This assignment is to challenge you on semaphores, mutexes, and random number generation from specific distributions.

**Storyline:**

A group of students, working their term project, reach the phase where they have to do the much dreaded “literature search.” For that purpose, they decide to meet an early morning at the library and work in this fashion:

1. **Book-hunting step:** Each student sets forth and gets a random book from the shelves (assume that all books on the shelves are relevant). No two students can take the same book; it’s just not efficient 😞.
2. **Reading step:** Each student separately reads their chosen book.
3. **Sharing step:** Once all students finish reading their respective books, they gather at the conference room and share their impressions and new knowledge. Sharing of impressions cannot happen unless all students are present in the room, and no student leaves the session until everyone is done sharing.
4. **Repetition step:** Once the sharing of impressions is over, the students resume the book hunting by dropping their books in the return pile and picking a new book starting again at step 1, repeating until they go through all the books on the shelves. No student should join a sharing session empty-handed unless the shelves are empty.

**Specifics:**

- There are \( N \) students in the group and initially \( M \) books on the shelves (\( 1 < N < M \)).
- The basic time unit is \( \mu s \) (microseconds).
- Each book \( b_i \) has a random number of pages \( P[b_i] \) taken over a normal distribution with a global per-book average of 400 pages. Each book can’t grow or shrink in page count (once the random number \( P[b_i] \) is set, it sticks with that book).
- The actual reading time of the book takes a random time \( X \) taken over an exponential distribution with mean value of \( TXP[b_i] \) time. The actual reading time varies every time a book is read and should be decided at the beginning of the reading of that book.
- The sharing session starts only when every student is done reading. The duration of the sharing session is a random time \( D \) over an exponential distribution with the average proportional to the summed number of pages of the books shared in that session, i.e. \( TX\Sigma_{i=\text{shared}} P[b_i] \).

The math notation above is used only to convey the concepts, and are not required to be copied as-is in the programs. Use your own notation and conventions as long as they mathematically agree with the above.

**Your assignment:**

Now, your assignment is to develop a multi-threaded program that correctly simulates the above situation for arbitrary values of \( N, M, T \). Each student is represented by a POSIX thread whose code should look like:

```c
student_thread()
{
    booktype book;
    booleantype done;
    meet_at_the_library();
    do {
        book = pick_a_book();
        read_book(book);
        done = share_impressions(book);
    } until (done);
    exit_thread();
}
```
Beyond the basic requirement of compile-ability and executability [20pts] (compiles, runs, does not crash, does not deadlock for any legal values of N, M, T), there are four requirements:

- [0pts] 1st requirement: do not change the student_thread code. You are not allowed to alter the code of student_thread, in the sense that you cannot change the arguments and you cannot introduce more local variables than the existing book and done local variables. You can change the names of the datatypes, of the functions and of the variables (you should have a reason though), and you can also convert the do-until loop to a do-while or even to a for loop; and that’s pretty much it. You may end up using int instead of boolean type (and perhaps instead of booktype). This requirement has no points but is an absolute requirement for your solution to receive a passing grade.

- [40pts] 2nd requirement: the functions should reflect the described behavior. Particularly, meet_at_the_library() should not return until all threads are inside the meet_at_the_library code; pick_a_book() should return a book that is different from other picked books in the iteration and different from all the books picked in the past iterations; read_book() should return when T times T[P[b]] time has passed; and finally, share_impressions() should return when T times T[T[P[b]]] has passed after the last student thread called share_impressions(). Also, share_impressions() should return a “true” on done when there are no books left.

As you probably understand by now, you are asked to develop the entire program around the given student_thread code. It is up to you to design and define the data types, declare the global variables, initialize them, decide what function libraries to use, and initialize and kick-start the threads; and, of course, to implement the functions meet_at_the_library, pick_a_book, read_book and share_impressions. You cannot use any kind of synchronization other than POSIX semaphores and POSIX mutexes (pthread_mutex) in a combination of your choice. Use of the C language is strongly preferred.

- [10pts] 3rd requirement: no busy-waiting. Your program should exhibit minimal CPU utilization. Expectedly, threads should mostly be sleeping while waiting on either semaphores/mutexes or a time delay. This requirement also implies that keeping track what books are read and picking a book are compute-efficient procedures.

- [10pts] 4th requirement: introduce debug print messages. Each of the meet_at_the_library(), pick_a_book(), read_book() and share_impressions() functions should have at least two debug print messages, one at the entry and one at the exit point. The debug messages should be prefixed with a proper timestamp, followed by the identification of the student thread and then the actual message, e.g.:

```
13:04:02.000000 Thread 01 : Entering share_impressions()
13:04:04.000000 Thread 01 : Exiting share_impressions(). Time taken: 00:00:02.000000
```

You may want to insert additional debug messages throughout the code depending on the workings and internal structure of the functions as you implement them. The debug print messages should print exclusively.

**Suggestion 1**: as you see in the specifics description, the times are given in multiples of T. Use large values of T to debug your program, and small values to run long runs for large values of M.

**Suggestion 2**: use a global variable or a define VERBOSE_MODE to enable or disable secondary debug messages.

**What/how to submit:**

Submit a report on Blackboard that explains your design and implementation choices [30 points], describes how the work was split among team members, followed by an appendix of your well-commented source code [80 points]. Teams of up to two students may submit a homework report. Larger teams will receive no grade. The source code should be listed in such a way that copy-pasting to an editor and then saving to a file should produce a compile-able file and, in turn, an executable that works. If you happen to use ready solutions from the literature (yeah, yeah, it’s now a euphemism for “the Web”, I know!) make sure you properly acknowledge and cite your sources and mark what part of your code is taken from what source. Perfect grade is 100 points (the points add up to 110).