FILE:
circuit.c

PURPOSE:
Determine circuit satisfiability for 16 inputs using both static and dynamic scheduling in openmp.

ALOGIRTHMS/LIBRARIES:
```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
```

The standard libraries are used for obvious standard functionality. Omp.h is included for threading.

The algorithm used is simply setting up the paralleized for loop using openmp and then calling Dr. Karlsson's check_circuit function to do all the hard work.

FUNCTIONS/STRUCTURE:
Main() handles the command line argument of thread_count and then begins timing immediately before calling the parallel function, parallel_static().

Parallel_static() uses openmp to set up a for loop from 0 to input_maximum (65535 for 16 inputs) and keeps track of how many solutions Dr. Karlsson's check_check_circuit() function finds.

Control returns to main(), which stops the timing and prints the total results. It then does the same process for parallel_dynamic() that it did for parallel_static(). Before exiting, it compares the times to see which scheduling method was faster and by how much.

COMPILE/RUN:
With the makefile in the directory, simply use
```
make circuit
```

Without the makefile, you can use
```
gcc -g -Wall -fopenmp -o circuit circuit.c -lm
```

Run using
```
./circuit <n>
```
where n is the number of threads, and is required

You can also use
```
make circuit_debug
```

which just compiles with -DDEBUG and takes no command line argument for number of processors.

TESTING:
By testing multiple consecutive runs on the same number of processors, it seems like for my machine, static is almost always faster.

Below is the output for tests on different numbers of processors, with 10 runs per processor number. You can reproduce this test by using make circuit_debug (which just compiles with -DDEBUG).

```
2 processors:
Static was faster by 1.106302e-03
Static was faster by 8.685370e-04
Static was faster by 9.397050e-04
Static was faster by 8.301460e-04
Static was faster by 1.189263e-03
Static was faster by 8.597920e-04
Static was faster by 9.549920e-04
Static was faster by 6.863990e-04
Static was faster by 9.199900e-04
Static was faster by 9.674460e-04

4 processors:
Static was faster by 6.181000e-06
Static was faster by 1.202545e-03
Static was faster by 8.086530e-04
Static was faster by 8.099240e-04
Static was faster by 1.066740e-03
Static was faster by 7.734280e-04
Static was faster by 7.798050e-04
Static was faster by 8.342110e-04
Static was faster by 7.725160e-04
Dynamic was faster by 6.626250e-04

8 processors:
Static was faster by 3.810940e-04
Static was faster by 6.144080e-04
Static was faster by 4.844400e-04
Static was faster by 4.707160e-04
Static was faster by 4.627900e-04
Static was faster by 5.044640e-04
Static was faster by 1.836460e-04
Static was faster by 2.146620e-04
Static was faster by 6.289460e-04
Dynamic was faster by 2.420450e-04

16 processors:
Dynamic was faster by 3.314470e-04
Static was faster by 1.347270e-04
Static was faster by 8.956540e-04
Static was faster by 1.310840e-04
Static was faster by 3.832200e-04
Static was faster by 2.441150e-04
Static was faster by 4.369570e-04
Static was faster by 2.275500e-04
Static was faster by 6.408130e-04
Static was faster by 7.207320e-04
```

```
32 processors:
Dynamic was faster by 1.069279e-03
Static was faster by 1.937460e-04
Static was faster by 5.334350e-04
Static was faster by 7.335900e-04
Dynamic was faster by 3.078500e-05
Static was faster by 7.198240e-04
Dynamic was faster by 3.704890e-04
Static was faster by 4.387740e-04
Static was faster by 1.022085e-03
Static was faster by 6.608100e-04

64 processors:
Dynamic was faster by 9.338060e-04
Static was faster by 3.257620e-04
Static was faster by 2.299290e-04
Static was faster by 2.335150e-04
Static was faster by 5.761610e-04
Static was faster by 8.135580e-04
Static was faster by 5.575330e-04
Dynamic was faster by 2.046500e-04
Static was faster by 3.665260e-04
Static was faster by 9.731090e-04
```

SUBMITTED:
circuit.c
Makefile
(in addition to the file for the other part)

ADDITIONAL NOTES:

FILE:
prime.c

PURPOSE:
List all prime numbers less than given number n using a C implementation of Eratosthenes' sieve.

ALOGIRTHMS/LIBRARIES:
```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
```

The standard libraries are used for obvious standard functionality. Omp.h is included for threading.

The algorithm used is just a simple if-else implementation of the Eratosthenes' sieve's English description. An array of size user_maximum (the number provided at the command line) is declared and set to all 0s, which signifies a number is prime. Then the program sets all non-prime indices to 1 and prints the primes by seeing which indices are still 0 after the sieve.

FUNCTIONS/STRUCTURE:
Main() handles the the command line argument of n (putting it into a variable user_maximum), which is the number greater than all the prime numbers to be generated. It then allocates a marked[] integer array of size user_maximum sets all indices to all 0s, which signifies a number is prime, and saves the start time before calling parallel_static().

Parallel_static() sets up a for loop that cycles from 2 to input_maximum – 1, setting all the non-prime indices in marked[] to 1 by marking all the multiples of a given i. This loop breaks down these tasks into the find_next_unmarked() function, which returns the next index greater than i that is 0, and the mark_multiples() function, which marks all multiples of i.

Control returns to main(), which stops the timing and calls print_primes() to loop through the list and print all indices with a value of 0, and then prints the timing information. It subsequently does the same process for parallel_dynamic() that it did for parallel_static(). Before exiting, it compares the times to see which scheduling method was faster and by how much.

COMPILE/RUN:
With the makefile in the directory, simply use

```
make prime
```

Without the makefile, you can use

```
gcc -g -Wall -fopenmp -o prime prime.c -lm
```

Run using

```
./prime <n>
```

where all primes less than n are desired, and n is required

You can also use

```
make prime_debug
```

which just compiles with -DDEBUG and takes no command line argument for n. It produces a compiler warning that the variable global_count is set but not used, but this is expected, since DEBUG doesn't print the count. Rewriting a separate debug function to avoid this warning would have produced a lot of redundant code, so I figured having the warning was the lesser of two evils.

TESTING:
Testing to see if the correct primes were generated was done first by checking against the output provided on the assignment description and lists on the Internet for other numbers.

For static and dynamic timing, you can make prime_debug to reproduce the tests below. Though there are some outliers it looks like lower numbers are generally faster with dynamic, while higher numbers (1000 and up) are almost 100% static.

n = 10:

```
Dynamic was faster by 1.981650e-04
Static was faster by 1.930000e-06
Dynamic was faster by 3.167000e-06
Dynamic was faster by 1.759000e-06
Dynamic was faster by 1.610400e-05
Dynamic was faster by 2.257000e-06
Dynamic was faster by 2.209000e-06
Dynamic was faster by 3.400005e-07
Static was faster by 1.077000e-06
Dynamic was faster by 2.390002e-07

n = 100:
Dynamic was faster by 4.960000e-06
Static was faster by 1.041999e-06
Dynamic was faster by 2.479300e-05
Dynamic was faster by 2.406900e-05
Dynamic was faster by 2.018700e-05
Dynamic was faster by 1.108800e-05
Dynamic was faster by 2.427000e-05
Dynamic was faster by 1.884500e-05
Dynamic was faster by 2.246800e-05
Dynamic was faster by 2.783800e-05

n = 1000:
Static was faster by 2.734000e-05
Static was faster by 1.772700e-05
Dynamic was faster by 9.532000e-06
Static was faster by 1.230600e-05
Static was faster by 4.268600e-05
Static was faster by 1.109400e-05
Static was faster by 2.840600e-05
Static was faster by 4.088800e-05
Static was faster by 2.612200e-05
Static was faster by 2.808600e-05

n = 10000:
Static was faster by 4.574150e-04
Static was faster by 4.450110e-04
Static was faster by 1.078509e-03
Dynamic was faster by 1.368976e-03
Static was faster by 2.662740e-04
Static was faster by 1.626800e-05
Static was faster by 3.147340e-04
Static was faster by 1.841620e-04
Static was faster by 3.128540e-04
Static was faster by 2.572550e-04

n = 100000:
Static was faster by 1.684607e-03
Static was faster by 2.910397e-03
Static was faster by 2.903086e-03
Static was faster by 2.730038e-03
Static was faster by 2.456818e-03
Static was faster by 1.743630e-03
Static was faster by 1.690187e-03
Static was faster by 3.014931e-03
Static was faster by 2.349354e-03
Static was faster by 1.898024e-03
```

n = 1000000:

```
Static was faster by 2.563386e-02
Static was faster by 2.780908e-02
Static was faster by 2.256400e-02
Static was faster by 2.652600e-02
Static was faster by 2.303527e-02
Static was faster by 2.785637e-02
Static was faster by 2.602083e-02
Static was faster by 2.525656e-02
Static was faster by 2.621163e-02
Static was faster by 2.249186e-02
```

n = 10000000:

```
Dynamic was faster by 8.849524e-03
Static was faster by 5.743754e-02
Dynamic was faster by 1.653250e-02
Static was faster by 9.064785e-03
Static was faster by 3.464775e-02
Static was faster by 7.002858e-02
Static was faster by 4.593406e-02
Static was faster by 3.606871e-02
Static was faster by 6.835387e-02
Static was faster by 5.371574e-02
```

SUBMITTED:
prime.c
Makefile
(in addition to the file for the other part)

ADDITIONAL NOTES:
