# CS208 Practice 2: Stable Matching Algorithm Implementation and Testing

> Deadline: 2025-03-16.
>
> Verification Method: In-person demonstration during lab sessions.

---

## Requirements

### Task 1: Gale-Shapley Algorithm Implementation

Implement the Gale-Shapley stable matching algorithm using **C++**, **Java**, or **Python 3**.

#### Problem Description

Given **N** unmarried boys and **N** unmarried girls, where:

- Each boy has a strict preference ranking of all girls
- Each girl has a strict preference ranking of all boys

Develop an algorithm to find a **stable matching** that satisfies all participants' preferences.

#### Input Format:

1. **First Line**: Integer **N** (1 ≤ N ≤ 1000)
2. **Second Line**: N unique boy names (space-separated)
3. **Third Line**: N unique girl names (space-separated)
4. **Next N Lines**:
    - Boy preferences: Complete preference list for each boy (most preferred to least preferred)
5. **Following N Lines**:
    - Girl preferences: Complete preference list for each girl (most preferred to least preferred)

**Constraints**:

- All names are case-sensitive ASCII strings (1-10 characters)
- No duplicate names within gender groups
- Preference lists contain all members of the opposite gender

#### Output Format:

Your program should output $N$ lines.

Each line contains a boy's name and a girl's name separated by a space, representing a pair in the matching. Please output the boy's name before the girl.

You must make sure that your solution is a stable matching.

**Any valid stable matching** is acceptable if multiple solutions exist.

## Sample

### Sample Input 1

```
3
A B C
a b c
b a c
b a c
b c a
C B A
B C A
A C B
```

### Sample Output 1

```
A a
B b
C c
```

### Sample Input 2

```
5
Victor Wyatt Xavier Yancey Zeus
Amy Bertha Clare Diane Erika
Bertha Amy Diane Erika Clare
Diane Bertha Amy Clare Erika
Bertha Erika Clare Diane Amy
Amy Diane Clare Bertha Erika
Bertha Diane Amy Erika Clare
Zeus Victor Wyatt Yancey Xavier
Xavier Wyatt Yancey Victor Zeus
Wyatt Xavier Yancey Zeus Victor
Victor Zeus Yancey Xavier Wyatt
Yancey Wyatt Zeus Xavier Victor
```

### Sample Output 2

```
Victor Amy
Wyatt Clare
Xavier Bertha
Yancey Erika
Zeus Diane
```

## Implementation Notes

1. **Stability Verification**: Your solution must guarantee output stability.
2. **Efficiency Constraints**:
    - C++: ≤1 second for N=1000.
    - Java: ≤2 seconds.
    - Python: ≤3 seconds.
3. **Input Parsing**: Handle leading/trailing whitespace in input files.

## Task 2: Construct Test Cases

Design **at least 10 test cases** covering the following categories:

| Category | Purpose |
|---|---|
| **Basic Functionality** | Validate correctness for simple, non-conflicting preferences. |
| **Edge Cases** | Test minimum input size (( N=1 )), extreme preferences, and edge values. |
| **Multiple Stable Solutions** | Scenarios with multiple valid stable matchings. |
| **Conflict Scenarios** | Cyclic preferences (e.g., ( B1→G1→B2→G2→B1 )). |
| **Performance & Scalability** | Ensure efficiency for $N \geq 1000$. |
| **Random-Scale & Distribution Tests** | Validate robustness with programmatically generated inputs of varying sizes. |

## Task 3: Test Your Algorithm

### 1. Manual Verification

- **Objective**: Confirm that the output matching satisfies the **Gale-Shapley stability criteria**.
- **Steps**:
  1. Run your algorithm on small test cases (e.g., $N \leq 3$).
  2. Check whether there are any **unstable pairs**.
  3. Verify alignment with theoretical properties (e.g., proposer-optimality).

### 2. Diff-Based Testing

- **Objective**: Compare your implementation's output against a reference solution to detect discrepancies.
- **Procedure**:
  1. **Brute-Force Program**:
     - Generate **all possible stable matchings** using exhaustive search.
  2. **Comparison**:
     - Run both your **Gale-Shapley-based program** and the **Brute-Force Program** on the same input.
     - Use tools like `diff` or automated file comparison to validate if the output of the Gale-Shapley-based program and any output of the reference program.
     - **Reference**: See *Tutorial 2: Testing Algorithm Correctness.pdf*.
- **Think about and attempt to solve the following Questions:**
  - **Q**: *Does diff-based testing remain feasible for a little larger inputs (e.g., ( N > 10 or N > 20 ))?*
  - **Q**: *How to test when ( N ) exceeds a threshold (e.g., ( N > 10 or N > 20))?*

### 3. Performance Testing

- **Objective**: Ensure the algorithm meets **time complexity requirements** (e.g., $O(N^2)$) and scales efficiently.
- **Steps**:
    1. **Baseline Benchmark**:
        - Execute your program for ( N = 1000 ) and ensure completion within **1 second** .
    2. **Scalability Analysis**:
        - Measure execution time for ( N = 100, 500, 1000, 2000 ) to observe growth trends.
        - Plot time vs. $N^2$ to confirm $O(N^2)$ complexity.
    3. **Optimization**:
        - If performance lags, profile the code to identify bottlenecks (e.g., nested loops, redundant operations).
        - Optimize critical sections (e.g., replace linear searches with hash maps for preference lookups).

## Task 4: Fairness Discussion

Does the result differ if **boys propose first vs. girls propose** first? Explain why?

---

# Submission

1. **Code**: Implemented Gale-Shapley algorithm (C++/Java/Python).
2. **Test Suite**:

- Input files, expected outputs, and test scripts.
- Explanation of test case categories and purposes.
- Brute-force program

3. **Report** explaining:
    - Correctness and performance validation methodology.
    - Execution time measurements for ( N = 100, 500, 1000, 2000 ).
    - Answers to the fairness discussion.