SIEMENS ASSIGNMENT 2 REPORT

✨ *AMIR KEDIS* ✨

Introduction

This assignment is focused on testing a packet reception scenario from multiple modules. The task is to design how to test this feature and implement a C++ program to automate the verification of all possible scenarios and their expected results.

In this report, I will explain the approach, assumptions, implementation, input and output formats, test cases, validation, and results of my solution.

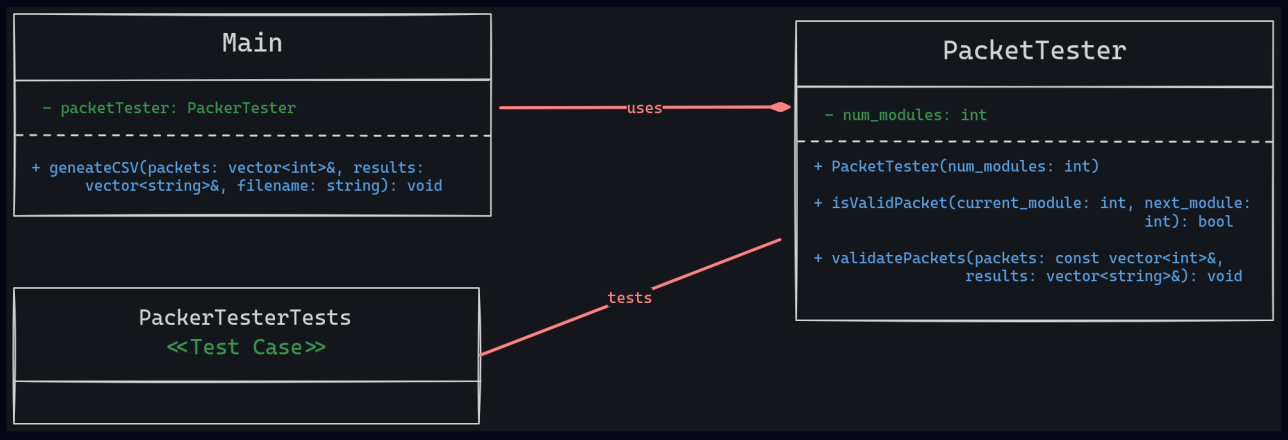
Approach / System Design

- 👍 should handle:
  - 👍 various sequences of packets
  - 👍 edge cases such as:
    - ⚠️ only one packet
    - ⚠️ negative / invalid module number
    - ⚠️ out of range
    - ⚠️ etc

Implementation

For the system design the system consists of 3 main components:

- PacketTester: the class that handles the validation of a certain sequence.
- Main Driver program: Runs the test cases and writes the output to a CVS file.
- Unit tests: validates the PacketTester class with different test cases.



File structure

```
.
├── CMakeLists.txt
├── docs
│   ├── report.md
│   └── sample_output.csv
├── output.csv
├── README.md
├── src
│   ├── main.cpp
│   ├── packet_tester.cpp
│   └── packet_tester.hpp
└── tests
    └── test_packettester.cpp

4 directories, 9 files
```

Input and Output Formats

- input format
  - number of packets: int
  - module number of each packet: vector<int>

```
4
```

```
1 3 4 4
```

- output format
  - the output will be a CSV file with the following columns:
    - PacketId
    - ModuleNumber
    - ValidModule

```
PacketID,MouduleNumber,ValidModule
1,1,Yes
2,3,No
3,4,Yes
4,4,Yes
```

Assumptions

❓ Regarding M (the Maximum Module Number):

- I assumed that the maximum module number is a constant given to class PacketTester for testing.
- other valid assumption that I could have done here is always taking the largest number in the array as the maximum module number. by this approach is not as realistic as the first one.

❓ Regarding the input:

⚠️ Packet Modules are always positive integers.

⚠️ the sequence of packets must be non-empty.

⚠️ the system must handle positive number of M (maximum module number).

Test cases

Unit Tests for PacketTester:

1. Sample Test Case:

❓ Input: {1, 3, 4, 4}

⚠️ Expected Output: {"Yes", "No", "Yes", "Yes"}

2. Edge Case Around Number of Modules:

❓ Input: {1, 3, 4, 4, 1, 2}

⚠️ Expected Output: {"Yes", "No", "Yes", "Yes", "Yes", "Yes"}

3. Only One Packet:

❓ Input: {1}

⚠ Expected Output: {"Yes"}

4. Same Value:

❓ Input: {1, 1, 1, 1, 1}
⚠ Expected Output: {"Yes", "Yes", "Yes", "Yes", "Yes"}

5. Monotonic Increase:

❓ Input: {1, 2, 3, 4, 5}
⚠ Expected Output: {"Yes", "Yes", "Yes", "Yes", "Yes"}

6. Monotonic Increase Exceeding Max Module Number:

❓ Input: {1, 2, 3, 4, 5}
⚠ Expected Output: {"Yes", "Yes", "Yes", "Yes", "No"}

7. Monotonic Decrease:

❓ Input: {5, 4, 3, 2, 1}
⚠ Expected Output: {"Yes", "No", "No", "No", "No"}

8. Negative Numbers:

❓ Input: {-5, -4, -3, -2, -1}
⚠ Expected: Throw exception

9. Mixed Positive and Negative:

❓ Input: {-5, 4, 3, -2, -1}
⚠ Expected: Throw exception

10. All Zeros:

❓ Input: {0, 0, 0, 0, 0}
⚠ Expected: Throw exception

11. No Packets:

❓ Input: {}
⚠ Expected: Throw exception