Theory of Computation
Practical Exam

Course Code: CS460 | **Department**: Computer Science
Total Marks: 100 | **Exam Type**: Hands-on Coding

Each student must submit at least two tasks from the set of tasks assigned to their section as below.

Section 1

1. Write a program that construct a DFA that accepts binary strings where the number of 1s is divisible by 3.
2. Write a program that simulates a PDA to check if a string is accepted by a given context-free language (e.g., balanced parentheses, $a^n b^n$).
3. Write a program that simulates a basic single-tape Turing machine that decides the language $L = \{ww \mid w \in \{0,1\}*\}$. Provide a Turing machine definition in code (or file format), and run a simulator that tests acceptance.

Section 2

1. Write a program that implement function that takes a regular expression as input and converts it into a DFA. Then simulate the DFA on a list of input strings.

```
# Sample starter
def regex_to_dfa(regex: str) -> DFA:
    pass  # implement conversion

# Test
assert regex_to_dfa("(a|b)*abb").accepts("aabb") == True
assert regex_to_dfa("(a|b)*abb").accepts("ababa") == False
```

2. Write a program that simulates a PDA to check if a string is accepted by a given context-free language (Palindrome (Odd Length)).
3. Write a program to simulate a Turing Machine that recognizes the language L= { $0^n 1^n 0^n 1^n$}.

Theory of Computation
Practical Exam

Course Code: CS460 | **Department**: Computer Science
Total Marks: 100 | **Exam Type**: Hands-on Coding

Section 3

1. Write a program that minimizes a given DFA using the partition refinement method.
2. Write a program that takes a CFG and converts it to CNF. Output the new set of CNF productions.
3. Write a program to automatically generate a PDA from a given CFG and simulate it on input strings.
4. Write a program to simulate a Turing Machine that increments a binary number by 1.

Section 4

1. Write a Python function that converts a given NFA (with ε-transitions) to an equivalent DFA using the subset construction algorithm.
2. Write a program that takes a CFG and checks if a given string has more than one parse tree, indicating that the grammar is ambiguous for that string.
3. Simulate a Turing Machine that computes the sum of two unary numbers separated by a +.
   Example input: 111+11 → Output: 11111.

Section 5

1. Construct a DFA that accepts all binary strings where the substring `101` appears at least once.
2. Write a program that converts a given Context-Free Grammar (CFG) into Greibach Normal Form (GNF).
3. Write a program to design a Turing Machine that recognizes the language L= { Accept binary numbers divisible by 3}

Theory of Computation
Practical Exam

Course Code: CS460 | **Department**: Computer Science
Total Marks: 100 | **Exam Type**: Hands-on Coding

Section 6

1. Write a function that takes two DFAs and checks if they accept the same language.
2. Write a program that (Cocke–Younger–Kasami) algorithm in Python to determine whether a string belongs to the language generated by a CFG in CNF.
3. Write a program that design a Turing Machine that recognizes the language L= { Accept unary strings where length is prime}

Submission Guidelines

- All code must be your own work. Do not use code generation tools such as ChatGPT, Copilot, or any AI code generators.
- Your submission must be structured in a GitHub repository following the naming convention: `automata_practical_exam_<your_id_number>`
- `README.md` — Must include:
  - Section number
  - Section name
  - Task being solved
  - Brief instructions on how to run and test the code for each section
- Provide full source code with expressive comments in any language.
- Include Unit tests for your program.
- Include `requirements.txt` file listing dependencies (if any).
- Bonus marks will be awarded for:
  - Turning your code into a reusable Python package.
  - Publishing your package publicly (ensure it includes documentation and tests).
  - Completing more than two tasks for extra credit.
- All projects must be pushed to GitHub repository before: `[20-05-2025]`
- Late submissions will not be accepted without a valid reason.

Evaluation Protocol

Criterion	Marks
Correctness of Code	50
Code Organization & Clarity	10
Unit Testing Coverage	15
GitHub Submission	10
Bonus Package & Publication	15
Total	100