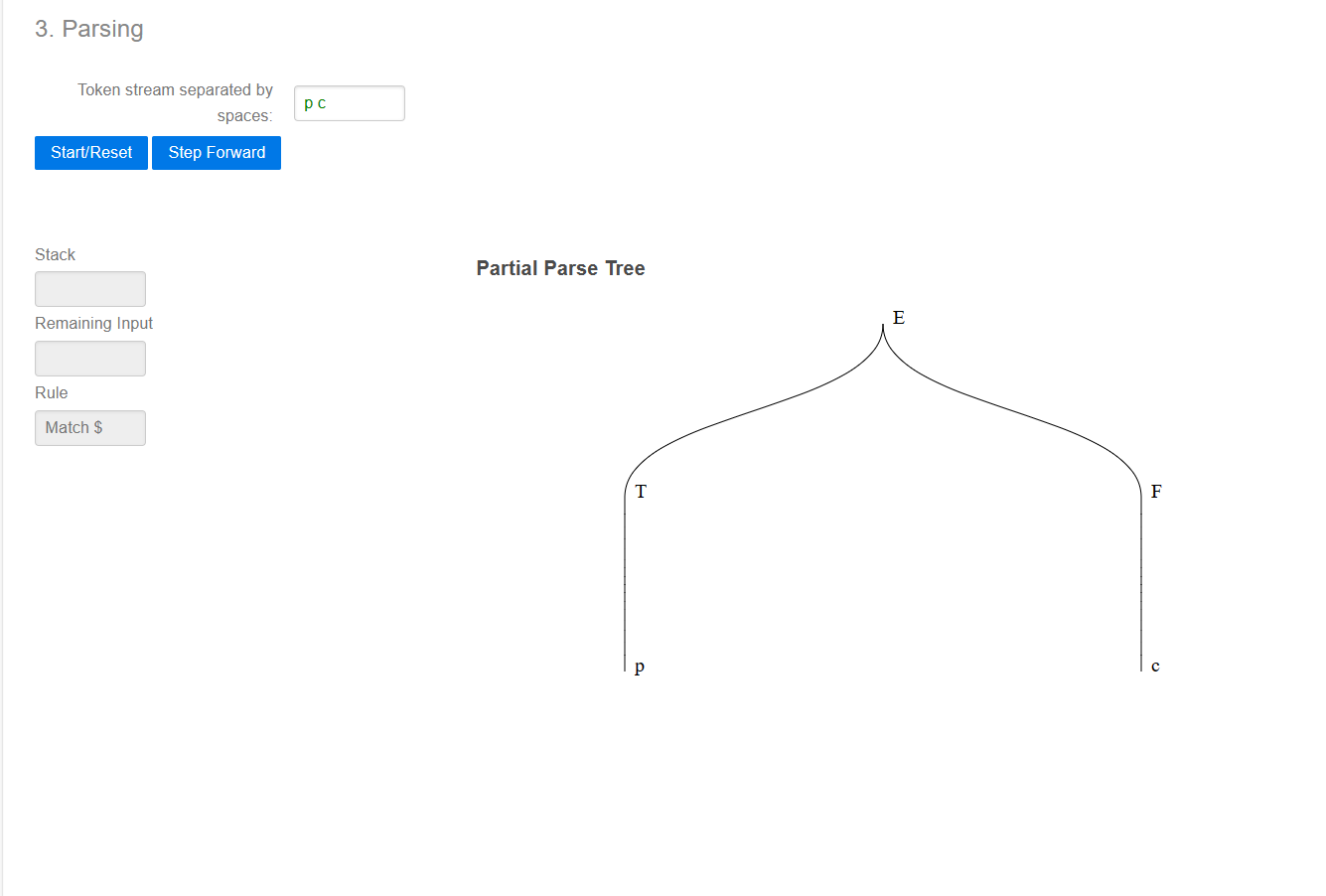
**Output:**

****

**Aim:** (B) Use Virtual Lab on LL1 parser to validate the string and verify your string validation using simulation.

**Output:**





**Conclusion:** In this practical we successfully implemented program to validate string using parsing table. Also used virtual lab simulation to verify and validate the same string.