`University of Wolverhampton
School of Mathematics and Computer Science

6CS005 High Performance Computing Week 6 Workshop

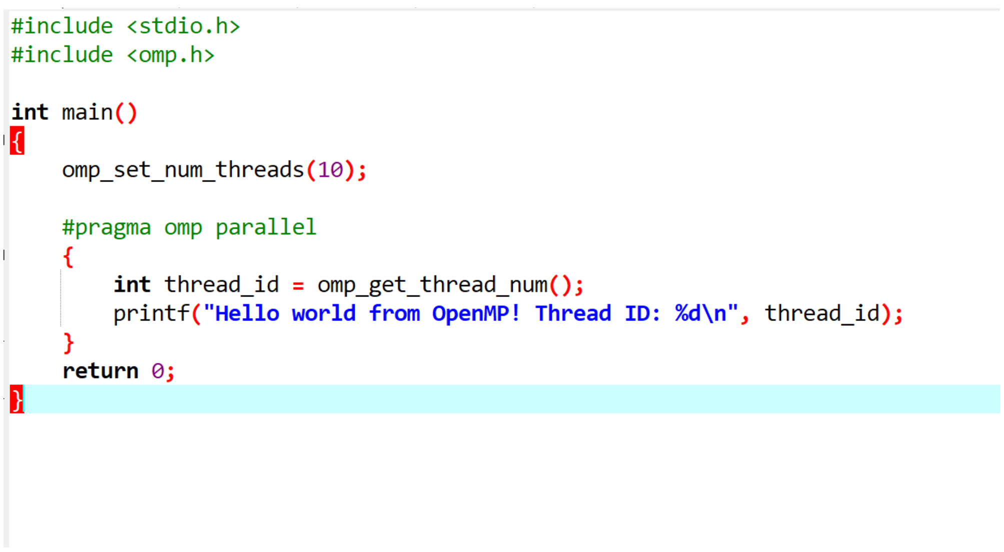
Tasks - OpenMP Multithreading

You may need to refer to the Week 6 lecture slides in order to complete these tasks.

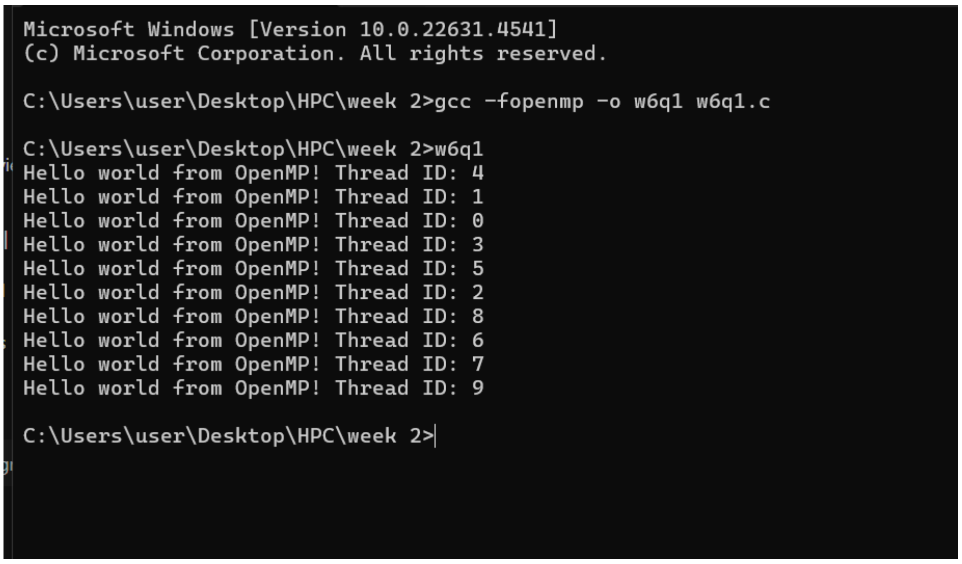
1. The following program prints out "Hello World!" with the default number of threads which is usually the number of CPU processor cores that are on your computer system:

```
#include <stdio.h>
void main()
{
  #pragma omp parallel
  printf("Hello world from OpenMP!\n");
}
```

   a. Enter and run the program to see how many processor cores are on the system you are using.
   b. Modify the program so that it run with exactly 10 threads.



```
#include <stdio.h>
#include <omp.h>

int main()
{
    omp_set_num_threads(10);

    #pragma omp parallel
    {
        int thread_id = omp_get_thread_num();
        printf("Hello world from OpenMP! Thread ID: %d\n", thread_id);
    }
    return 0;
}
```

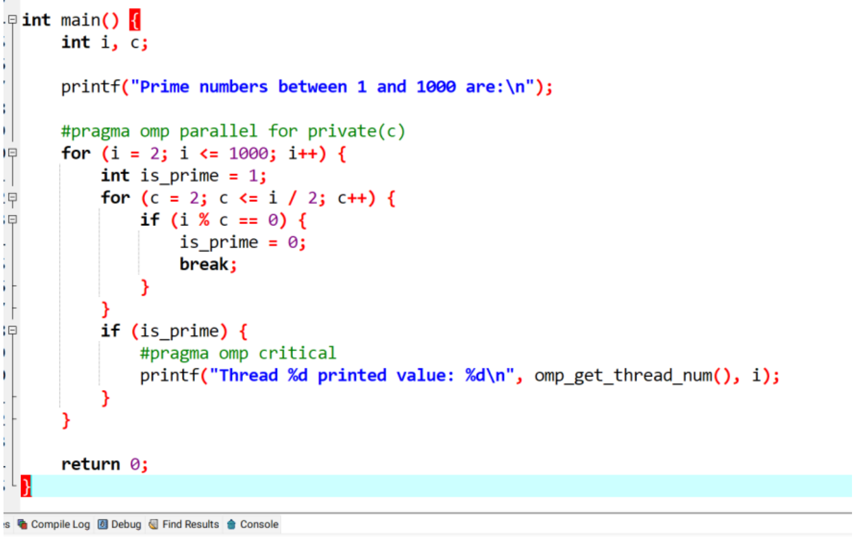


```
Microsoft Windows [Version 10.0.22631.4541]
(c) Microsoft Corporation. All rights reserved.

C:\Users\user\Desktop\HPC\week 2>gcc -fopenmp -o w6q1 w6q1.c

C:\Users\user\Desktop\HPC\week 2>w6q1
Hello world from OpenMP! Thread ID: 4
Hello world from OpenMP! Thread ID: 1
Hello world from OpenMP! Thread ID: 0
Hello world from OpenMP! Thread ID: 3
Hello world from OpenMP! Thread ID: 5
Hello world from OpenMP! Thread ID: 2
Hello world from OpenMP! Thread ID: 8
Hello world from OpenMP! Thread ID: 6
Hello world from OpenMP! Thread ID: 7
Hello world from OpenMP! Thread ID: 9

C:\Users\user\Desktop\HPC\week 2>
```

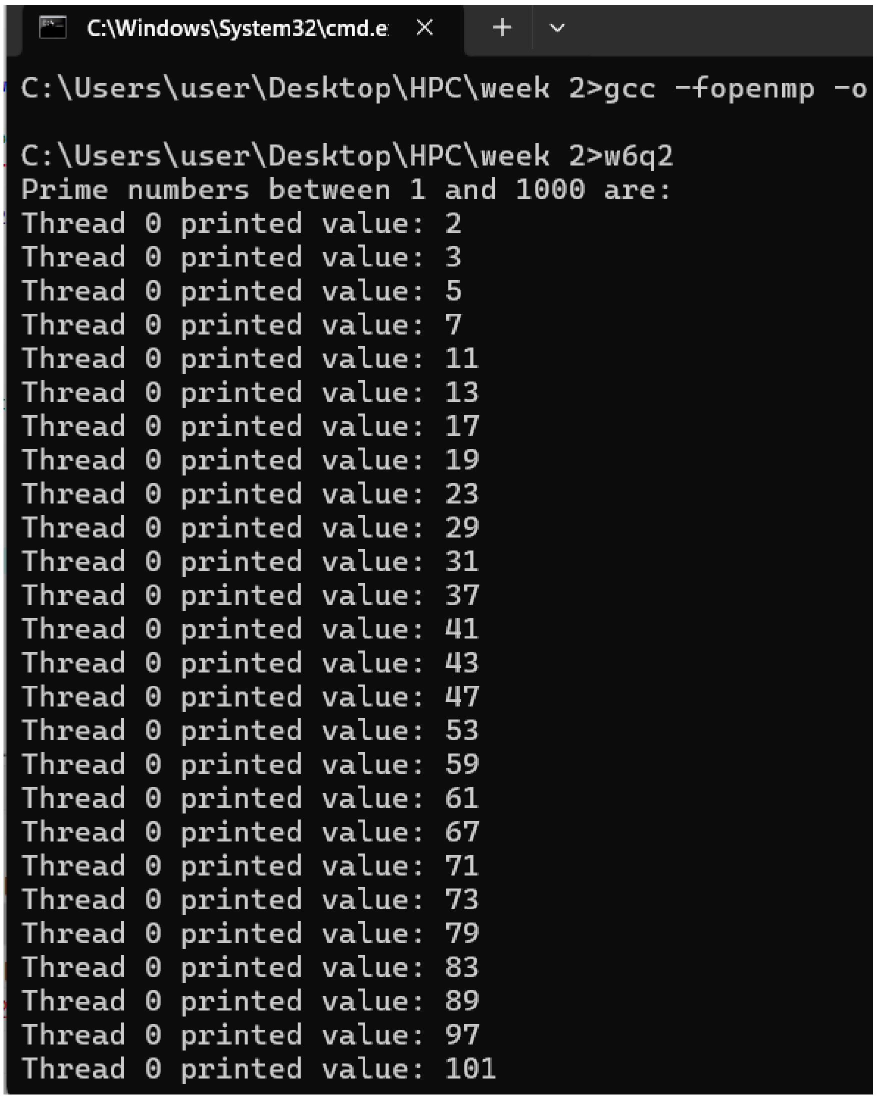
2. The following program prints out all the prime numbers from 1 to 1000:

```
#include <stdio.h>
void main()
{
  int i, c;
  printf("Prime numbers between 1 and 1000 are :\n");
  for(i = 1; i <= 1000; i++){
    for(c = 2; c <= i - 1; c++){
      if ( i % c == 0 )
        break;
    }
    if ( c == i )
      printf("%d\n", i);
  }
}
```

Convert this to a multithread OpenMP program using the default number of threads to calculate the prime numbers.

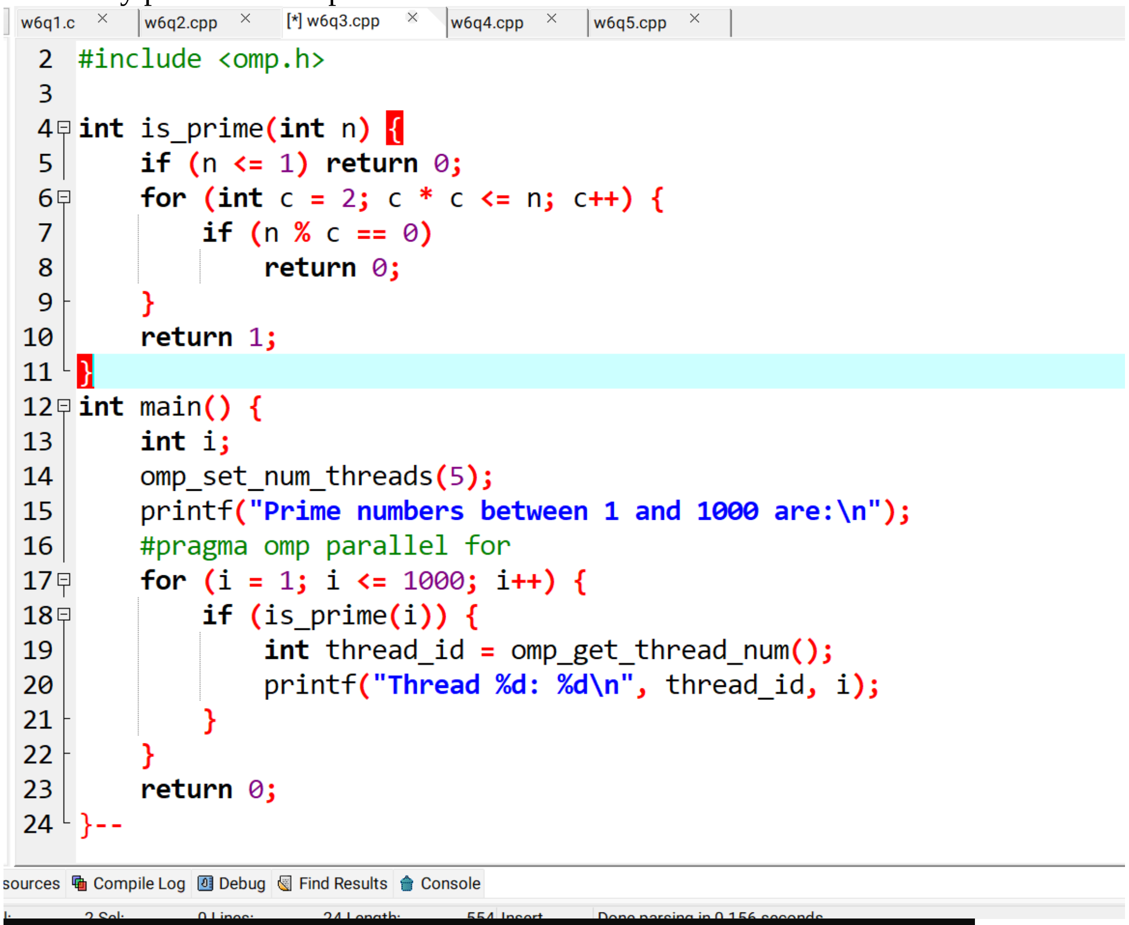


```
int main() {
    int i, c;

    printf("Prime numbers between 1 and 1000 are:\n");

    #pragma omp parallel for private(c)
    for (i = 2; i <= 1000; i++) {
        int is_prime = 1;
        for (c = 2; c <= i / 2; c++) {
            if (i % c == 0) {
                is_prime = 0;
                break;
            }
        }
        if (is_prime) {
            #pragma omp critical
            printf("Thread %d printed value: %d\n", omp_get_thread_num(), i);
        }
    }

    return 0;
}
```

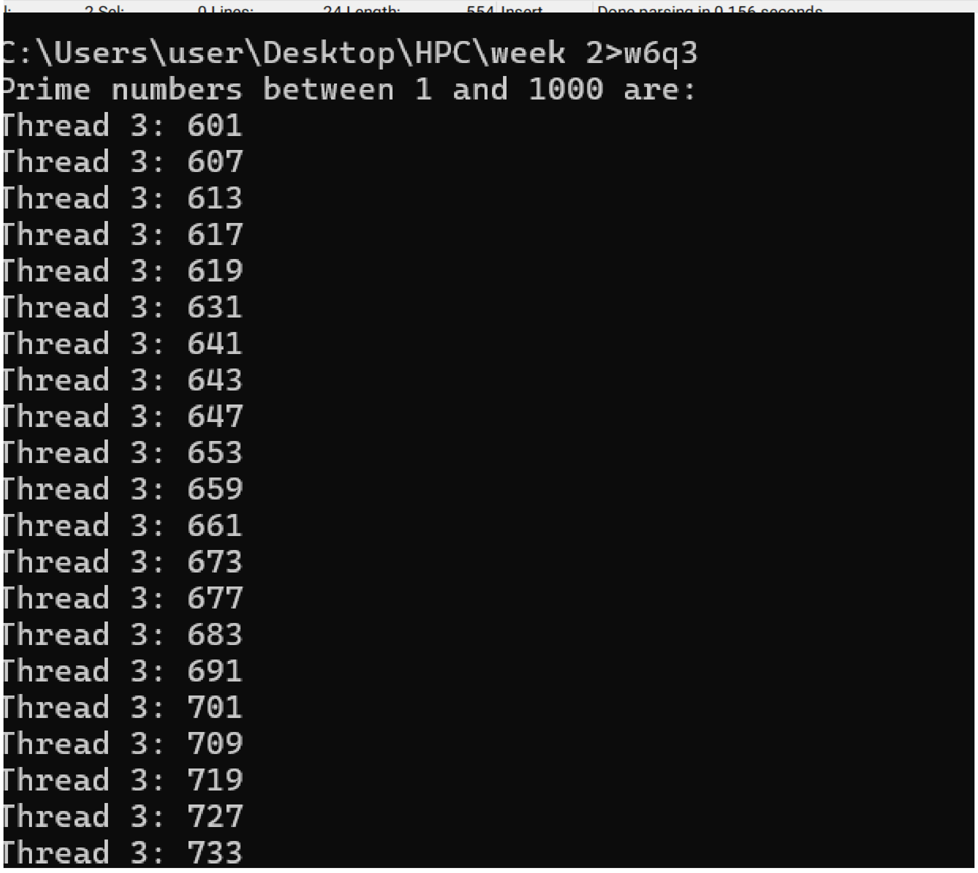


```
C:\Users\user\Desktop\HPC\week 2>gcc -fopenmp -o

C:\Users\user\Desktop\HPC\week 2>w6q2
Prime numbers between 1 and 1000 are:
Thread 0 printed value: 2
Thread 0 printed value: 3
Thread 0 printed value: 5
Thread 0 printed value: 7
Thread 0 printed value: 11
Thread 0 printed value: 13
Thread 0 printed value: 17
Thread 0 printed value: 19
Thread 0 printed value: 23
Thread 0 printed value: 29
Thread 0 printed value: 31
Thread 0 printed value: 37
Thread 0 printed value: 41
Thread 0 printed value: 43
Thread 0 printed value: 47
Thread 0 printed value: 53
Thread 0 printed value: 59
Thread 0 printed value: 61
Thread 0 printed value: 67
Thread 0 printed value: 71
Thread 0 printed value: 73
Thread 0 printed value: 79
Thread 0 printed value: 83
Thread 0 printed value: 89
Thread 0 printed value: 97
Thread 0 printed value: 101
```

3. Convert the above program so that it uses exactly 5 threads and each thread prints out their thread ID when they print out their prime number.

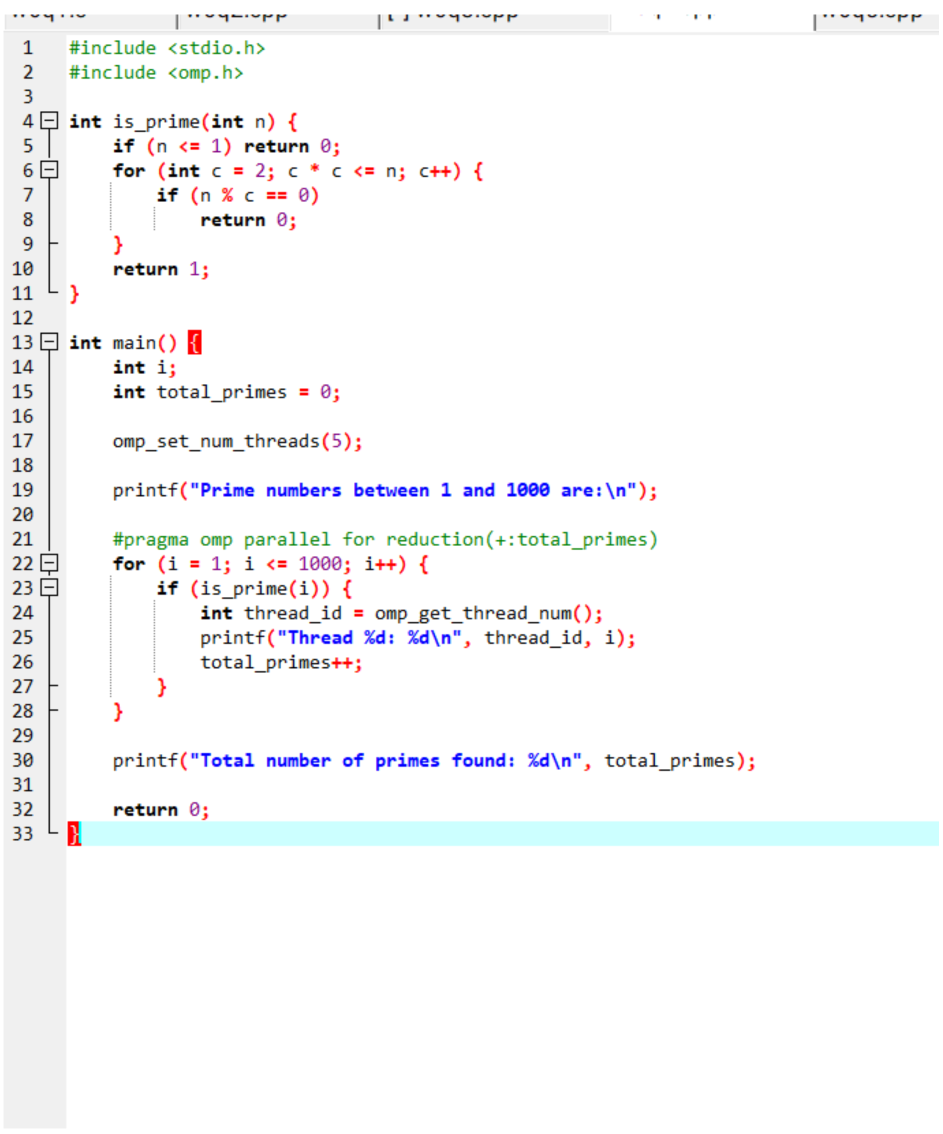


```c
#include <omp.h>

int is_prime(int n) {
    if (n <= 1) return 0;
    for (int c = 2; c * c <= n; c++) {
        if (n % c == 0)
            return 0;
    }
    return 1;
}
int main() {
    int i;
    omp_set_num_threads(5);
    printf("Prime numbers between 1 and 1000 are:\n");
    #pragma omp parallel for
    for (i = 1; i <= 1000; i++) {
        if (is_prime(i)) {
            int thread_id = omp_get_thread_num();
            printf("Thread %d: %d\n", thread_id, i);
        }
    }
    return 0;
}
```

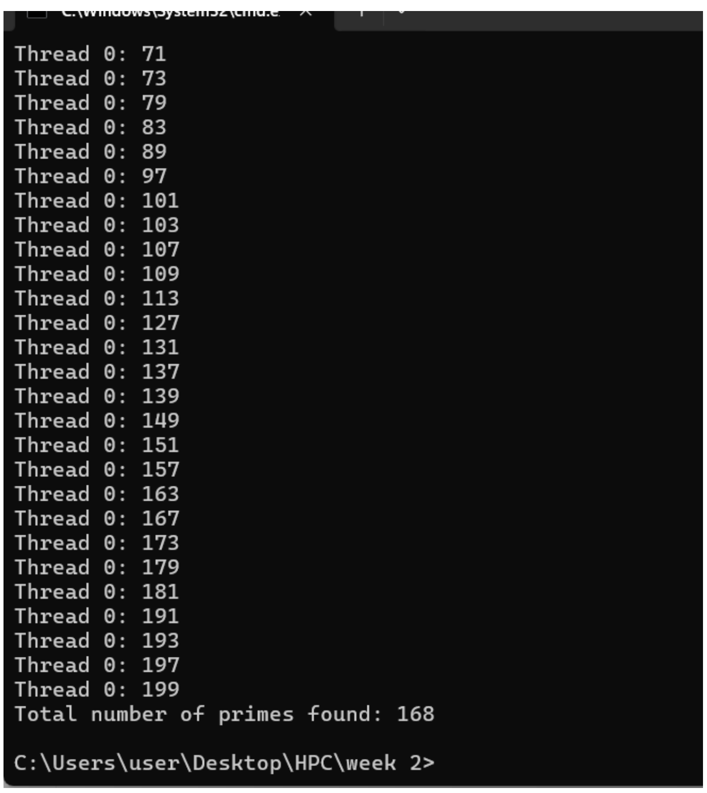


```
C:\Users\user\Desktop\HPC\week 2>w6q3
Prime numbers between 1 and 1000 are:
Thread 3: 601
Thread 3: 607
Thread 3: 613
Thread 3: 617
Thread 3: 619
Thread 3: 631
Thread 3: 641
Thread 3: 643
Thread 3: 647
Thread 3: 653
Thread 3: 659
Thread 3: 661
Thread 3: 673
Thread 3: 677
Thread 3: 683
Thread 3: 691
Thread 3: 701
Thread 3: 709
Thread 3: 719
Thread 3: 727
Thread 3: 733
```

4. Modify the program in (3) so that the program prints out the total number of prime numbers found. Check this whether this is correct by running it with exactly 1 thread.



```c
#include <stdio.h>
#include <omp.h>

int is_prime(int n) {
    if (n <= 1) return 0;
    for (int c = 2; c * c <= n; c++) {
        if (n % c == 0)
            return 0;
    }
    return 1;
}

int main() {
    int i;
    int total_primes = 0;

    omp_set_num_threads(5);

    printf("Prime numbers between 1 and 1000 are:\n");

    #pragma omp parallel for reduction(+:total_primes)
    for (i = 1; i <= 1000; i++) {
        if (is_prime(i)) {
            int thread_id = omp_get_thread_num();
            printf("Thread %d: %d\n", thread_id, i);
            total_primes++;
        }
    }

    printf("Total number of primes found: %d\n", total_primes);

    return 0;
}
```



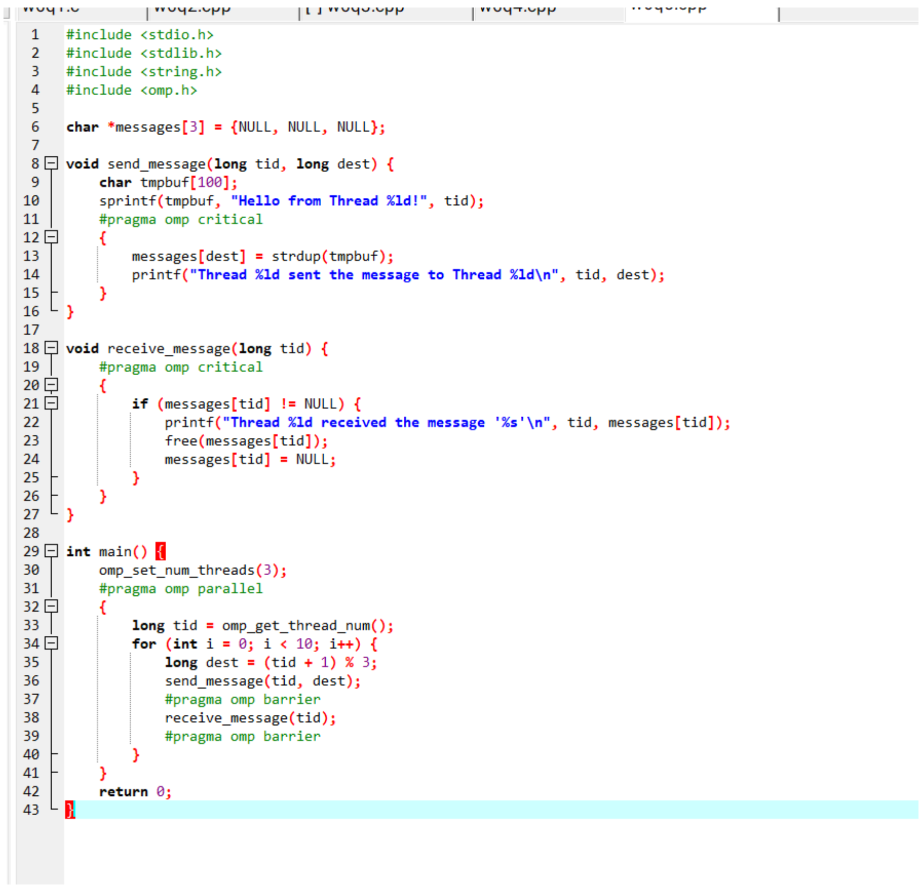
```
Thread 0: 71
Thread 0: 73
Thread 0: 79
Thread 0: 83
Thread 0: 89
Thread 0: 97
Thread 0: 101
Thread 0: 103
Thread 0: 107
Thread 0: 109
Thread 0: 113
Thread 0: 127
Thread 0: 131
Thread 0: 137
Thread 0: 139
Thread 0: 149
Thread 0: 151
Thread 0: 157
Thread 0: 163
Thread 0: 167
Thread 0: 173
Thread 0: 179
Thread 0: 181
Thread 0: 191
Thread 0: 193
Thread 0: 197
Thread 0: 199
Total number of primes found: 168

C:\Users\user\Desktop\HPC\week 2>
```

5. The following PThreads program demonstrates 3 thread sending string messages to each other, using a global array. The messages are meant to be sent in the following order:

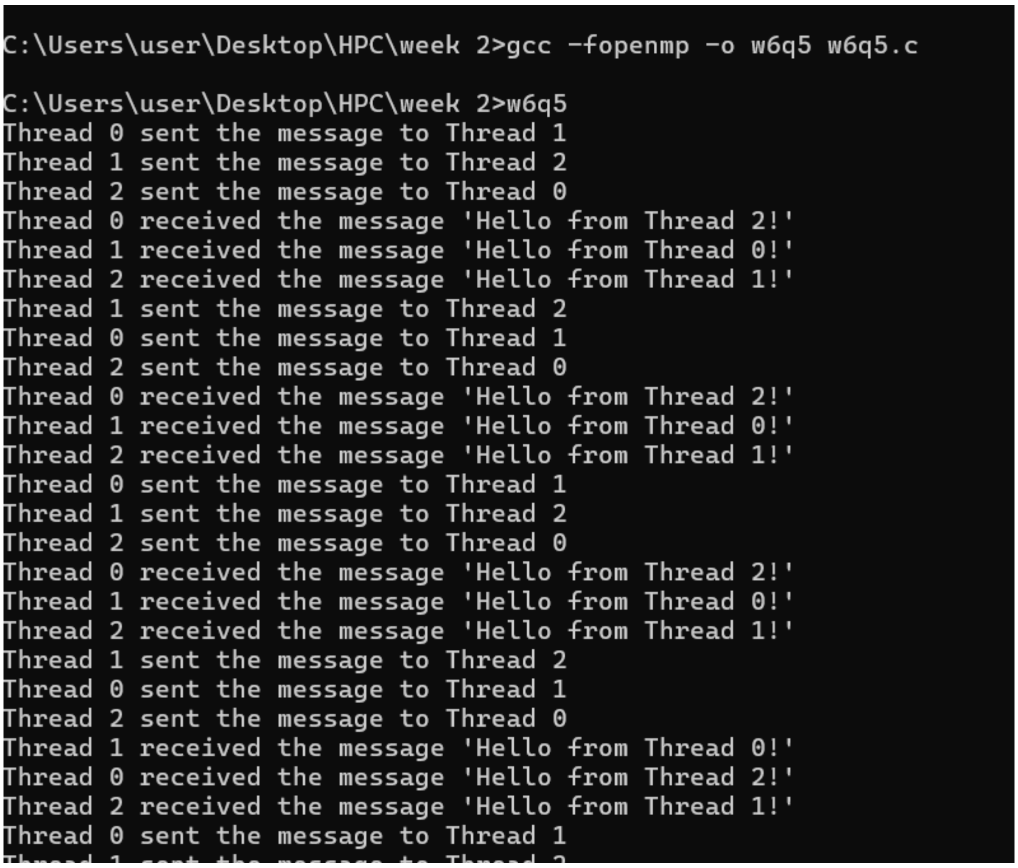
   a. Thread 0 sends Thread 1 a message
   b. Thread 1 receives the message
   c. Thread 1 sends Thread 2 a message
   d. Thread 2 receives the message
   e. Thread 2 sends Thread 0 a message
   f. Thread 0 receives the message
   g. This then repeats from (a) 10 times

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

char *messages[3] = {NULL, NULL, NULL};

void *messenger(void *p)
{
  long tid = (long)p;
  char tmpbuf[100];

  for(int i=0; i<10; i++)
  {
    /* Sending a message */
    long int dest = (tid + 1) % 3;
    sprintf(tmpbuf,"Hello from Thread %ld!", tid);
    char *msg = strdup(tmpbuf);
    messages[dest] = msg;
    printf("Thread %ld sent the message to Thread %ld\n",tid, dest);

    /* Receiving a message */
    printf("Thread %ld received the message '%s'\n",tid, messages[tid]);
    free(messages[tid]);
    messages[tid] = NULL;
  }
  return NULL;
}

void main()
{
  pthread_t thrID1, thrID2, thrID3;

  pthread_create(&thrID1, NULL, messenger, (void *)0);
  pthread_create(&thrID2, NULL, messenger, (void *)1);
  pthread_create(&thrID3, NULL, messenger, (void *)2);
  pthread_join(thrID1, NULL);
  pthread_join(thrID2, NULL);
  pthread_join(thrID3, NULL);
}
```

Convert the program to using OpenMP instead of Pthreads.



```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <omp.h>

char *messages[3] = {NULL, NULL, NULL};

void send_message(long tid, long dest) {
    char tmpbuf[100];
    sprintf(tmpbuf, "Hello from Thread %ld!", tid);
    #pragma omp critical
    {
        messages[dest] = strdup(tmpbuf);
        printf("Thread %ld sent the message to Thread %ld\n", tid, dest);
    }
}

void receive_message(long tid) {
    #pragma omp critical
    {
        if (messages[tid] != NULL) {
            printf("Thread %ld received the message '%s'\n", tid, messages[tid]);
            free(messages[tid]);
            messages[tid] = NULL;
        }
    }
}

int main() {
    omp_set_num_threads(3);
    #pragma omp parallel
    {
        long tid = omp_get_thread_num();
        for (int i = 0; i < 10; i++) {
            long dest = (tid + 1) % 3;
            send_message(tid, dest);
            #pragma omp barrier
            receive_message(tid);
            #pragma omp barrier
        }
    }
    return 0;
}
```



```
C:\Users\user\Desktop\HPC\week 2>gcc -fopenmp -o w6q5 w6q5.c

C:\Users\user\Desktop\HPC\week 2>w6q5
Thread 0 sent the message to Thread 1
Thread 1 sent the message to Thread 2
Thread 2 sent the message to Thread 0
Thread 0 received the message 'Hello from Thread 2!'
Thread 1 received the message 'Hello from Thread 0!'
Thread 2 received the message 'Hello from Thread 1!'
Thread 1 sent the message to Thread 2
Thread 0 sent the message to Thread 1
Thread 2 sent the message to Thread 0
Thread 0 received the message 'Hello from Thread 2!'
Thread 1 received the message 'Hello from Thread 0!'
Thread 2 received the message 'Hello from Thread 1!'
Thread 0 sent the message to Thread 1
Thread 1 sent the message to Thread 2
Thread 2 sent the message to Thread 0
Thread 0 received the message 'Hello from Thread 2!'
Thread 1 received the message 'Hello from Thread 0!'
Thread 2 received the message 'Hello from Thread 1!'
Thread 1 sent the message to Thread 2
Thread 0 sent the message to Thread 1
Thread 2 sent the message to Thread 0
Thread 1 received the message 'Hello from Thread 0!'
Thread 0 received the message 'Hello from Thread 2!'
Thread 2 received the message 'Hello from Thread 1!'
Thread 0 sent the message to Thread 1
```