```c
// Jessica Elkins
// Lab 12 - CS332
// 4/21/20
// BlazerID: jelkins3
// This program is a modification of the pthread_sum.c program.
// Instead of using global variables, this program passes an instance
// of a structure as an argument to the threads.

// TO COMPILE: gcc lab12.c -o lab12 -lpthread
// TO RUN: ./lab12 <# of elements> <# of threads>

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
#include <string.h>

typedef struct var{
        double *a;
        double sum;
        int N;
        int nthreads;
        int tid;
        pthread_t ptid;
} VAR;

void *compute(void *args){
        int myStart, myEnd, myN, i;
        VAR *info = (VAR *)args;

        int N = info->N;
        int tid = info->tid;
        int size = info->nthreads;
        //printf("tid = %d \n", tid);

        //determine start and end of computation for the current thread
        myN = N/size;
        myStart = tid*myN;
        myEnd = myStart + myN;

        //printf("start = %d  end = %d \n", myStart, myEnd);

        if(tid == (size-1))
                myEnd = N;

        //for(i = 0; i < N; i++){
        //      printf("a[%d] = %d\n", i, info->a[i]);
        //}

        // compute partial sum
        for(i = myStart; i < myEnd; i++){
                info->sum += info->a[i];
        }

        //printf("mysum: %g \n", info->sum);

        return (NULL);
}

int main(int argc, char **argv){
        VAR *info;
        pthread_t *tid;
        long i;
        int N, nthreads;
        double *a = NULL;

        if(argc != 3){
                printf("Usage: %s <# of elements> <# of threads> \n", argv[0]);
                exit(-1);
        }

        // getting the command line arguments
        N = atoi(argv[1]); // no. of elements
        nthreads = atoi(argv[2]); // no. of threads

        info = (VAR *)malloc(sizeof(VAR)*nthreads);
        a = (double *)malloc(sizeof(double)*N);

        long j;

        // filling up the array
        for(j = 0; j < nthreads; j++){
                info[j].a = (double *)malloc(sizeof(double)*N);
                for(i = 0; i < N; i++){
                        a[i] = (double)(i + 1);
                        info[j].a[i] = a[i];
                }
        }

        // creating the threads
        for(i = 0; i < nthreads; i++){
                info[i].tid = i;
                info[i].N = N;
                info[i].sum = 0.0;
                info[i].nthreads = nthreads;
                pthread_create(&info[i].ptid, NULL, compute, (void *)&info[i]);
        }

        //wait for them to complete
        for(i = 0; i < nthreads; i++){
                pthread_join(info[i].ptid, NULL);
        }

        // calculating the result
        double totalSum = 0.0;
        for(i = 0; i < nthreads; i++){
                totalSum += info[i].sum;
        }

        // printing results
        double shouldBe = ((double)N*(N+1)) / 2;
        printf("The total is %g, it should be equal to %g \n", totalSum, shouldBe);

        // freeing the malloced variables
        free(info);
        free(a);


        return 0;
}
```