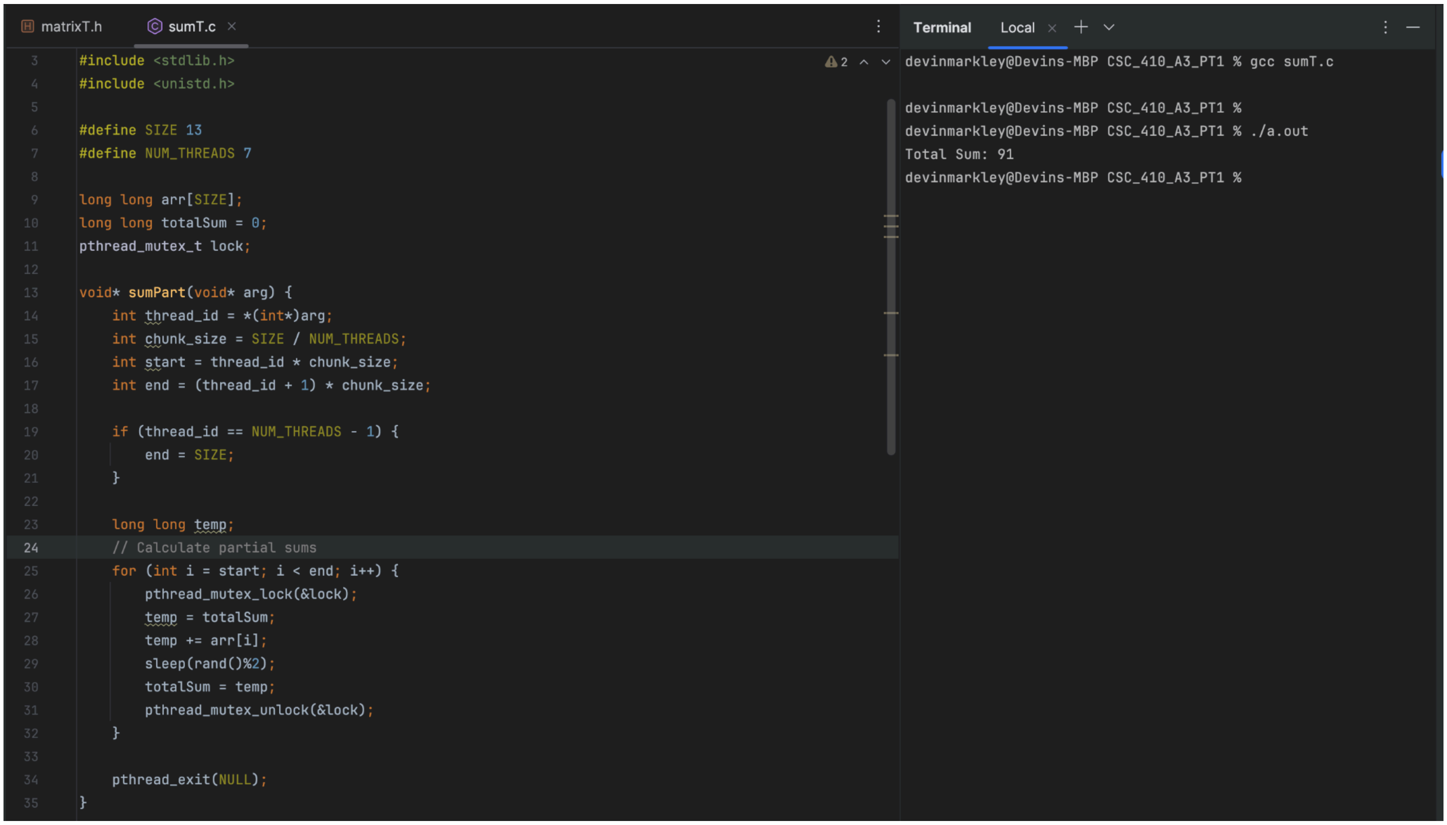
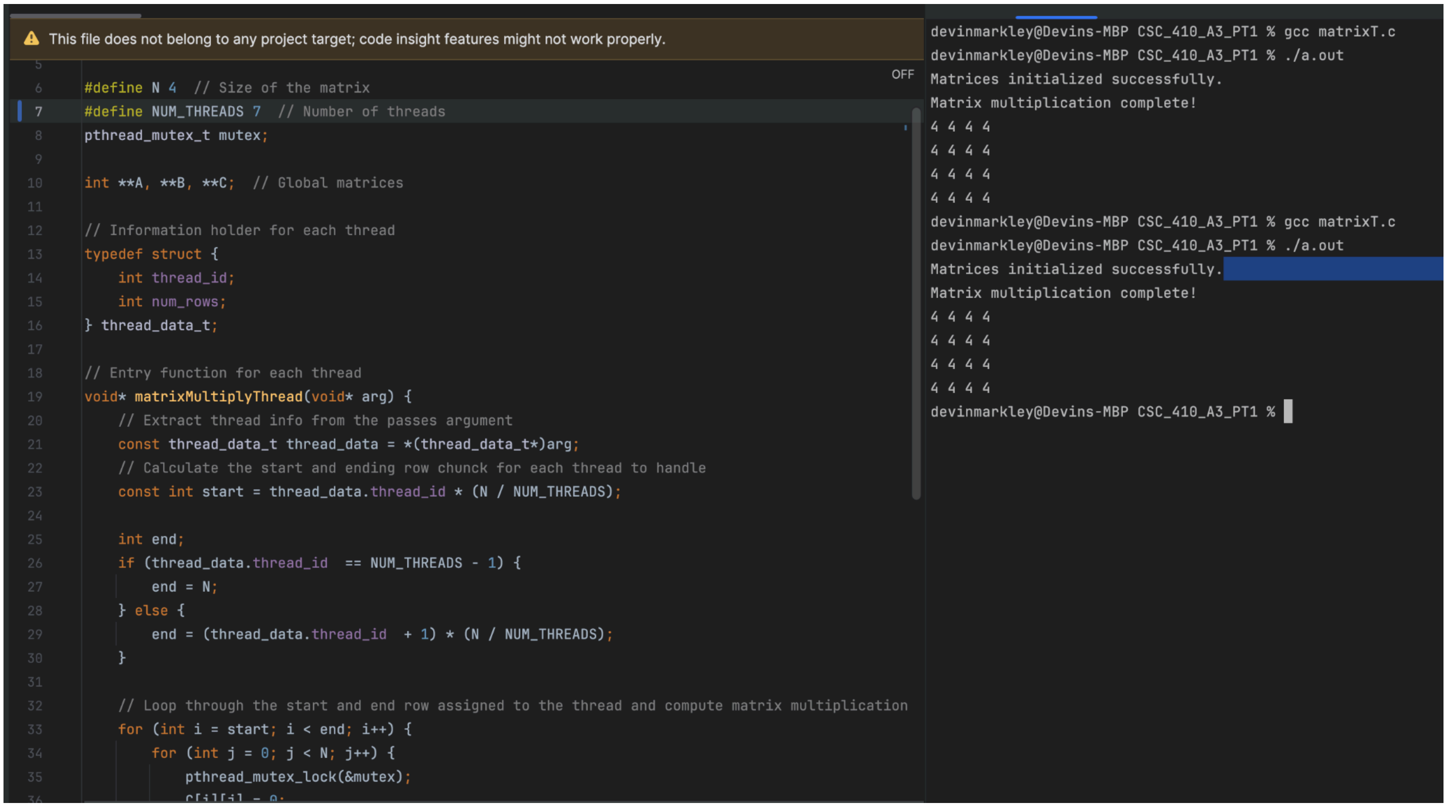
Task: 1:



```c
#include <stdlib.h>
#include <unistd.h>

#define SIZE 13
#define NUM_THREADS 7

long long arr[SIZE];
long long totalSum = 0;
pthread_mutex_t lock;

void* sumPart(void* arg) {
    int thread_id = *(int*)arg;
    int chunk_size = SIZE / NUM_THREADS;
    int start = thread_id * chunk_size;
    int end = (thread_id + 1) * chunk_size;

    if (thread_id == NUM_THREADS - 1) {
        end = SIZE;
    }

    long long temp;
    // Calculate partial sums
    for (int i = start; i < end; i++) {
        pthread_mutex_lock(&lock);
        temp = totalSum;
        temp += arr[i];
        sleep(rand()%2);
        totalSum = temp;
        pthread_mutex_unlock(&lock);
    }

    pthread_exit(NULL);
}
```

```
devinmarkley@Devins-MBP CSC_410_A3_PT1 % gcc sumT.c

devinmarkley@Devins-MBP CSC_410_A3_PT1 %
devinmarkley@Devins-MBP CSC_410_A3_PT1 % ./a.out
Total Sum: 91
devinmarkley@Devins-MBP CSC_410_A3_PT1 %
```

Task 2:



This file does not belong to any project target; code insight features might not work properly.

```c
#define N 4  // Size of the matrix
#define NUM_THREADS 7  // Number of threads
pthread_mutex_t mutex;

int **A, **B, **C;  // Global matrices

// Information holder for each thread
typedef struct {
    int thread_id;
    int num_rows;
} thread_data_t;

// Entry function for each thread
void* matrixMultiplyThread(void* arg) {
    // Extract thread info from the passes argument
    const thread_data_t thread_data = *(thread_data_t*)arg;
    // Calculate the start and ending row chunck for each thread to handle
    const int start = thread_data.thread_id * (N / NUM_THREADS);

    int end;
    if (thread_data.thread_id  == NUM_THREADS - 1) {
        end = N;
    } else {
        end = (thread_data.thread_id  + 1) * (N / NUM_THREADS);
    }

    // Loop through the start and end row assigned to the thread and compute matrix multiplication
    for (int i = start; i < end; i++) {
        for (int j = 0; j < N; j++) {
            pthread_mutex_lock(&mutex);
            C[i][j] = 0;
```

```
devinmarkley@Devins-MBP CSC_410_A3_PT1 % gcc matrixT.c
devinmarkley@Devins-MBP CSC_410_A3_PT1 % ./a.out
Matrices initialized successfully.
Matrix multiplication complete!
4 4 4 4
4 4 4 4
4 4 4 4
4 4 4 4
devinmarkley@Devins-MBP CSC_410_A3_PT1 % gcc matrixT.c
devinmarkley@Devins-MBP CSC_410_A3_PT1 % ./a.out
Matrices initialized successfully.
Matrix multiplication complete!
4 4 4 4
4 4 4 4
4 4 4 4
4 4 4 4
devinmarkley@Devins-MBP CSC_410_A3_PT1 %
```

Task 3:

For tasks 1 and 2, I utilized mutex to prevent more than one thread from accessing my global variables at a time. A mutex ensures that only one thread can access a shared resource at a time, avoiding data races and ensuring thread safety. I was sure that if the chunks couldn't be evenly distributed by ensuring that the last thread to be created would have an end equalling the input size.