CptS 223 – Advanced Data Structures in C++

CptS 223 – Advanced Data Structures in C++ Programming Assignment 2: Josephus Problem Variant

Assigned: Wednesday, February 5, 2025

Learner Objectives

By completing this assignment, students will be able to:

- Develop a C++ program using STL `list` and `vector` sequence containers.
- Open, edit, parse, and close `.csv` files in C++.
- Evaluate the performance of the STL `list` and `vector` operations.
- Analyze performance differences between similar data structures on medium-sized datasets.

Prerequisites

Students should be able to:

- Design, implement, and test medium-scale programs in C++.
- Build and execute programs in a Linux environment.
- Understand STL `vector` and `list` operations.

Overview & Problem Statement

This assignment implements a variant of the **Josephus problem** (Wikipedia).

The problem scenario is:

- A group of friends is planning their spring break trip and must choose one of **'N' proposed cities**.
- They decide to use the Josephus problem to select the destination fairly.
- **N** represents the number of destinations under consideration.
- A random number **M**, where **M < N**, determines how destinations are eliminated.
- Starting at destination `0`, move in numerical order, eliminating every `M`-th destination.
- Repeat until only one destination remains, which is selected for travel.

Students should refer to the **attached PPT example** of the Josephus problem for a detailed explanation of the elimination process.

Your program must implement **two versions** of this problem using:

1. `std::list`
2. `std::vector`

Implementation Requirements

Class & File Design (45 pts)

1. Destination Class (7 pts)

Create a `Destination` class with:

- **Files:** `Destination.h` and `Destination.cpp`
- **Public Methods:**
  - Constructor to initialize `position` and `name`.
  - Destructor.
  - `printPosition()` - Displays the destination's position.
  - `printDestinationName()` - Displays the destination's name.
- **Private Data Members:**
  - `int position` – Position in the sequence (or ID).
  - `std::string name` – City name.

2. Josephus Classes (38 pts)

Implement two separate classes:

1. **`ListMyJosephus`** (using `std::list`)
2. **`VectorMyJosephus`** (using `std::vector`)

Each class should be in separate `.h` and `.cpp` files and must include:

- **Public Methods:**
  - Constructor (`M`, `N` as parameters).
  - Destructor.
  - `clear()` – Empties the sequence.
  - `currentSize()` – Returns remaining destinations.
  - `isEmpty()` – Checks if the sequence is empty.
  - `eliminateDestination()` – Eliminates a destination based on the rules.
  - `printAllDestinations()` – Prints remaining destinations sorted by position.
- **Private Data Members:**
  - `int M` – Elimination interval.
  - `int N` – Total initial destinations.
  - `std::list` (for `ListMyJosephus`) or `std::vector` (for `VectorMyJosephus`).

Testing & Reporting (35 pts)

Test Implementations

Create two separate test files:

- `TestListMyJosephus.cpp`
- `TestVectorMyJosephus.cpp`

Note: These files **do not** involve any automated testing. They should just contain code where students can run a simulation and print the output.

Each test should:

1. **(4 pts)** Use a `for` loop to iterate from **N = 1 to 1025**. For each iteration, do the steps 2 to 5
2. **(10 pts)** Randomly pick one of the **25 rows** from `destinations.csv` (each row has ~16,000 columns, aka cities/destinations) and populate the container with the first `N` destinations.
3. **(5 pts)** Randomly pick M (where M < N) and Run a full simulation until only one destination remains.
4. **(5 pts)** Output the elimination sequence after each round.
5. **(2 pts)** Print the final selected destination.
6. **(7 pts)** Improve the test files by:
    - Capturing timestamps before and after each simulation, to compute time taken.
    - Store the time durations in an array or a container of your choice.
    - Compute and print the **average simulation time** for both `TestListMyJosephus` and `TestVectorMyJosephus`.
    - Write the recorded times (from the container) into `results.log`.

Timing Notes

- Print statements **can be included**, but students must ensure an equal number of print statements in both `ListMyJosephus` and `VectorMyJosephus` implementations.
- For very small values of `N`, timing computations may show `00000`, which is expected.

Makefile Requirement (5 pts)

Create a `Makefile` with flags: `-g -Wall -std=c++14`.

Discussion Questions (10 pts)

Students must attach a `README` file answering the following questions, with responses connected to this specific problem:

1. **(2 pts)** Does machine processing power affect execution time?
2. **(5 pts)** Which performs better: `std::list` or `std::vector`? Under what conditions?
3. **(3 pts)** How does `N` impact runtime compared to `M`?

Submission Instructions

Students should submit their work directly to **Canvas** by:

- Zipping their entire project folder and submitting it.
- Attaching a `README` file with answers to discussion questions.
- Including **screenshots** to demonstrate that their code works.

Grading (100 pts total)

- **95 pts** – Correct implementation following requirements.
- **5 pts** – Code style, clarity, and design principles.
- **Max 60 pts** if the code does not compile.

---End of Assignment---