حوری دهش 9821413

## سوال 1:

```
lock(m)
guest_count += 1
if guest_count < N:
    wait(cv_guest, m)
else:
    signal(cv_host)
    openDoor()
    signal(cv_guest)
unlock(m)
```

توضیحات:

1. lock(m): قفل کردن mutex تا فقط یک نخ اجازه دسترسی به این بخش را داشته باشد
2. guest_count += 1: افزایش تعداد مهمانها
3. if guest_count < N: اگر تعداد مهماها کمتر از N باشد، منتظر ورود بقیه مهمانها می ماند
4. wait(cv_guest, m): انتظار برای ورود بقیه مهمانها به خانه
5. else: اگر تعداد مهمانها برابر یا بیشتر از N باشد، openDoor() را صدا می زند
6. openDoor(): باز کردن در برای ورود مهمانها
7. signal(cv_host): ارسال سیگنال به میزبان برای اطلاع از آماده بودن برای باز کردن در
8. signal(cv_guest): ارسال سیگنال به مهمانها برای ورود به خانه
9. unlock(m): باز کردن mutex تا نخ های دیگر هم به این بخش دسترسی داشته باشند

## سوال 2:

### سودوکدهای لازم برای پیاده سازی توابع این سوال:

```
// تعریف دو Mutex برای حفاظت از متغیرهای مشترک
mutex globalMutex; // برای حفاظت از متغیرهای مشترک
mutex readerCountMutex; // برای حفاظت از تعداد خوانندگان فعال

// تعریف متغیرهای مشترک
int activeReaders = 0; // تعداد خوانندگان فعال
bool writeInProgress = false; // نشان‌دهنده انجام عملیات نویسندگی است

// تعریف Conditional Variables
condition_variable noReaders; // برای اطمینان از عدم وجود خوانندگان فعال
condition_variable noWriters; // برای اطمینان از عدم وجود نویسندگان فعال

// توابع قفل خوانندگان و نویسندگان
void readLock() {
    unique_lock<mutex> lock(globalMutex); // قفل اصلی را بگیرید
    unique_lock<mutex> countLock(readerCountMutex); // قفل برای تعداد خوانندگان را بگیرید

    // اگر نویسنده فعالی در حال نوشتن نباشد و هیچ خواننده فعالی نباشد، می‌توان خواند
    noWriters.wait(lock, [] { return !writeInProgress && activeReaders == 0; });

    activeReaders++;
}

void readUnlock() {
    unique_lock<mutex> countLock(readerCountMutex); // قفل برای تعداد خوانندگان را بگیرید

    // کاهش تعداد خوانندگان فعال و در صورت نیاز، اطلاع رساندن به نویسندگان
    if (--activeReaders == 0) {
        noReaders.notify_one();
    }
}

void writeLock() {
    unique_lock<mutex> lock(globalMutex); // قفل اصلی را بگیرید

    // اگر هیچ خواننده فعالی نباشد و هیچ نویسنده فعالی نباشد، می‌توان نوشت
    noReaders.wait(lock, [] { return activeReaders == 0 && !writeInProgress; });

    writeInProgress = true; // نشان دهید که یک نویسنده فعال است
}


void writeUnlock() {
    unique_lock<mutex> lock(globalMutex); // قفل اصلی را بگیرید

    writeInProgress = false; // نشان دهید که نوشتن تمام شده است

    // اطلاع رساندن به خوانندگان و نویسندگان
    noReaders.notify_all();
    noWriters.notify_all();
}
```

## سوال 3:

```
#include <stdio.h>

// تعریف سمافورها و میوتکس‌ها
Semaphore northSem = 1;    // سمافور برای کنترل ترافیک به سمت شمال
Semaphore southSem = 1;    // سمافور برای کنترل ترافیک به سمت جنوب
Mutex bridgeMutex;         // میوتکس برای حفاظت از متغیرهای مشترک
Semaphore limitSem = 5;    // سمافور برای محدود کردن تعداد ماشین‌های در حال عبور از هر سمت
int flag = 0;              // وضعیت درخواست‌ها، اگر flag برابر ۱ باشد، درخواست‌ها تغییر کرده است.

// متغیرهای مشترک
int northCount = 0;        // تعداد ماشین‌های در حال عبور به سمت شمال از پل
int southCount = 0;        // تعداد ماشین‌های در حال عبور به سمت جنوب از پل


// تابع برای ماشین‌های در حال حرکت به سمت شمال
void north() {
    wait(northSem); // منتظر اجازه عبور از سمت شمال باشید

    wait(bridgeMutex); // ورود به بخش حساس

    // بررسی اینکه آیا ماشین‌های در حال حرکت به سمت جنوب پل را عبور کرده‌اند یا خیر
    if (southCount > 0) {
        flag = 1;    // تغییر وضعیت درخواست ها
        signal(northSem); // آزاد کردن سمافور ترافیک به سمت شمال
        signal(bridgeMutex); // خروج از بخش حساس
        wait(southSem); // منتظر اجازه عبور از سمت جنوب باشید
        wait(bridgeMutex); // ورود مجدد به بخش حساس
    }

    // بررسی تعداد ماشین‌های در حال عبور از سمت شمال
    wait(limitSem);

    // عبور از پل به سمت شمال
    northCount++;
    signal(bridgeMutex); // خروج از بخش حساس

    // ادامه ترافیک به سمت شمال
    signal(northSem); // آزاد کردن سمافور ترافیک به سمت شمال
    signal(limitSem);  // آزاد کردن یک جایگاه از سمافور محدودیت
    flag = 0;      // درخواست ها تغییری نکرده اند
}
```

```
// تابع برای ماشین‌های در حال حرکت به سمت جنوب
void south() {
    wait(southSem); // منتظر اجازه عبور از سمت جنوب باشید

    wait(bridgeMutex); // ورود به بخش حساس

    // بررسی اینکه آیا ماشین‌های در حال حرکت به سمت شمال پل را عبور کرده‌اند یا خیر
    if (northCount > 0) {
        flag = 1;      // تغییر وضعیت درخواست ها
        signal(southSem); // آزاد کردن سمافور ترافیک به سمت جنوب
        signal(bridgeMutex); // خروج از بخش حساس
        wait(northSem); // منتظر اجازه عبور از سمت شمال باشید
        wait(bridgeMutex); // ورود مجدد به بخش حساس
    }

    // بررسی تعداد ماشین‌های در حال عبور از سمت جنوب
    wait(limitSem);

    // عبور از پل به سمت جنوب
    southCount++;
    signal(bridgeMutex); // خروج از بخش حساس

    // ادامه ترافیک به سمت جنوب
    signal(southSem); // آزاد کردن سمافور ترافیک به سمت جنوب
    signal(limitSem);  // آزاد کردن یک جایگاه از سمافور محدودیت
    flag = 0;      // درخواست ها تغییری نکرده اند

}
```

## سوال 4:

```
semaphore s1 = 1; // سمافور برای مدیریت ورود به غار
semaphore s2 = 15; // سمافور برای مدیریت تعداد بازدیدکنندگان درون غار

Cave_exploration() {
    wait(s2); // چک کردن تعداد بازدیدکنندگان حاضر در غار
    wait(s1); // درخواست ورود به غار

    // ورود به غار
    Enter_the_cave();

    signal(s1); // اجازه ورود به یک بازدیدکننده دیگر
    wait(s2); // چک کردن تعداد بازدیدکنندگان درون غار
    signal(s2); // افزایش تعداد بازدیدکنندگان درون غار

    // مشاهده نقاشی‌ها
    Look_at_the_paintings();

    wait(s1); // درخواست خروج از غار
    wait(s2); // چک کردن تعداد بازدیدکنندگان درون غار

    // خروج از غار
    Exit_the_cave();

    signal(s2); // کاهش تعداد بازدیدکنندگان درون غار
    signal(s1); // اجازه خروج به بازدیدکننده دیگر
}
```