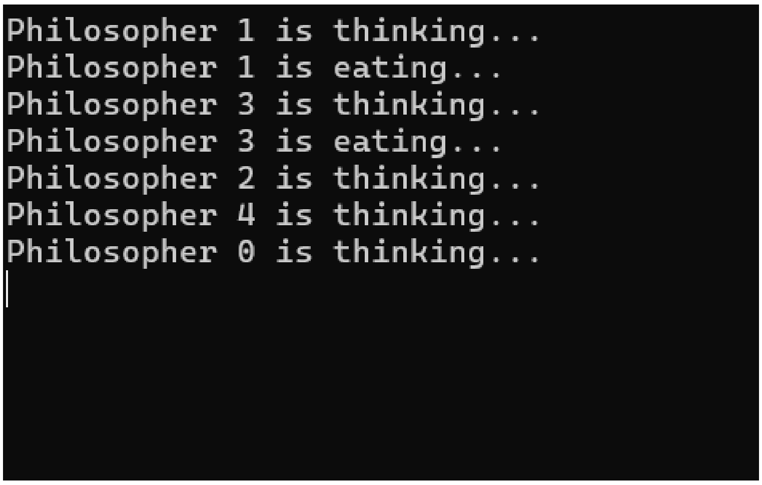
12. Design a C program to simulate the concept of Dining-Philosophers problem.

PROGRAM :

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
#define NUM_PHILOSOPHERS 5
pthread_mutex_t chopsticks[NUM_PHILOSOPHERS];
void* philosopherLifeCycle(void* arg) {
int id = *((int*)arg);
int left_chopstick = id;
int right_chopstick = (id + 1) % NUM_PHILOSOPHERS;
while (1) {
// Think
printf("Philosopher %d is thinking...\n", id);
// Pick up chopsticks
pthread_mutex_lock(&chopsticks[left_chopstick]);
pthread_mutex_lock(&chopsticks[right_chopstick]);
// Eat
printf("Philosopher %d is eating...\n", id);
sleep(rand() % 3 + 1); // Eating time
// Put down chopsticks
pthread_mutex_unlock(&chopsticks[left_chopstick]);
pthread_mutex_unlock(&chopsticks[right_chopstick]);
// Repeat the cycle
}
}
int main() {
pthread_t philosophers[NUM_PHILOSOPHERS];
int philosopher_ids[NUM_PHILOSOPHERS];
// Initialize mutex locks
for (int i = 0; i < NUM_PHILOSOPHERS; ++i) {
```

```
pthread_mutex_init(&chopsticks[i], NULL);
}
// Create philosopher threads
for (int i = 0; i < NUM_PHILOSOPHERS; ++i) {
philosopher_ids[i] = i;
pthread_create(&philosophers[i], NULL, philosopherLifeCycle, (void*)&philosopher_ids[i]);
}
// Wait for threads to finish (although they run indefinitely)
for (int i = 0; i < NUM_PHILOSOPHERS; ++i) {
pthread_join(philosophers[i], NULL);
}
// Destroy mutex locks
for (int i = 0; i < NUM_PHILOSOPHERS; ++i) {
pthread_mutex_destroy(&chopsticks[i]);
}
return 0;
}
```

OUTPUT :



```
Philosopher 1 is thinking...
Philosopher 1 is eating...
Philosopher 3 is thinking...
Philosopher 3 is eating...
Philosopher 2 is thinking...
Philosopher 4 is thinking...
Philosopher 0 is thinking...
```