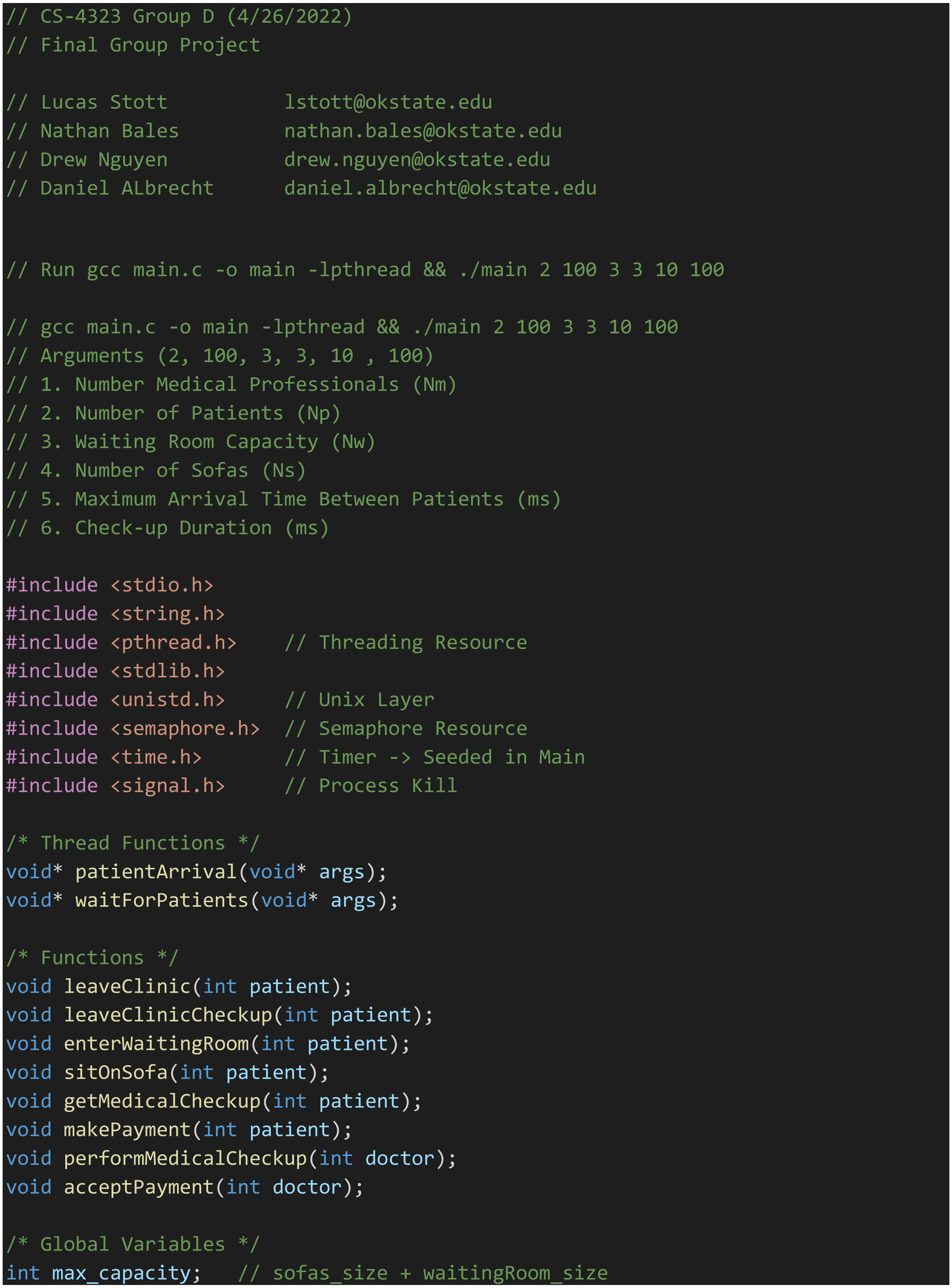
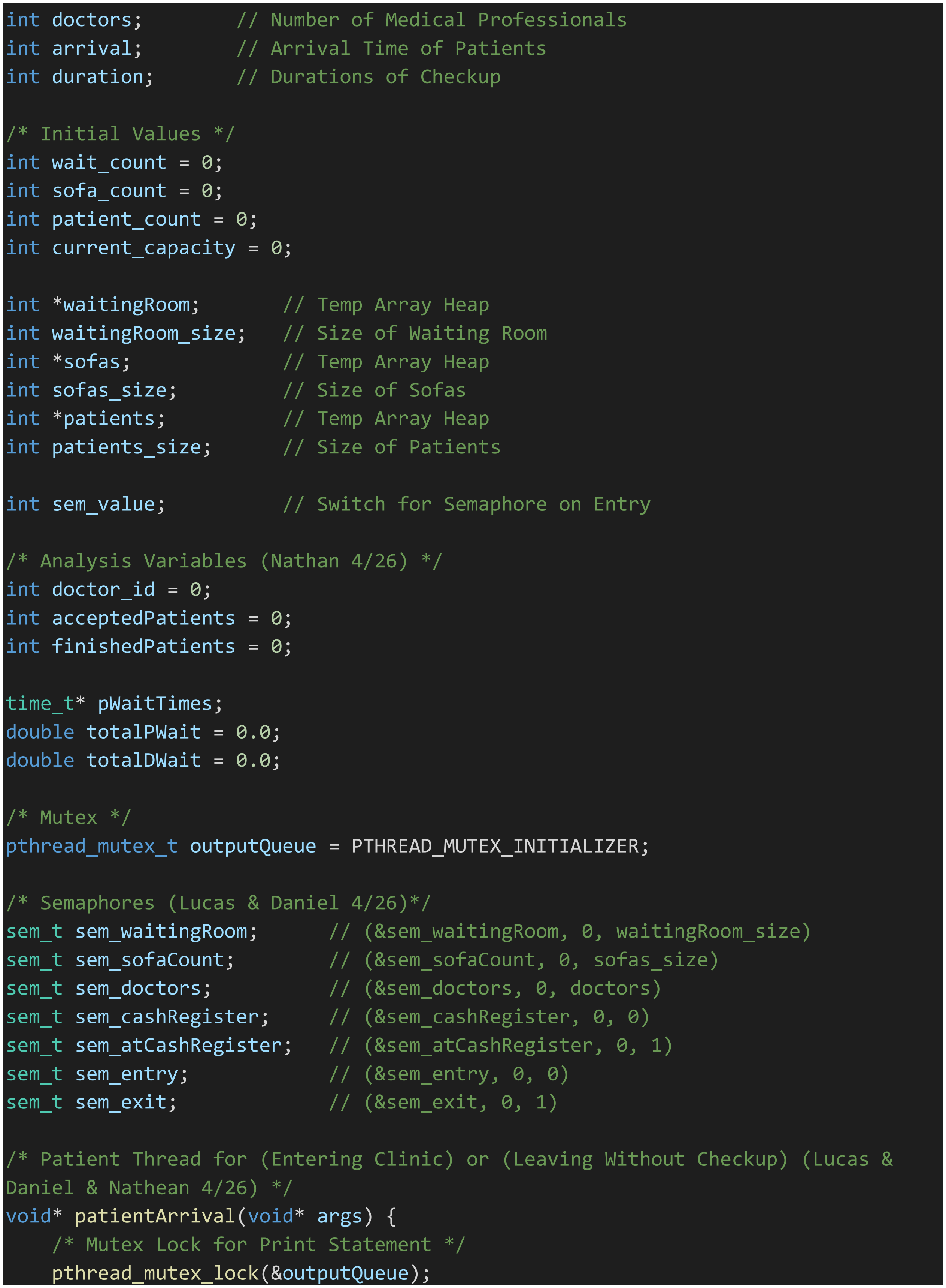
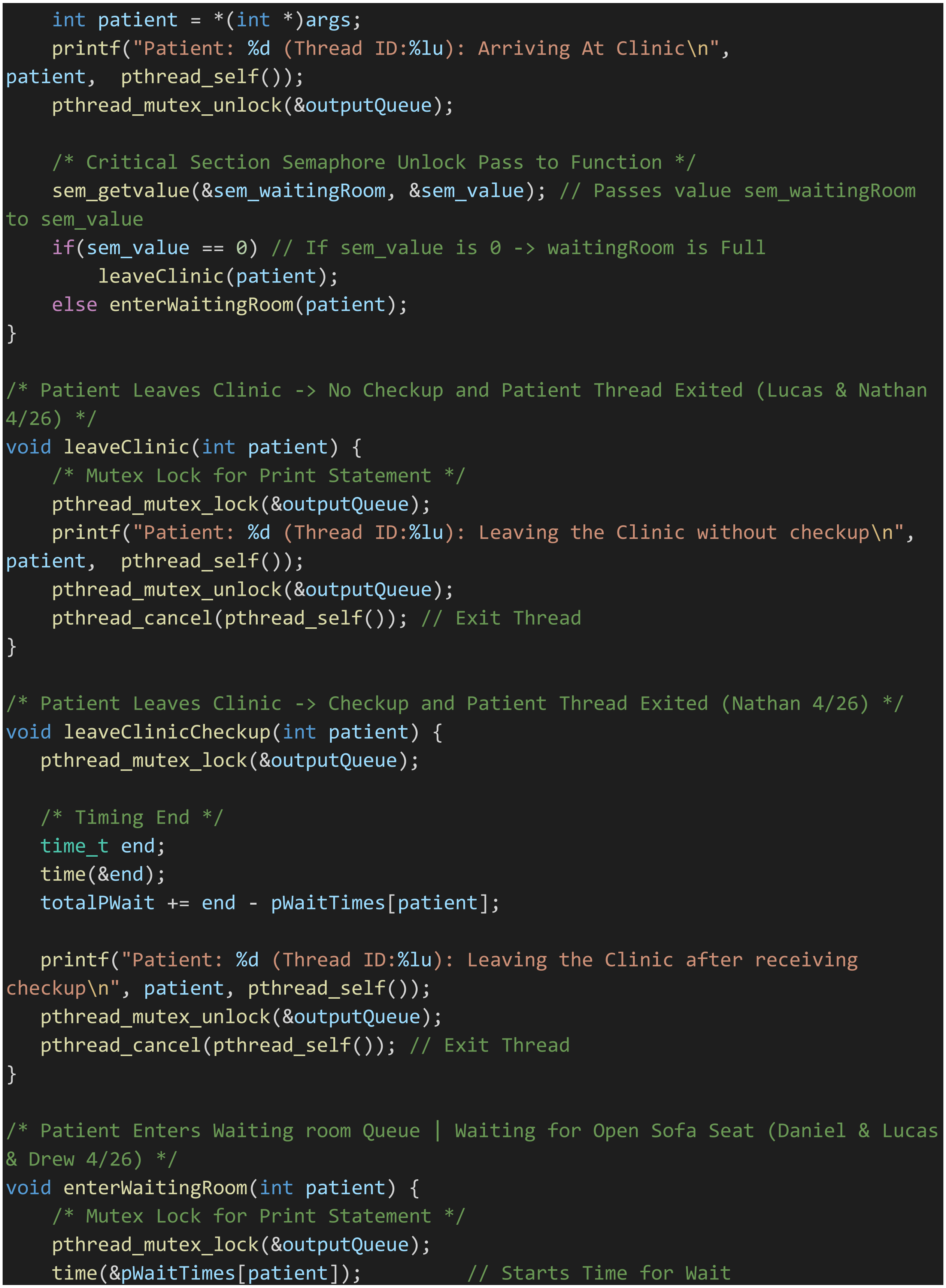


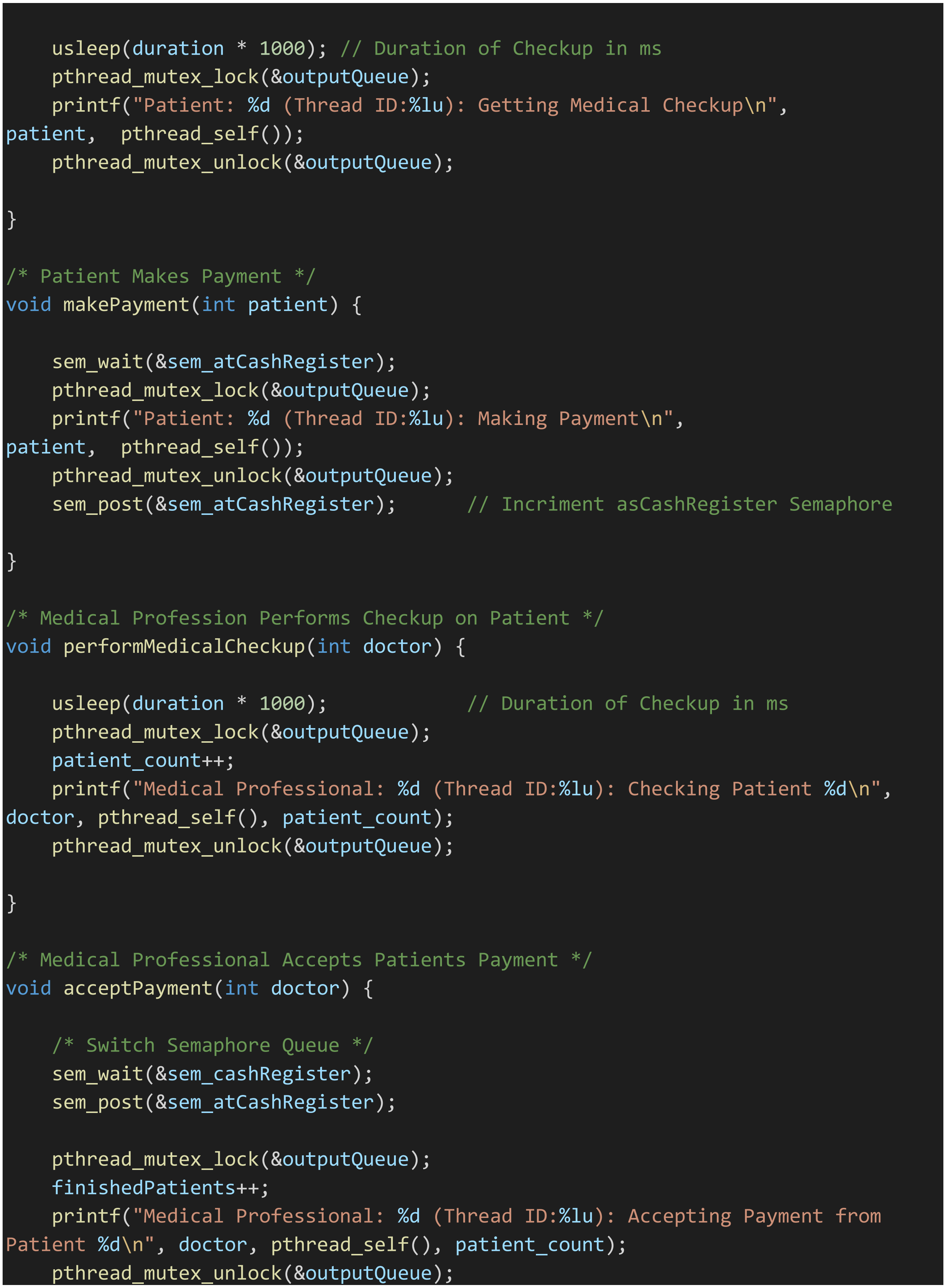
```c
// CS-4323 Group D (4/26/2022)
// Final Group Project

// Lucas Stott          lstott@okstate.edu
// Nathan Bales         nathan.bales@okstate.edu
// Drew Nguyen          drew.nguyen@okstate.edu
// Daniel ALbrecht      daniel.albrecht@okstate.edu


// Run gcc main.c -o main -lpthread && ./main 2 100 3 3 10 100

// gcc main.c -o main -lpthread && ./main 2 100 3 3 10 100
// Arguments (2, 100, 3, 3, 10 , 100)
// 1. Number Medical Professionals (Nm)
// 2. Number of Patients (Np)
// 3. Waiting Room Capacity (Nw)
// 4. Number of Sofas (Ns)
// 5. Maximum Arrival Time Between Patients (ms)
// 6. Check-up Duration (ms)

#include <stdio.h>
#include <string.h>
#include <pthread.h>    // Threading Resource
#include <stdlib.h>
#include <unistd.h>     // Unix Layer
#include <semaphore.h>  // Semaphore Resource
#include <time.h>       // Timer -> Seeded in Main
#include <signal.h>     // Process Kill

/* Thread Functions */
void* patientArrival(void* args);
void* waitForPatients(void* args);

/* Functions */
void leaveClinic(int patient);
void leaveClinicCheckup(int patient);
void enterWaitingRoom(int patient);
void sitOnSofa(int patient);
void getMedicalCheckup(int patient);
void makePayment(int patient);
void performMedicalCheckup(int doctor);
void acceptPayment(int doctor);

/* Global Variables */
int max_capacity;   // sofas_size + waitingRoom_size
```



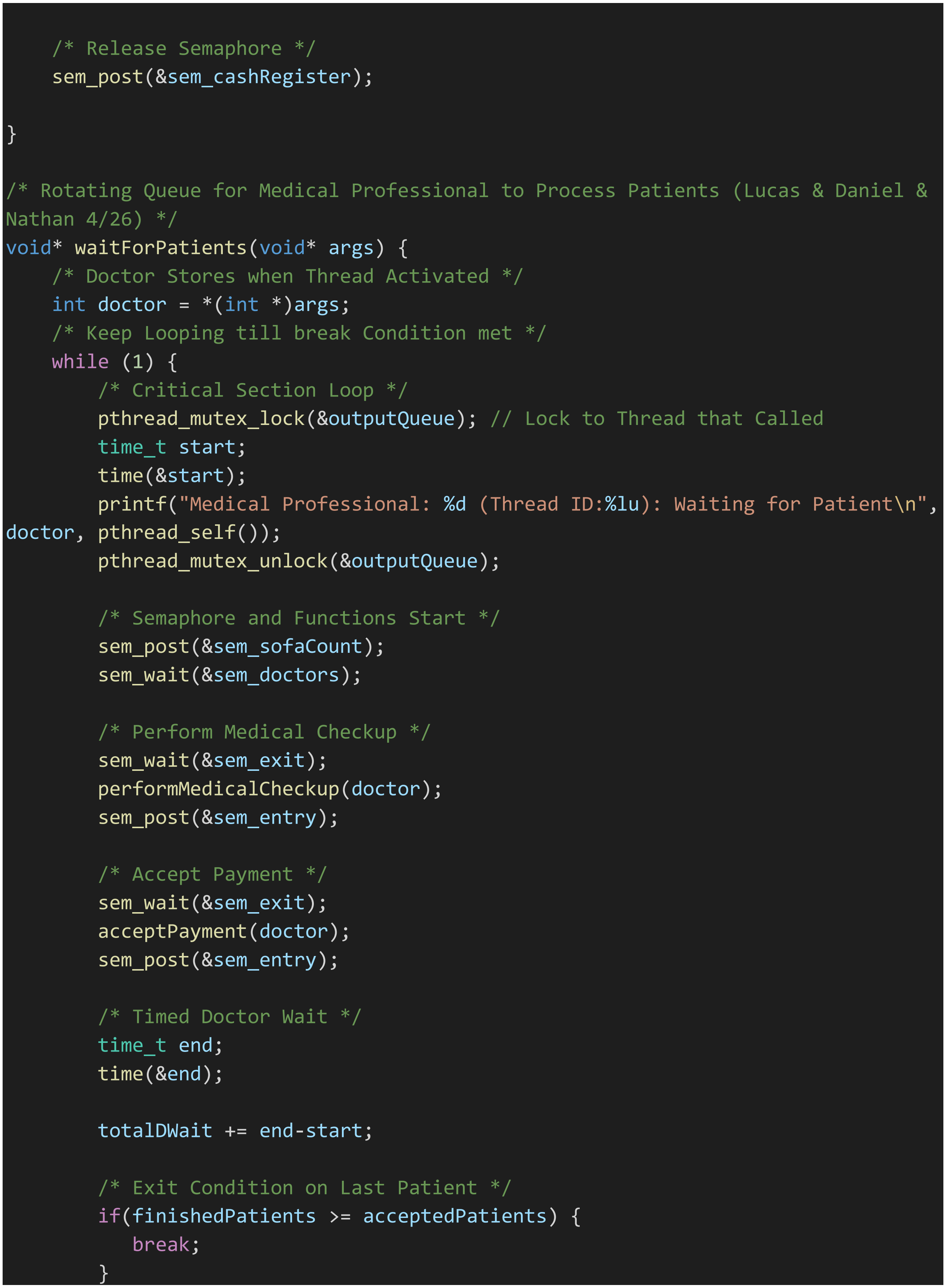
```
int doctors;        // Number of Medical Professionals
int arrival;        // Arrival Time of Patients
int duration;       // Durations of Checkup

/* Initial Values */
int wait_count = 0;
int sofa_count = 0;
int patient_count = 0;
int current_capacity = 0;

int *waitingRoom;       // Temp Array Heap
int waitingRoom_size;   // Size of Waiting Room
int *sofas;             // Temp Array Heap
int sofas_size;         // Size of Sofas
int *patients;          // Temp Array Heap
int patients_size;      // Size of Patients

int sem_value;          // Switch for Semaphore on Entry

/* Analysis Variables (Nathan 4/26) */
int doctor_id = 0;
int acceptedPatients = 0;
int finishedPatients = 0;

time_t* pWaitTimes;
double totalPWait = 0.0;
double totalDWait = 0.0;

/* Mutex */
pthread_mutex_t outputQueue = PTHREAD_MUTEX_INITIALIZER;

/* Semaphores (Lucas & Daniel 4/26)*/
sem_t sem_waitingRoom;      // (&sem_waitingRoom, 0, waitingRoom_size)
sem_t sem_sofaCount;        // (&sem_sofaCount, 0, sofas_size)
sem_t sem_doctors;          // (&sem_doctors, 0, doctors)
sem_t sem_cashRegister;     // (&sem_cashRegister, 0, 0)
sem_t sem_atCashRegister;   // (&sem_atCashRegister, 0, 1)
sem_t sem_entry;            // (&sem_entry, 0, 0)
sem_t sem_exit;             // (&sem_exit, 0, 1)

/* Patient Thread for (Entering Clinic) or (Leaving Without Checkup) (Lucas &
Daniel & Nathean 4/26) */
void* patientArrival(void* args) {
    /* Mutex Lock for Print Statement */
    pthread_mutex_lock(&outputQueue);
```



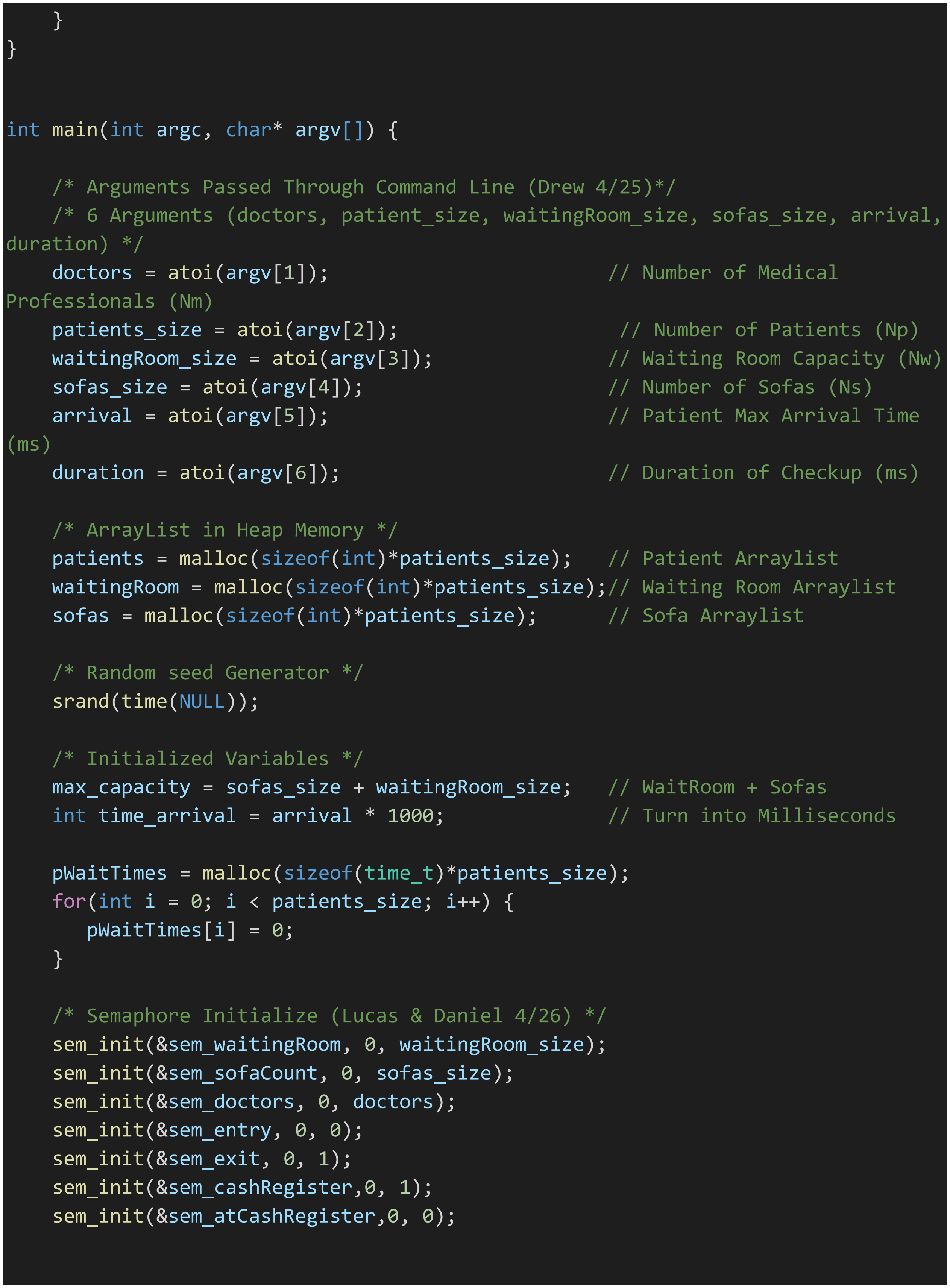
```
    int patient = *(int *)args;
    printf("Patient: %d (Thread ID:%lu): Arriving At Clinic\n",
patient,  pthread_self());
    pthread_mutex_unlock(&outputQueue);

    /* Critical Section Semaphore Unlock Pass to Function */
    sem_getvalue(&sem_waitingRoom, &sem_value); // Passes value sem_waitingRoom
to sem_value
    if(sem_value == 0) // If sem_value is 0 -> waitingRoom is Full
        leaveClinic(patient);
    else enterWaitingRoom(patient);
}

/* Patient Leaves Clinic -> No Checkup and Patient Thread Exited (Lucas & Nathan
4/26) */
void leaveClinic(int patient) {
    /* Mutex Lock for Print Statement */
    pthread_mutex_lock(&outputQueue);
    printf("Patient: %d (Thread ID:%lu): Leaving the Clinic without checkup\n",
patient,  pthread_self());
    pthread_mutex_unlock(&outputQueue);
    pthread_cancel(pthread_self()); // Exit Thread
}

/* Patient Leaves Clinic -> Checkup and Patient Thread Exited (Nathan 4/26) */
void leaveClinicCheckup(int patient) {
   pthread_mutex_lock(&outputQueue);

   /* Timing End */
   time_t end;
   time(&end);
   totalPWait += end - pWaitTimes[patient];

   printf("Patient: %d (Thread ID:%lu): Leaving the Clinic after receiving
checkup\n", patient, pthread_self());
   pthread_mutex_unlock(&outputQueue);
   pthread_cancel(pthread_self()); // Exit Thread
}

/* Patient Enters Waiting room Queue | Waiting for Open Sofa Seat (Daniel & Lucas
& Drew 4/26) */
void enterWaitingRoom(int patient) {
    /* Mutex Lock for Print Statement */
    pthread_mutex_lock(&outputQueue);
    time(&pWaitTimes[patient]);         // Starts Time for Wait
    acceptedPatients++;                  // Incriment Number of Accepted Patients
    printf("Patient: %d (Thread ID:%lu): Entering Waiting Room\n",
patient,  pthread_self());
    pthread_mutex_unlock(&outputQueue);

    /* Outside Critical Section */
    sem_wait(&sem_waitingRoom); // Incriment Semaphore sem_waitingRoom
    sitOnSofa(patient);
}

/* Patient Sits on Sofa Queue | Waiting for Doctor (Lucas & Daniel 4/26) */
void sitOnSofa(int patient) {
    /* Mutex Lock for Print Statement */
    pthread_mutex_lock(&outputQueue);
    printf("Patient: %d (Thread ID:%lu): Sitting On Sofa\n",
patient,  pthread_self());
    pthread_mutex_unlock(&outputQueue);

    /* Outside Critical Section */

    sem_wait(&sem_sofaCount); // Wait if Semaphore is Full

    /* Activate Waiting Room and Doctor Semaphors */
    sem_post(&sem_waitingRoom);
    sem_post(&sem_doctors);

    /* Get Checkup */
    sem_wait(&sem_entry);
    getMedicalCheckup(patient);
    sem_post(&sem_exit);

    /* Make Payment */
    sem_wait(&sem_entry);
    makePayment(patient);
    sem_post(&sem_exit);

    /* Leave Clinic */
    sem_wait(&sem_entry);
    leaveClinicCheckup(patient);
    sem_post(&sem_exit);
}

/* Function Blocks by Lucas & Nathan & Daniel & Drew 4/26) */
/* Patient Starts Checkup With Medical Professional */
void getMedicalCheckup(int patient) {
```



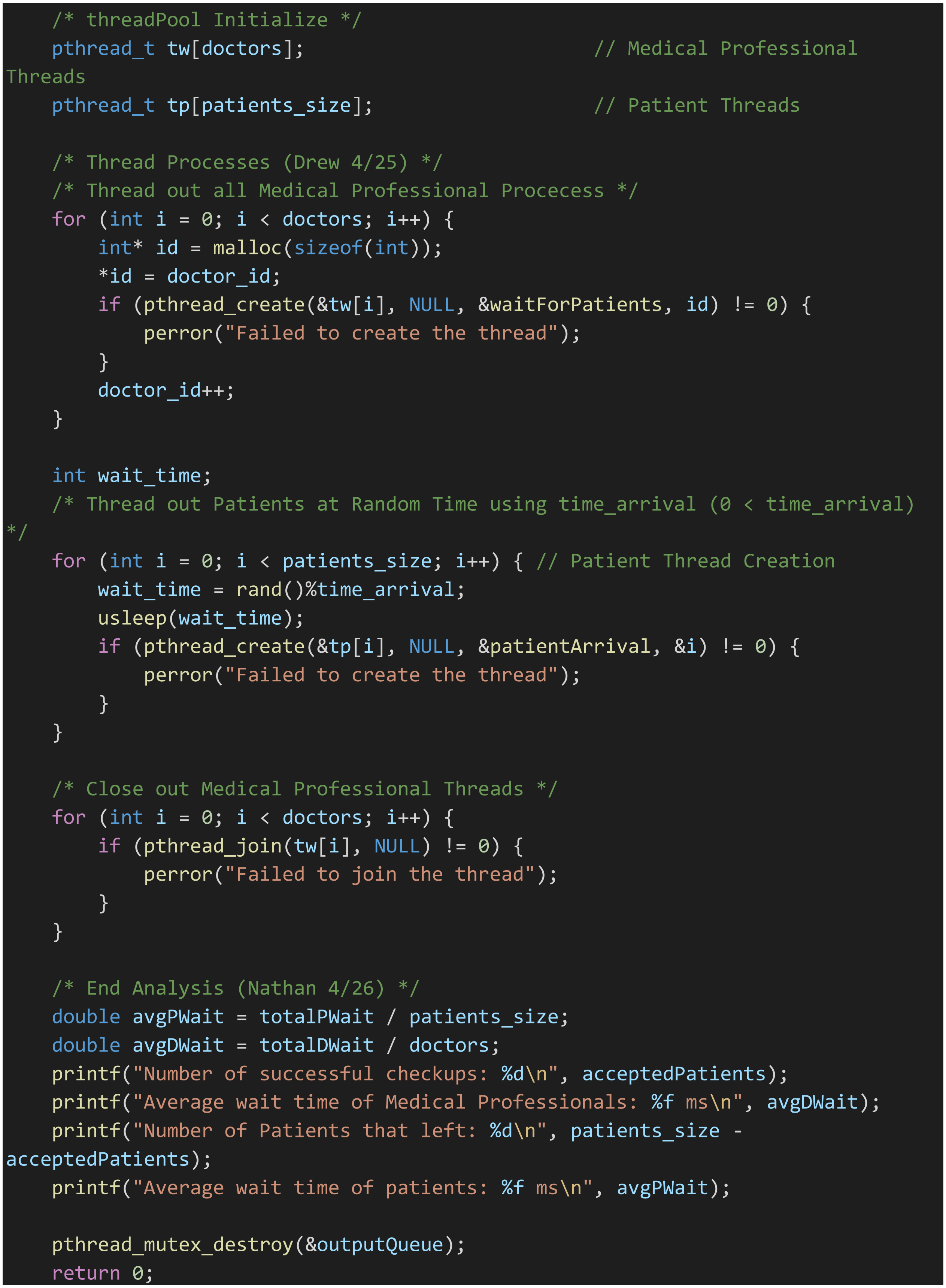
```
    usleep(duration * 1000); // Duration of Checkup in ms
    pthread_mutex_lock(&outputQueue);
    printf("Patient: %d (Thread ID:%lu): Getting Medical Checkup\n",
patient,  pthread_self());
    pthread_mutex_unlock(&outputQueue);

}

/* Patient Makes Payment */
void makePayment(int patient) {

    sem_wait(&sem_atCashRegister);
    pthread_mutex_lock(&outputQueue);
    printf("Patient: %d (Thread ID:%lu): Making Payment\n",
patient,  pthread_self());
    pthread_mutex_unlock(&outputQueue);
    sem_post(&sem_atCashRegister);      // Incriment asCashRegister Semaphore

}

/* Medical Profession Performs Checkup on Patient */
void performMedicalCheckup(int doctor) {

    usleep(duration * 1000);            // Duration of Checkup in ms
    pthread_mutex_lock(&outputQueue);
    patient_count++;
    printf("Medical Professional: %d (Thread ID:%lu): Checking Patient %d\n",
doctor, pthread_self(), patient_count);
    pthread_mutex_unlock(&outputQueue);

}

/* Medical Professional Accepts Patients Payment */
void acceptPayment(int doctor) {

    /* Switch Semaphore Queue */
    sem_wait(&sem_cashRegister);
    sem_post(&sem_atCashRegister);

    pthread_mutex_lock(&outputQueue);
    finishedPatients++;
    printf("Medical Professional: %d (Thread ID:%lu): Accepting Payment from
Patient %d\n", doctor, pthread_self(), patient_count);
    pthread_mutex_unlock(&outputQueue);
```



```
    /* Release Semaphore */
    sem_post(&sem_cashRegister);

}

/* Rotating Queue for Medical Professional to Process Patients (Lucas & Daniel &
Nathan 4/26) */
void* waitForPatients(void* args) {
    /* Doctor Stores when Thread Activated */
    int doctor = *(int *)args;
    /* Keep Looping till break Condition met */
    while (1) {
        /* Critical Section Loop */
        pthread_mutex_lock(&outputQueue); // Lock to Thread that Called
        time_t start;
        time(&start);
        printf("Medical Professional: %d (Thread ID:%lu): Waiting for Patient\n",
doctor, pthread_self());
        pthread_mutex_unlock(&outputQueue);

        /* Semaphore and Functions Start */
        sem_post(&sem_sofaCount);
        sem_wait(&sem_doctors);

        /* Perform Medical Checkup */
        sem_wait(&sem_exit);
        performMedicalCheckup(doctor);
        sem_post(&sem_entry);

        /* Accept Payment */
        sem_wait(&sem_exit);
        acceptPayment(doctor);
        sem_post(&sem_entry);

        /* Timed Doctor Wait */
        time_t end;
        time(&end);

        totalDWait += end-start;

        /* Exit Condition on Last Patient */
        if(finishedPatients >= acceptedPatients) {
           break;
        }
```



```
    }
}


int main(int argc, char* argv[]) {

    /* Arguments Passed Through Command Line (Drew 4/25)*/
    /* 6 Arguments (doctors, patient_size, waitingRoom_size, sofas_size, arrival,
duration) */
    doctors = atoi(argv[1]);                        // Number of Medical
Professionals (Nm)
    patients_size = atoi(argv[2]);                   // Number of Patients (Np)
    waitingRoom_size = atoi(argv[3]);               // Waiting Room Capacity (Nw)
    sofas_size = atoi(argv[4]);                     // Number of Sofas (Ns)
    arrival = atoi(argv[5]);                        // Patient Max Arrival Time
(ms)
    duration = atoi(argv[6]);                       // Duration of Checkup (ms)

    /* ArrayList in Heap Memory */
    patients = malloc(sizeof(int)*patients_size);   // Patient Arraylist
    waitingRoom = malloc(sizeof(int)*patients_size);// Waiting Room Arraylist
    sofas = malloc(sizeof(int)*patients_size);      // Sofa Arraylist

    /* Random seed Generator */
    srand(time(NULL));

    /* Initialized Variables */
    max_capacity = sofas_size + waitingRoom_size;   // WaitRoom + Sofas
    int time_arrival = arrival * 1000;              // Turn into Milliseconds

    pWaitTimes = malloc(sizeof(time_t)*patients_size);
    for(int i = 0; i < patients_size; i++) {
       pWaitTimes[i] = 0;
    }

    /* Semaphore Initialize (Lucas & Daniel 4/26) */
    sem_init(&sem_waitingRoom, 0, waitingRoom_size);
    sem_init(&sem_sofaCount, 0, sofas_size);
    sem_init(&sem_doctors, 0, doctors);
    sem_init(&sem_entry, 0, 0);
    sem_init(&sem_exit, 0, 1);
    sem_init(&sem_cashRegister,0, 1);
    sem_init(&sem_atCashRegister,0, 0);
```



```
    /* threadPool Initialize */
    pthread_t tw[doctors];                         // Medical Professional
Threads
    pthread_t tp[patients_size];                   // Patient Threads

    /* Thread Processes (Drew 4/25) */
    /* Thread out all Medical Professional Procecess */
    for (int i = 0; i < doctors; i++) {
        int* id = malloc(sizeof(int));
        *id = doctor_id;
        if (pthread_create(&tw[i], NULL, &waitForPatients, id) != 0) {
            perror("Failed to create the thread");
        }
        doctor_id++;
    }

    int wait_time;
    /* Thread out Patients at Random Time using time_arrival (0 < time_arrival)
*/
    for (int i = 0; i < patients_size; i++) { // Patient Thread Creation
        wait_time = rand()%time_arrival;
        usleep(wait_time);
        if (pthread_create(&tp[i], NULL, &patientArrival, &i) != 0) {
            perror("Failed to create the thread");
        }
    }

    /* Close out Medical Professional Threads */
    for (int i = 0; i < doctors; i++) {
        if (pthread_join(tw[i], NULL) != 0) {
            perror("Failed to join the thread");
        }
    }

    /* End Analysis (Nathan 4/26) */
    double avgPWait = totalPWait / patients_size;
    double avgDWait = totalDWait / doctors;
    printf("Number of successful checkups: %d\n", acceptedPatients);
    printf("Average wait time of Medical Professionals: %f ms\n", avgDWait);
    printf("Number of Patients that left: %d\n", patients_size -
acceptedPatients);
    printf("Average wait time of patients: %f ms\n", avgPWait);

    pthread_mutex_destroy(&outputQueue);
    return 0;
```

```
}
```