Exercise 1: “A simple rendezvous”

Suppose we have 2 threads A and B. Thread A does some complex calculations to some shared data. After that thread B will sort the results.

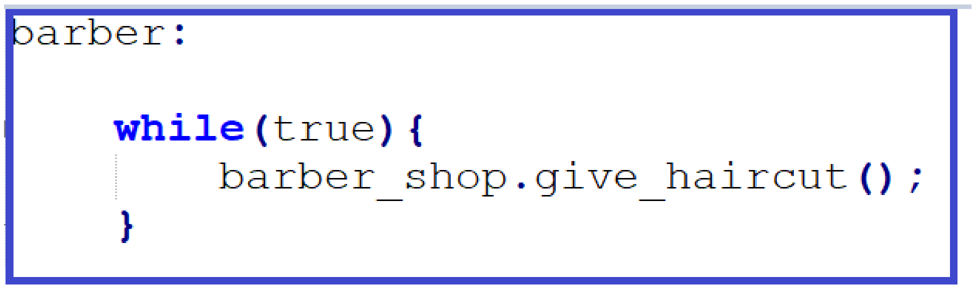
How can we make sure that thread B only sorts the processed data?

Fill in the blanks with semaphore operations(up/down). Some blanks may not require an answer. The CPU’s scheduling cannot be assumed.The sem can be either binary or counting.

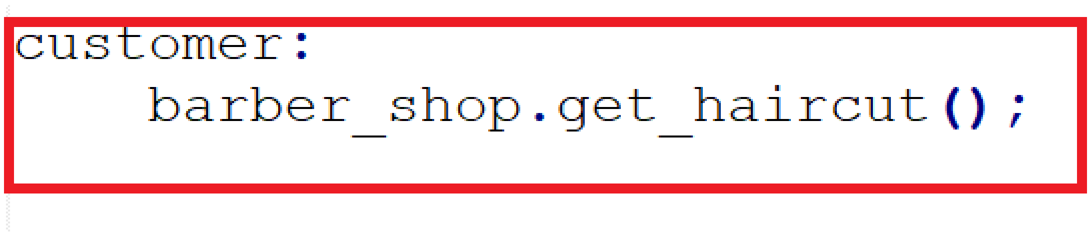
```
sem_t rendezvous = ...;


A:

...
complex_calc();
...


B:


...
sort();
...
```

Exercise 2: “Sleeping Barber with monitor”

Try to modify the semaphore solution into a solution with monitors. Assume signal-continue policy(after a signal call a thread continues executing code until it exits the monitor or waits on a condition) and eggshell(threads already inside the monitor have priority over threads blocked outside). Of course, if you don’t like the template you can change it.
The code the threads will run is a simple call of the monitor’s methods.



```
barber:

    while(true){
        barber_shop.give_haircut();
    }
```



```
customer:
    barber_shop.get_haircut();
```

Below is the suggested template for the monitor class:

```
monitor barber_shop{
    int free_chairs = N;
    condition customers,barber;

    give_haircut(){
        if(...)             //barbershop is empty
            ...             //sleep
        free_chairs++;
        ...                 //take the customer for a haircut
        //cut hair

    }

    get_haircut{
        if(free_chairs!=0){
            free_chairs--;
            if(...)             //i'm the first to enter
                ...             //wake the barber up!
            ...                 //sit in a chair and wait
            //get_haircut
        }
        else{
            //leave
        }

    }

}
```

Exercise 3: “Pool of workers”

Our main thread receives a number of integers and has to decide if they are prime numbers or not. To speed up the process of calculation, it spawns a pool of N workers to do the calculations using int primetest(int k). Using semaphores write pseudocode that implements the synchronization in this philosophy:
(Keep in mind race conditions on shared variables)
(Semaphores can be either binary or counting)

```
main thread:
create workers
while (job exists){
    wait for a worker to become available
    assign next job to an available worker
    notify the worker to do the job



}
```

```
worker thread:
while(true){
    notify main that I am available
    wait for notification by main
    process assigned job
}
```