```
// GROUP D
// By Corey Green
// decoreyon.green@okstate.edu
// CS 4323
// finalGroupProject
// 4-26-22
#include "Robby.h"
#include "Kyle.h"
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <semaphore.h>
#include <pthread.h>
#include <sys/types.h>
#include <time.h>
// func for leaving clinic
/*
 * Mutex set for one patient to leave at a time so no two patients get counted as one
 * struct threadStruct *contents is a pointer to a stuct containing the patient information
 * is succsessful is an int holding the number of successful pateints for the summary
 */

void leaveClinic(struct threadStruct *contents, int isSuccessful)
{
   // mutex for keeping up with patients who left
   pthread_mutex_unlock(&mutex[9]);
   pthread_mutex_lock(&mutex[9]);
   left++;
   pthread_mutex_unlock(&mutex[9]);
```

```
    if (isSuccessful){
        printf("Patient %d (ThreadID: %d): Leaving the clinic after receiving checkup\n", contents->id,
contents->threadID);
        // mutex for getting the paitent's wait time
        contents->waitTime = clock() - contents->waitTime;
        pthread_mutex_lock(&mutex[10]);
        summary.patientsAvgWaitTime += contents->waitTime;
        pthread_mutex_unlock(&mutex[10]);
            // mutex for incrementing patients who got successful checkups
        pthread_mutex_lock(&mutex[10]);
        summary.successfulCheckups++;
        pthread_mutex_unlock(&mutex[10]);
    } else {
        printf("Patient %d (Thread ID: %d): Leaving without checkup.\n", contents->id, contents->threadID);
        // mutex for incrementing patients who got left without checkups
        pthread_mutex_lock(&mutex[10]);
        summary.patientsThatLeft++;
        pthread_mutex_unlock(&mutex[10]);
    }
}
// func for medical checkup behavior
/*
 * Mutex set for one patient to get checkup at a time, similar to the leaveClinic func
 * struct threadStruct *contents is a pointer to a stuct containing the patient information
 */
void getMedicalCheckup(struct threadStruct *contents)
{
    int staffId;
    // mutex for incrementing sofa capacity and announcing who's being checked up
```

```
    pthread_mutex_lock(&mutex[1]);
    totalSofaCapacity++;
    pthread_mutex_unlock(&mutex[1]);
    pthread_mutex_lock(&mutex[0]);
    totalRoomCapacity++;
    printf("Patient %d (ThreadID: %d): Getting checkup\n", contents->id, contents->threadID);

    pthread_mutex_unlock(&mutex[0]);

    pthread_mutex_lock(&mutex[3]);

    buffer = contents->id;
    pthread_mutex_unlock(&mutex[4]);
    pthread_mutex_lock(&mutex[5]);

    contents->bondId = buffer;

    pthread_mutex_unlock(&mutex[3]);
    // to ensure summary prints last
    usleep(checkupTime * 1000);

}
// func for making payment
/*
 * Mutex set setup to work as one register so that one patient pays at a time
 * struct threadStruct *contents is a pointer to a stuct containing the patient information
 */
void makePayment(struct threadStruct *contents)
{
```

```
    // mutex to insure one patient pays at a time
    pthread_mutex_lock(&mutex[8]);
    printf("Patient %d (ThreadID: %d): Making payment to Medical Staff %d\n", contents->id, contents->threadID, contents->bondId);
    pthread_mutex_unlock(&mutex[2]);
    pthread_mutex_lock(&mutex[6]);
    pthread_mutex_unlock(&mutex[8]);
}
```