Assignment 2: Due Date: July 31 (Tuesday)

In this project, you are asked to write a program which uses three threads to concurrently compute the sum of square roots from 1 to n, where n is a multiple of 3 and is specified in the command line.

The program gets the value n from the command line argument (using argc / argv) and converts it to an integer. Next, it creates two threads using pthread_create(). Then, the main thread computes the sum of square roots from 1 to n/3, while the two child threads compute the sum of the square roots from n/3+1 to 2n/3 and 2n/3+1 to n, respectively, at the same time as main thread's calculation. All three partial sums should be added together to a shared global variable by each thread at the end of execution. The main thread calls pthread_join() to wait for the termination of the two child threads and then prints the result.

Below is an example of running your thread program:

spirit> thr 9999
sum of square roots: 666616.459197

You need to think about the location in program execution to call thread creation / mutex locking etc carefully. Be sure that three threads do calculation concurrently, no excessive mutex locking which creates unnecessary overhead, and is inefficient. Use mutex when necessary to avoid race conditions which cause your program to produce incorrect answers randomly. Programs which fail to do so will receive partial credit only even if the result is (sometimes) correct. You can assume the input is always valid and won't cause overflows if you use standard data types (such as int and doubles). You don't need to handle errors.

If you're not sure where to start, a guideline for the overall structure is provided below (pseudocode). You don't have to use this structure if your own code can meet all requirements above.

```
int start_points[4]
double grand_sum
mutex grand_sum_lock

function do_partial(start_point) {
   set sum to sum of square roots from start_point[0] to start_point[1]
   add sum to grand_sum, protected by grand_sum_lock
}

function main() {
   read n from argv and calculate start_points at 1, n/3+1, 2n/3+1, n+1
   start thread_1 using do_partial with start_points+1 as arg
   start thread_2 using do_partial with start_points+2 as arg
   call do_partial with start_points+0 as arg in main thread
   join thread_1 and thread_2
   print grand_sum as the final output
}
```