Programming Assignment 3
Implementing Atomic MRMW Register from M-Valued Regular Registers

Submission Date: ~~27th March 2022~~ 3rd March 2022, 9:00 pm

Goal: The goal of this assignment is to implement a m-valued MRMW Atomic registers from a Regular m-valued SRSW register as studied in the class. Implement this algorithm in C++ and compare its performance with C++ atomic variables implementation.

Details. Assume that the by default the behaviour of C++ variables are **regular**. Given this, you have to implement MRMW Atomic registers using the algorithm given in the book **in C++**. You can assume that the system accommodates a total of N threads which is known to you.

You have to test the performance of your implementation with the inbuilt C++ atomic variable implementation (https://en.cppreference.com/w/cpp/atomic). To test the performance of your implementation, develop an application, atomic-test is as follows. Once, the program starts, it creates n threads. Each of these threads, will read/write from/to the atomic register k times. The pseudocode of the test function is as follows:

Listing 1: main thread

```
1  void main()
2  {
3      ...
4      create shVar = 0;     // Atomic shared variable initialized to 0
5      ...
6      create N testAtomic threads;
7      ...
8  }
```

Listing 2: testAtomic thread

```
1  void testAtomic()
2  {
3      int lVar;     // local variable
4      int id = thread.getID();
5      for (i=0; i < k; i++)
6      {
7          action = randomly decide to either read with probability p or
8          write with probability (1-p);
9          reqTime = getSysTime();
10         cout << i << "th action requested at " << reqTime << " by thread "
11         << id;
```

```
12
13          if (action == read)
14          {
15              lVar = shVar.read();       // replace the syntax for C++ atomics accordingly
16              cout << "Value read: " << lVar;
17          }
18          else      // Write action
19          {
20              lVar = k * id;   // the value written by each thread is unique
21              shVar.write(lVar);
22              cout << "Value writte: " << lVar;
23          }
24          complTime = getSysTime();
25          cout << i << "th action " << action "completed at " << actEnterTime <<
26          " by thread " << id;
27          sleep(t1);   // Simulate performing some other operations.
28      }
29  }
```

Here $t1$ is a delay value exponentially distributed with an average of λ seconds. The objective of having these time delays is to simulate that these threads are performing some complicated time consuming tasks.

Input: The input to the program will be a file, named inp-params.txt, consisting of all the parameters described above:N, k, λ, p. A sample input file is: 20 100 5 0.8.

Output: Your program should output to a file in the format given in the testAtomic function which should demonstrate that atomicity behaviour of the implementation.

Report: You have to submit a report for this assignment. This report should contain a comparison of the performance of your implementation and inbuilt C++ atomic variable implementation. You must run both these algorithms multiple times to compare the performances and display the result in form of a graph.

You run both these algorithms varying the number of threads from 10 to 50 while keeping other parameters same. Please have k, the number of actions requests by each thread, fixed to 100 in all these experiments. You measure the average time taken by each thread to perform read or write operation.

Specifically you will have a graph with the following curves in y-axis comparing your algorithm with the C++ atomic library implementation:

1. Average time taken by the write operations.
2. Average time taken by the read operations.
3. Average time taken by both read and write operations.

To rule out any outliers, please obtain the value of each point after averaging it over 5 times. Please give an analysis of the results in the report while explaining any anomalies observed.

Deliverables: You have to submit the following:

- Source file: The source file containing the actual program to execute. Name it as SrcAssgn3-⟨rollno⟩.cpp.
- Readme: A readme.txt that explains how to execute the program. Please name the files as: Assgn3-Readme-⟨RollNo⟩.txt.

- Report: The report as explained above. Please name it as Assgn3-Report-⟨RollNo⟩.txt

Zip all the three files and name it as ProgAssgn3-⟨rollno⟩.zip. Then upload it on the google classroom page of this course. Submit it by the deadline mentioned above.
We have the following grading policy:

1. Design as described in the report and analysis of the results: 50%
2. Execution of the programs based on the description in the readme: 40%
3. Code documentation and indentation: 10%

As mentioned before, all assignments for this course have the late submission policy of a penalty of 10% each day after the deadline. We will consider a late assignment for a maximum of 6 days. Any submission beyond that will not be considered. **Kindly remember that all submissions are subjected to plagiarism checks.**