Name: Varshitha M Gowda
USN: 1NC22IS063

Modern Theory of Computation (22IST62)

AY: 2024-25

CCE 2 - Practical Assignment
Assignment Date: 28/05/2025, Submission Date: 09/06/2025
Submitted To: Ms. Ankita Shukla, Assistant Professor

Name: Varshitha M Gowda
USN: 1NC22IS063

Question 1: Simulating a DFA

Objective: Implement a Deterministic Finite Automaton (DFA) to recognize whether a given binary string is divisible by 5.

Language:

$$L = \{w \in \{0,1\}^* \mid \text{the decimal value of } w \text{ is divisible by } 5\}$$

Approach: We define a DFA with 5 states, each representing the remainder when dividing the number represented by the binary string by 5. The transition table is implemented as a dictionary in Python.

Python Code:

```
def dfa_divisible_by_5(binary_str):
    transition = {
        0: {'0': 0, '1': 1},
        1: {'0': 2, '1': 3},
        2: {'0': 4, '1': 0},
        3: {'0': 1, '1': 2},
        4: {'0': 3, '1': 4}
    }
    current_state = 0
    for char in binary_str.strip():
        if char not in '01':
            return "Rejected"
        current_state = transition[current_state][char]
    return "Accepted" if current_state == 0 else "Rejected"

# Test Cases
print(dfa_divisible_by_5("10000"))  # Rejected
print(dfa_divisible_by_5("1010"))   # Accepted
```

Explanation: This DFA checks the remainder modulo 5 as it reads the input. Final state 0 indicates acceptance.

Name: Varshitha M Gowda
USN: 1NC22IS063

Question 2: Context-Free Grammar Parser

Objective: Parse a string using a CFG that defines the language:

$$L = \{WW^R\} \cup \{W(a+b)W^R\},\ W \in \{a,b\}^*$$

Python Code:

```
def is_cfg_match(string):
    def is_palindrome(s):
        return s == s[::-1]

    n = len(string)
    for i in range(n + 1):
        W = string[:i]
        WR = string[i:]
        if is_palindrome(W + WR):
            return "Accepted"
        if i < n and is_palindrome(W + string[i] + WR):
            return "Accepted"
    return "Rejected"

# Test cases
print(is_cfg_match("aabbbbaa"))  # Accepted
print(is_cfg_match("abababa"))   # Accepted
```

Explanation: This program checks if the string can be split into a palindrome structure matching the CFG by scanning possible splits and verifying symmetry.

Name: Varshitha M Gowda
USN: 1NC22IS063

Question 3: PDA Simulation

Language:

$$L = \{a^n b^m c^m d^n \mid n \geq 1, m \geq 1\}$$

Approach: We simulate stack-based operations to match a with d and b with c.

Python Code:

```
def pda_anbmcmdn(s):
    stack1 = []
    stack2 = []
    i = 0

    while i < len(s) and s[i] == 'a':
        stack1.append('A')
        i += 1
    if not stack1: return "Rejected"

    while i < len(s) and s[i] == 'b':
        stack2.append('B')
        i += 1
    if not stack2: return "Rejected"

    while i < len(s) and s[i] == 'c':
        if not stack2:
            return "Rejected"
        stack2.pop()
        i += 1

    while i < len(s) and s[i] == 'd':
        if not stack1:
            return "Rejected"
        stack1.pop()
        i += 1

    return "Accepted" if not stack1 and not stack2 and i == len(s)
       else "Rejected"

# Test cases
print(pda_anbmcmdn("aabbccdd"))   # Accepted
print(pda_anbmcmdn("aabcdd"))     # Accepted
print(pda_anbmcmdn("aaabbccddd")) # Accepted
```

Explanation: Two stacks simulate matching of a with d and b with c. If both stacks empty and input is consumed, the string is accepted.

Name: Varshitha M Gowda
USN: 1NC22IS063

Question 4: Turing Machine Simulator

Objective: Recognize if a binary string has equal numbers of 0s and 1s.

Approach: We simulate the marking of pairs of 0 and 1 by replacing them to track counts.

Python Code:

```
def turing_equal_0s_1s(tape):
    tape = list(tape.strip())
    while '0' in tape or '1' in tape:
        try:
            zero_index = tape.index('0')
            tape[zero_index] = 'X'
            one_index = tape.index('1')
            tape[one_index] = 'X'
        except ValueError:
            return "Rejected"
    return "Accepted"

# Test cases
print(turing_equal_0s_1s("1010"))  # Accepted
print(turing_equal_0s_1s("110"))   # Rejected
```

Application: Such a Turing Machine can be useful in:

- Compiler Design: Ensuring matching pairs (e.g., parentheses).
- DNA Sequencing: Identifying balanced base pairs.
- Syntax Validation: Ensuring structural balance in programs.