C/C++ Test: Set #3

Develop an application that does the following:

Maintains a queue that can hold following types of **Event** data:

```
struct EventRequest {

        char EventType; /* ‘R’ for EventRequest */

        int RetryCount;         /* E.g. 0, 1, 2, 3, etc. */

};

struct EventStatus {

        char EventType; /* ‘S’ for EventStatus */

        char StatusType;        /* Can be only one of : ‘P’, ‘M’,  ‘C’ or ‘T’ */

        short RetryCount;       /* E.g. 0, 1, 2, 3, etc. */

};
```

In order to demonstrate the working of this application, prepare the following methods/functions:

1. **Push()**: Adds to one end of the queue - any of the above Event type packets that is passed as input.

2. **Pop()**: Removes an element from other end of the queue and returns the same. Returns appropriate value when queue is empty or on error.

NOTE: *The exact signature of the above functions has not been provided since there are multiple alternatives to designing this, and hence we are flexible w.r.t. parameters and return types.*

Given the above specifications, the application is expected to execute as follows:

1. Create/initialize the required queue.

2. Create and **Push()** each of event packets (given below) to the queue in the same sequence as listed.

3. **Pop()** each element from the queue and check event type and process in the following way:

- If it's an EventStatus: store its StatusType value and output its contents (in the format specified). If the StatusType is either 'C' or 'T' and its RetryCount value is < 2, **Push()** this packet back to the queue after incrementing it's RetryCount by 1.
- If it's an EventRequest: check if the last received StatusType is either 'C' or 'T'. If so, output its contents (in the format specified). Else **Push()** this packet back to the queue after incrementing it's RetryCount by 1.

4. Continue to call **Pop()** and process all the events (as given above) - until the queue is empty. Then exit the program.

Guidelines

1. Make suitable assumptions and decisions regarding data types, data structures and their relationships.

2. Error & boundary conditions should be appropriately handled

3. Application output should clearly demonstrate the required functionality

4. Application code should be optimized for least memory usage and least processing time - during execution.

The list of Event packets to be pushed to the queue is given below:

'S', 'P', 0	<EventStatus>
'R', 0	<EventRequest>
'S', 'M', 0	<EventStatus>
'S', 'P', 0	<EventStatus>
'S', 'T', 0	<EventStatus>
'S', 'P', 0	<EventStatus>
'S', 'C', 0	<EventStatus>
'S', 'M', 0	<EventStatus>

The format of displaying popped Event packet is given below:

EventStatus: <EventType>, <StatusType>, <RetryCount>

EventRequest: <EventType>, <RetryCount>

Output examples (for reference only) are given below:

EventStatus: S, T, 8

EventStatus: S, M, 0

EventRequest: R, 19