Homework #3: Sleeping Barber Problem (100 points)
Submit a compressed (.tgz) file with **source code** and **Makefile** to Canvas

You must write a *multithreaded* program in **C** that successfully simulates the **Sleeping Barber** concurrency problem defined as follows:

- A barbershop has:
  - B barbers
  - C clients (that arrive at random times), and
  - N chairs in the waiting room
- If there are no clients, the barber sleeps until a client arrives
- When a client arrives, *three* possibilities exist:
  1. If the barber is asleep:
     - Client wakes barber and gets a haircut
  2. If the barber is busy and there are chairs available in the waiting room:
     - Client waits for barber to finish, then gets a haircut
  3. If the barber is busy and there are NO available chairs in the waiting room:
     - Client leaves without a haircut

Your solution must meet the following requirements:

- Your program must accept **5** command line arguments:
  - `num_barbers` := number of barber threads
  - `num_clients` := number of client threads
  - `num_chairs` := number of chairs in waiting room
  - `arrival_t` := maximum time between clients
  - `haircut_t` := time required for a haircut (in μs)
    - All parameters must be *greater than* zero

- Your program must work for any reasonable number of *barbers* and *clients*
  - … and NOT result in deadlock
  - Haircuts must occur **concurrently** between a barber and a client…
    - Client cannot "cut his/her own hair"

- Each barber must output its progress to **stdout**, i.e.,
  - Barber's id number (from 0 to `num_barbers` – 1)
  - Barber's status:
    - Sleeping
    - Cutting hair

- Each client must output its progress to **stdout**, i.e.,
  - Client's id number (from 0 to `num_clients` – 1)
  - Client's status:
    - Arriving
    - Leaving (did not get a haircut)
    - Waiting
    - Getting haircut

- Clients must "arrive" to the barbershop at random times
  - I.e., between [1, `arrival_t`) microseconds
  - In other words, your main thread will wait [1, arrival_t) μs between creating client threads

- After all clients *and* barbers have finished, you must output the following statistics to **stderr**:
  - Total number of *successful* **haircuts**
  - The *average* **sleep time** (in μs) for all barbers
  - The *total* number of clients that **left without a haircut**
  - The *average* **wait time** (in μs) for all **clients** (who got a haircut)

Hints:

- struct timeval
- gettimeofday()
- *semaphores*
- usleep()
- sleep()
- pthread_cancel()
- next_client_time = random() % arrival_t + 1
  - usleep(next_client_time)
  - pthread_create(client, …)

Examples:

```
UNIX> ./hw3
usage: ./hw3 num_barbers num_clients num_chairs arrival_t haircut_t

//1 barber, 1 client, 1 chair, arrival_t = 100, haircut_t = 10 μs
UNIX> ./hw3 1 1 1 100 10
barber   0: sleeping...
client   0: arriving...
client   0: waiting...
barber   0: haircut...
client   0: haircut...
barber   0: sleeping...
TOTALS:
Total haircuts:                  1
Avg Barber sleep time:       162.00
Number clients that left:         0
Avg Client wait time:        112.00
```

```
// More barbers than clients + slow arrivals + fast haircuts = low wait times
//          and no clients left without haircut
// 100 barbers, 10 clients, 1000 chairs, arrival_t = 10000, haircut_t  = 10
UNIX> ./hw3 100 10 1000 10000 10 > output.txt
TOTALS:
Total haircuts:                  10
Avg Barber sleep time:      3765.06
Number clients that left:         0
Avg Client wait time:        125.20

// verify that haircuts occurred concurrently (barber comes before client..)
UNIX> grep –n haircut output.txt
103:barber   5: haircut...
104:client   0: haircut...
108:barber   6: haircut...
109:client   1: haircut...
113:barber   3: haircut...
114:client   2: haircut...
118:barber   2: haircut...
119:client   3: haircut...
123:barber   4: haircut...
124:client   4: haircut...
128:barber   1: haircut...
129:client   5: haircut...
133:barber   0: haircut...
134:client   6: haircut...
138:barber   8: haircut...
139:client   7: haircut...
143:barber  10: haircut...
144:client   8: haircut...
148:barber   9: haircut...
149:client   9: haircut...

// More clients than barbers + many chairs + fast arrivals + slow haircuts =
//          long wait times for clients & no clients that left
// 10 barbers, 100 clients, 10000 chairs, arrival_t = 10, haircut_t = 10000
UNIX> ./hw3 10 100 10000 10 10000 > output.txt
TOTALS:
Total haircuts:                 100
Avg Barber sleep time:      1102.00
Number clients that left:         0
Avg Client wait time:      39882.52

//Again, verify haircut concurrency:
UNIX> grep -n 'haircut' output.txt | head -n 4
13:barber   0: haircut...
14:client   0: haircut...
17:barber   7: haircut...
18:client   1: haircut...

// One barber + many clients + one chair + fast arrivals + slow haircuts =
//          many clients left without a haircut
UNIX> ./hw3 1 100 1 10 10000 > output.txt
TOTALS:
Total haircuts:                   2
Avg Barber sleep time:       124.00
Number clients that left:        98
Avg Client wait time:      24716.50
```