```c
/**
 * Homework 2: Parallel, binary max function
 *
 * CS4414 Operating Systems
 * Fall 2017
 *
 * Maurice Wong - mzw7af
 *
 * main.c - max function program
 *
 * COMPILE:     make
 * OBJECTS:     main.o
 * RUN:         ./max
 */

#include "barrier.h"
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

int readNums(int* nums, int len);
void* max(void *v);

/**
 * Args struct for thread function
 */
typedef struct argstruct {
    int tid;
    int numThreads;
    int *nums;
    barrier *b;
    int result;
} argstruct;

int main() {
    // read in nums from stdin
    int len = 8092;
    int* nums = malloc(len * sizeof(int));
    int numCount = readNums(nums, len);

    // create threads
    int numThreads = numCount / 2;
    pthread_t tids[numThreads];
    argstruct args[numThreads];

    pthread_attr_t attr;
    pthread_attr_init(&attr);
    int i;

    barrier b;
    initBarrier(&b, numThreads);
    // setup argstructs:
    for (i = 0; i < numThreads; i++) {
        args[i].tid = i;
```

```
        args[i].nums = nums;
        args[i].numThreads = numThreads;
        args[i].b = &b;
    }
    for (i = 0; i < numThreads; i++) {
        pthread_create(&tids[i], &attr, max, &args[i]);
    }

    for (i = 0; i < numThreads; i++) {
        pthread_join(tids[i], NULL);
    }
    printf("%d\n", args[0].result);
    free(nums);
    return 0;
}

/**
 * Reads in numbers from stdin into nums until newline
 * returns the # of numbers read in
 */
int readNums(int *nums, int len) {
    char line[1024];
    int numCount = 0;
    while (fgets(line, 1024, stdin) != NULL && line[0] != '\n') {
        if (numCount == len) {
            len *= 2;
            nums = realloc(nums, len * sizeof(int));
        }
        sscanf(line, "%d", &nums[numCount]);
        numCount++;
    }
    return numCount;
}

/**
 * Thread function to calculate the max of two numbers:
 *
 * In the nums array, 2 spots are designated as input for thread i:
 * 2*i and 2*i + 1
 * The upper half of threads is rendered inactive on each round.
 * Active threads will write their result to 2*i to continue with.
 * Inactive threads will write their result to 2*i - n + 1,
 * where n = num of active threads. This places the next round of active numbers
 * into the lower half of nums.
 */
void* max(void *v) {
    argstruct *a = (argstruct *) v;
    int threadCount = a->numThreads;
    int firstNum, secondNum, max;
    while (threadCount >= 1) {
        int tid = a->tid;
        if (tid < threadCount) {
            // thread is active, compare max
            firstNum = a->nums[tid * 2];
```

```
            secondNum = a->nums[(tid * 2) + 1];
            max = firstNum > secondNum ? firstNum : secondNum;
            waitBarrier(a->b);
            if (tid < threadCount >> 1) {
                // thread active on next round
                a->nums[2 * tid] = max;
            } else {
                // thread is inactive on next round
                a->nums[(2*tid) - threadCount + 1] = max;
            }
            waitBarrier(a->b);
        } else {
            // thread is inactive, wait barrier twice
            waitBarrier(a->b);
            waitBarrier(a->b);
        }
        threadCount = threadCount >> 1;
    }
    a->result = max;
    pthread_exit(NULL);
}
```