HARSH KUMAR AGARWAL
2019423

Assuming we know the basic dining philosopher problem with few changes as per the question. Ass41 implements using blocking and ass42 implements using a non blocking mechanism.

**Struct created:**

helperStruct: we create this struct so that we can pass value all at once through one variable to the function while using thread.
It contains fork,forknext and a numerical value
My_semaphore: this struct was asked in the question to be created, it is used to avoid any possible deadlock

**Functions created(blocking implementation):**

PrintValue(): This function is used for debugging.it takes semaphore and prints its value.
wait(): we first lock the critical section(here the change in value of val), then we make other thread wait if the val is less than 0 (this means that the portion of that code is running in any thread, and this thread needs to wait)
signal():we first lock the critical section(here the change in value of val), then we give signal if that portion of the code is free(checked using the condition val<=0)
working(): Here first we copy data to local variables, then we use the wait function created before to make sure that the resources are available. Once we do that we print as was required in the question, then use a sleep function(so that the user can actually see the working while running) to demonstrate eating of philosophers. Then we use the signal function created before to send the signal to the wait function that the thread was done with that portion of the code.
main(): Here we ask the user for the number of philosophers, make threads, initialise them. We call other functions in different threads, and finally join the threads.

**Functions created(Non-blocking implementation)**

printValue and main have the same implementation for non blocking.
wait(): here we first check if the value of sem->val is greater than 0, we keep it waiting until it is achieved(it means that critical section is free of any operation by thread now). Then we try to lock(enter) the critical section and keep on trying until we get the chance to do so(the operation of that specific thread is kept there until we enter the critical section), we reduce the val by 1 and unlock the critical section.
signal(): We try to lock(enter) the critical section and keep on trying until we get the chance to do so(the operation of that specific thread is kept there until we enter the critical section), we increase the val by 1 and unlock the critical section.

**Basic Idea of what we are trying to do:**

In both the cases we are basically trying somehow to let enter only thread at a time to the critical section(both wait and signal function is used to do so).

Once we are able to do so, we let only thread to execute at a time which has both the bowls(we created a single bowl as philosophers have to use both of them at a time and this helps to avoid any kind of deadlock).Then we let the philosophers pick one fork at a time, after the philosopher have eaten(printing the value and waiting shows this action), we keep back the resources(means gives signal to other thread that the current thread is done with that part). This keeps on going in a loop.

To exit, user have to terminate it(using ctrl+c).

For any further explanation please contact me.