# Programming Exercise – Event Pattern Matching

NOTE THAT THIS IS A MADE UP PROBLEM, AND IS NOT NECESSARILY REPRESENTATIVE OF REALITY

## Objective

Implement a processor that looks for and counts specific patterns in an event log.

Assume that calls to the ParseEvents function can be called from multiple threads concurrently with the same device ID. Take steps to ensure proper operation under heavy multi-threaded use.

## Problem Overview

A design flaw has been found in a type of HVAC unit. The unit can be in one of four stages (0-3) depending on environmental conditions. The manufacturer has noted a design flaw where the unit can accidentally go through a specific sequence of stages leading to possible damage.

The manufacturer has noted that a fault is indicated by four operations that occur in sequence:

1. Stage 3 for five minutes or more
2. Stage 2,
3. Any number of cycles between stage 2 and 3 for any duration
4. Stage 0

Your task is to implement a log parser that detects and counts occurrences of this fault sequence.

Your parser should derive from the given IEventCounter interface and supply an implementation for two methods:

1. `void ParseEvents(string deviceID, StreamReader eventLog)` – Inspect and parse a stream of operation records associated with the given device ID, and count occurrences of the "fault" sequence.
2. `int GetEventCount(string deviceId)` – Gets the total number of "fault" sequences observed for the given device.

## Log Format

Logs are in a tab separated file with each row as a Time/Value pair. Note that an entry is recorded each time the unit changes stage, however extra recordings may be taken (for a variety of reasons) that do not indicate a change of stage, and are simply redundant pieces of information.

For Example,

```
Timestamp               Value
2011-03-07 06:25:32     2
2011-03-07 09:15:55     3       ← Indicates a stage change as previous recorded value is '2'
2011-03-07 12:00:00     3       ← Redundant recording, possibly a "heartbeat"
2011-03-07 12:03:27     2       ← Indicates a stage change
```

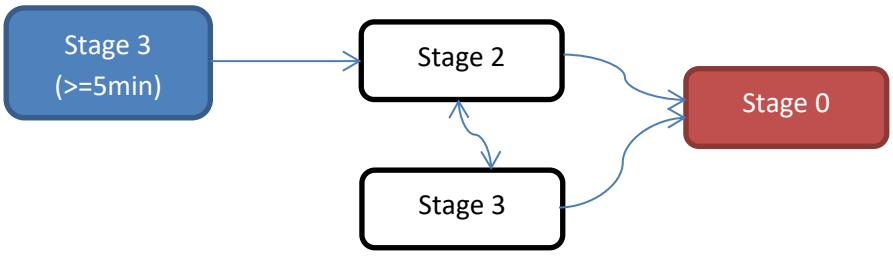
```
2011-03-07 20:23:01      0
```

Logs may contain any number of entries. The end of a log is indicated by exhaustion of the Reader (i.e., no more data is available).

### Fault Event Pattern

Your implementation should count occurrences of a specific sequence of operation. A fault occurs when a unit is in stage 3 for five minutes or more, followed by a direct transition to stage 2, followed by any number of cycles between stage 3 and 2, followed by a transition to stage 0 from either stage 2 or 3.

The following diagram illustrates the sequence of operations that lead to a fault:




Note: all log file entries can be assumed to be in chronological order.

## General Instructions

If you are unable to provide a complete solution, you should make every effort to provide a working submission – your code should compile without error and should supply at least some of the expected behavior. We will evaluate your submission by applying a series of unit tests to verify that your implementation behaves as outlined in this specification.

In addition, we strongly encourage you to document your thoughts in comments, or a supplementary text file and include these notes as part of your submission. This will allow us to evaluate the clarity of your thinking about the problem and solution approach.

We will also be evaluating the quality of the code you write and looking at the choices you make regarding naming, formatting, organization, decomposition, data structures, etc.

## Coding Instructions

The interface for constructing your solution in C# is provided, however you are welcome to code your solution in any main-stream, Object-Oriented programming language. Regardless of what language you choose, your submission must:

1. Implement the IEventCounter concept by supplying the two methods described above; and
2. Include testing to demonstrate your solution calculates the correct results under a variety of input conditions.

## What to Submit

1. Your implementation of the IEventCounter interface
2. Any code you devised to test your implementation

3. Any artifacts you feel are relevant (e.g., build instructions, notes on your approach, etc.)

## Bonus Challenge 1: Dynamic Web Page

A simple Web API project has been provided in the "WebBonus" archive file. It is an ASP.NET project that can be run using the Express edition of Visual Studios 2017. Launch the service in Visual Studio using IIS Express, and you should see a simple endpoint hosted on port 50196.

The API is as follows:

GET api/events gets a list of all devices that have been processed or are being processed.

GET api/events/{id} gets just the number of detected events for the given device ID.

POST api/events/{id} simulates parsing log data for the given event.

Your task is to create a single web page that will use the given API to:

1. Display a simple list of processed device IDs and their associated "event count"
2. Add new or update existing entries in the list

The page should update dynamically as the count for each device slowly increases. You can assume this is a single user environment; don't worry about multiple users adding/updating devices.

Use any javascript framework you wish to complete this portion of the challenge.