**Program Design:-**

* There are two programs, **First** program is using an inbuilt C++ atomic variable. The atomic variable is defined under <atomic> header.
* I have used the atomic integer variable, shVar and initialized it to 0.
* The load ( ) function of <atomic> header, is used to read the current value of the atomic variable shVar into locVar.
* The store( ) function of <atomic> header, is used to write the value of locVar into shVar.
* The memory\_order\_relaxed parameter under load( ) and store ( ) function made sure that operations are happening atomically at some point.
* Rest of the program is as per the code provided in the assignment.
* In the **Second** program, I have implemented an atomic Multi Reader Multi Writer Register. The base registers that I have used (assumed) for the construction are of type atomic Multi Reader Single Writer type. I have used an array (the size of the array will be the number of threads used on the program) of such registers.
* As it is mentioned in the assignment, I have assumed the default behavior of C++ variables is regular. So, the construction of the MRMW register is as follows:- First, I have shown how to implement SRSW Atomic from SRSW regular, then I have shown how to construct a Multi-Reader Atomic Registers from an array of Single Reader Atomic Registers. Then, at last, I have shown how to construct a Multi Writer Atomic Register.
* SRSW Regular 🡪 SRSW Atomic 🡪 MRSW Atomic 🡪 MRMW Atomic

**Algorithm for Implementing Atomic MRMW Register**

* All the construction, that I have shown has been the same construction that is shown in the book.
* Each read remembers the latest (highest timestamp) timestamp/value pair ever read so that it is available to future reads. If a later read then reads an earlier value (one having a lower timestamp), it ignores that value and simply uses the remembered latest value. Similarly, the writer remembers the latest timestamp it wrote, and tags each newly written value with a later timestamp (a timestamp greater by 1).
* So, This algorithm requires the ability to read–write a value and a timestamp as a single unit. So, for this purpose, I have created a class named StampedValue, where data variable stamp is used for storing timestamps, and value is used for storing value in the register.
* MAXStampedValue function is used for giving the maximum StampedValue object between the two StampedValue objects provided as parameters.
* I have created a Base pure Abstract Register class, having function read and write.
* I have first constructed SRSW Atomic Class. I have followed the same pseudocode, that has been given in the book. I have used the StampValued class object in order to get the atomic class. I have commented on the main function of it. You can uncomment it check, whether it is working or not.
* In the book, they have used StampValued class objects in different – different ways in order to obtain various classes of registers implementation. I have also followed the same process.
* For the second construction, which is of MRSW atomic register from an array of SRSW atomic registers, I have again used the pseudo-code given in the book. Here I have used SRSW atomic register class objects which I constructed above. I have created a two-dimensional array of atomic SRSW registers. I follow the same method that is described in the book. I have again also commented on the main function of it. You can uncomment it check, whether it is working or not.
* At last, I have created an MRMW atomic register from an array of atomic MRSW registers. When will create an object of MRMW class, its constructor will create an array of atomic MRSW registers which will be initialized with value 17. (As I have passed it in test atomic function). Now when the action is written, then we are going through the entire array, and select the register which contains a maximum timestamp, and then we are updating the register corresponding to that thread’s timestamp by 1 and we are also updating the register value with the value provided. So in this way, multiple writers can write on the register. For the read operation, each reader scans the whole register array and then returns the register value corresponding to the register to have the highest timestamps.

**Performance Comparision:-**

For all the graphs below, the Number of threads has been varied from 10 to 50, whereas other parameters k(number of iteration) = 100, probability p = 0.8, and lambda = 5 are fixed for all the experiments.

READ OPERATION:-

WRITE OPERATION:-

Average Read & Write:-

For the readability purpose of output that will get generated after execution, I have used mutex lock a few times. That’s why the overall average read & write time has been increased, but since I have used lock in both the code, So this increase is relative.