The Assemblers
Spring 2019

Developing Soft and Parallel Programming Skills Using Project-Based Learning

---

Authors:
Aaja Christie
Davidson Fleurantin
Mamadou Diallo
Sheng Chen
Matthew Kabat

Planning and Scheduling

Assignee Name	Email (@student.gsu.edu)	Task	Duration (hours)	Dependency	Due Date (March 29)	Note
Aaja Christie	achristie3	Task 2 and the report	1	Slack GitHub	March 29	
Davidson Fleurantin	dfleurantin1	Task 4	5	Arm Programmig	March 24	
Mamadou Diallo (*Coordinator*)	mdiallo15	Task 3a	6	Introduction to parallel program 4 slides	March 28	Review answers before submitting
Sheng Chen	schen36	Task 5	1	Edit video	March 29, 2019	None
Matthew Kabat	mkabat1	Task 3b	6	Have a raspberry pi	March 29	

Task 3a.
Foundation

1. What is race condition?
    a. Race condition is the behavior of a system where the output is affected by another event whose timing, or sequence is uncontrollable. If the events doesn't go in the order they are intended to, this becomes a bug that may not be apparent and easy to debug.
2. Why race condition is difficult to reproduce and debug?
    a. A race condition is difficult to reproduce as its end result is undetermined and it is dependent on the timing between threads.
    b. During debug, the problems created or faced tend to disappear which makes the whole process difficult, this is why it is a better option to create and work on a better and more efficient software design instead of trying to reproduce and debug a race condition later on.
3. How can it be fixed? Provide an example from your Project_A3
    a. The best thing to do is avoid race conditions, than to try to fix them as they are hard to debug.
4. Summarize the Parallel Programming Patterns section in the "Introduction to Parallel Computing_3.pdf" in your own words. (No more than 150 words)
   s

   This section talks about the two most important programming patterns used in parallel applications. They are Strategies and Concurrent Execution Mechanisms.
    a. The Strategies are composed of Implementation and Algorithm. Implementation determines how the tasks of the program are going to be processed while implementation patterns is more like how the pi operates, it determines how and when the program the tasks should be executed.
    b. The Concurrent Execution Mechanisms are composed of Process/Thread Control and Coordination. The thread control controls which thread will be processed and when. The Coordination mutually communication data so that the threads will be completed.
    c. The last programming pattern is hybrid computation which is the combination of Strategies a24nd the Concurrent Execution Mechanisms. This pattern uses both OpenMP and the Message Passing Interface also known as MPI.
5. Compare the following
    a. Collective Synchronization (barrier) with Collective communication (reduction)

i. Collective Synchronization doesn't not move on until all processes reach a sync point. It doesn't this by calling a barrier function.
ii. Collective Communication collects data from each of the processes and communicates that data in order to execute the task.

b. Master-worker with fork join
   i. Master-Worker is just like it sounds. There is a central node known as the "Master" that splits the task to the "Workers" and keep the results.
   ii. Fork Join is when a parent thread splits work to be computed on to separate threads, then joins the results from those smaller threads.

Dependency

1) Where can we find parallelism in programming?

   Parallelism can be found when there is a sequence of operations that are needed to be performed in order to get a result. That sequence of operation must deal with control, data, and system dependencies.

2) What is dependency and what are its types?

   a) A dependency is when an operation depends on its preceding operations to complete, and produce a result before the next operation can be performed.

3) When is a statement dependent and when it is dependent (provide two examples)?

   a) Two statements are dependent when their order of execution affects the computational outcome?

   b) Two statements are independent when their order of execution doesn't matter.

   Example: the two below statements doesn't dependent on the other

   int a = 10;

   int b = 20;

4) When can two statements be executed in parallel?

   a) Two statement can be executed in parallel if and only if they don't dependent on each other, no exceptions.

5) How can dependency be removed?

   a) Dependency can be removed by either removing statements or by rearranging them so that one is not dependent on the other.

6) How do we compute dependency for the following two loops and what types/s of dependency?

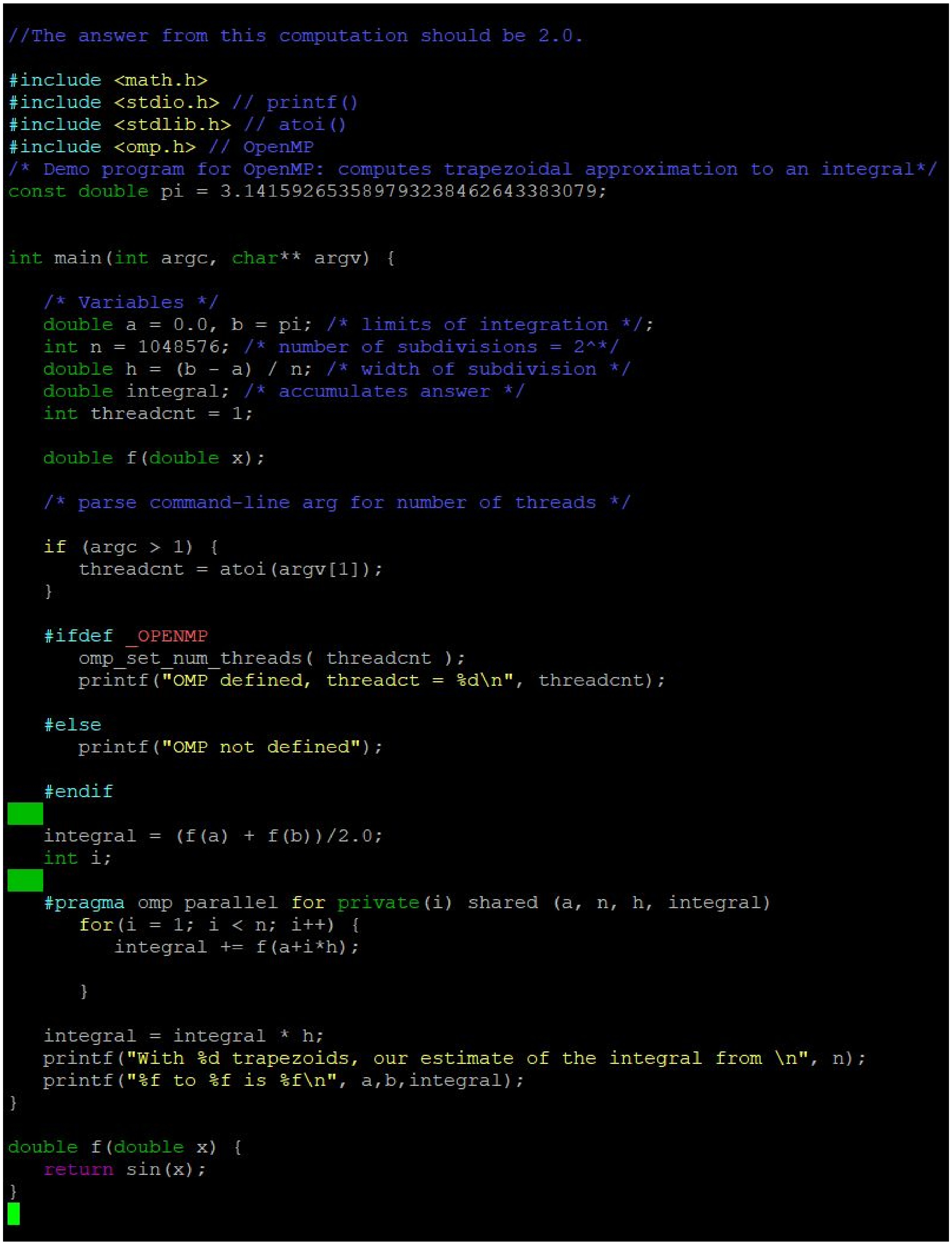
As for the first loop, when we unroll it into separate iterations, we find that each statement is independent from the others. As for the second loop, in each iteration, the second statement is dependent on the first.

Task 3b

Parallel Programming Basics

1. First, I created trap-networking.c. This is a C program designed to compute the integral seen below. It does so by approximating it using in 2^20 equal subdivisions. These subdivisions are split evenly between the number of threads you assign to the program.

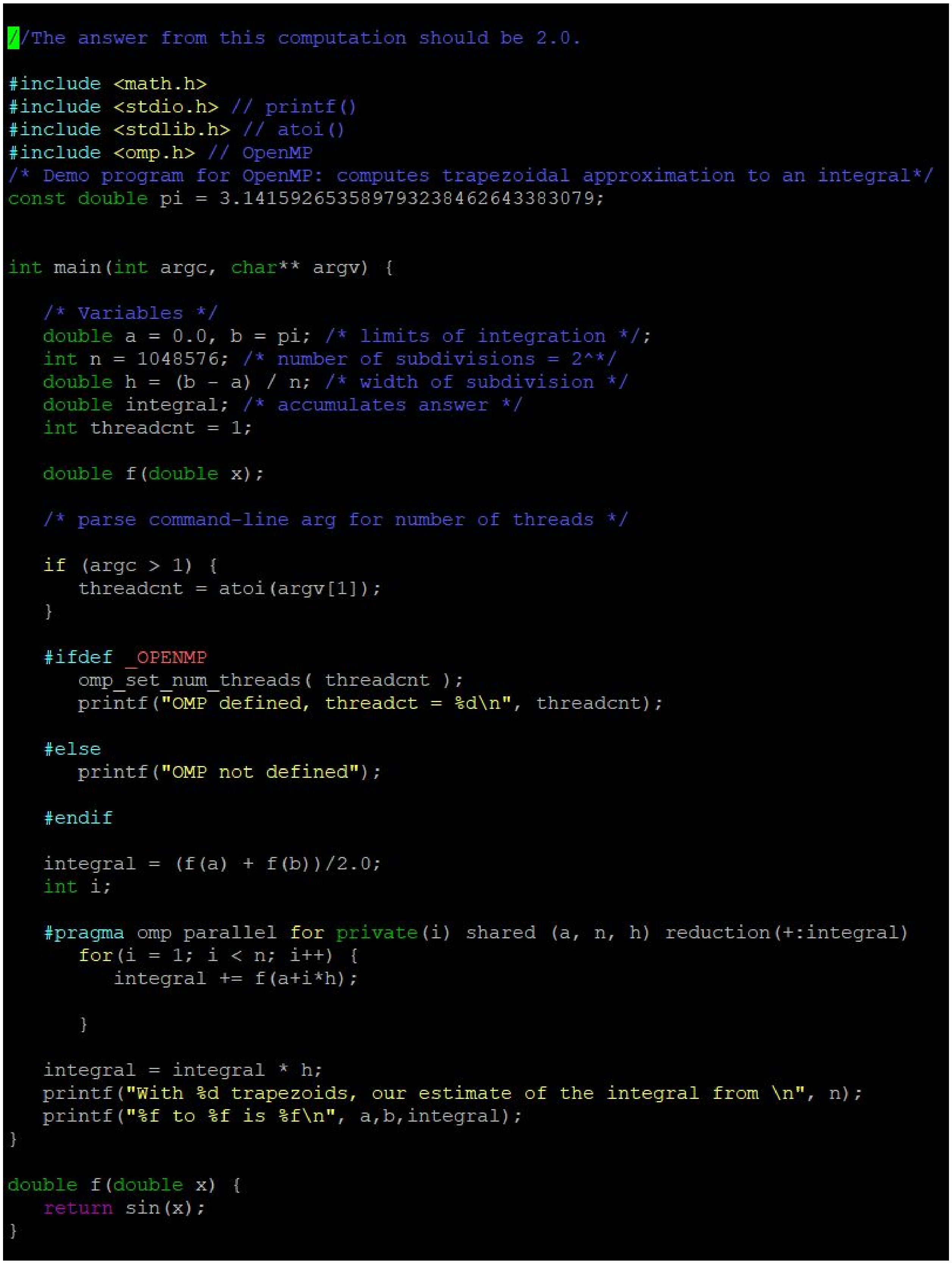
$$\int_0^\pi \sin(x)dx$$



```c
//The answer from this computation should be 2.0.

#include <math.h>
#include <stdio.h> // printf()
#include <stdlib.h> // atoi()
#include <omp.h> // OpenMP
/* Demo program for OpenMP: computes trapezoidal approximation to an integral*/
const double pi = 3.141592653589793238462643383079;


int main(int argc, char** argv) {

   /* Variables */
   double a = 0.0, b = pi; /* limits of integration */;
   int n = 1048576; /* number of subdivisions = 2^*/
   double h = (b - a) / n; /* width of subdivision */
   double integral; /* accumulates answer */
   int threadcnt = 1;

   double f(double x);

   /* parse command-line arg for number of threads */

   if (argc > 1) {
      threadcnt = atoi(argv[1]);
   }

   #ifdef _OPENMP
      omp_set_num_threads( threadcnt );
      printf("OMP defined, threadct = %d\n", threadcnt);

   #else
      printf("OMP not defined");

   #endif

   integral = (f(a) + f(b))/2.0;
   int i;

   #pragma omp parallel for private(i) shared (a, n, h, integral)
      for(i = 1; i < n; i++) {
         integral += f(a+i*h);

      }

   integral = integral * h;
   printf("With %d trapezoids, our estimate of the integral from \n", n);
   printf("%f to %f is %f\n", a,b,integral);
}

double f(double x) {
   return sin(x);
}
```

trap-notworking.c

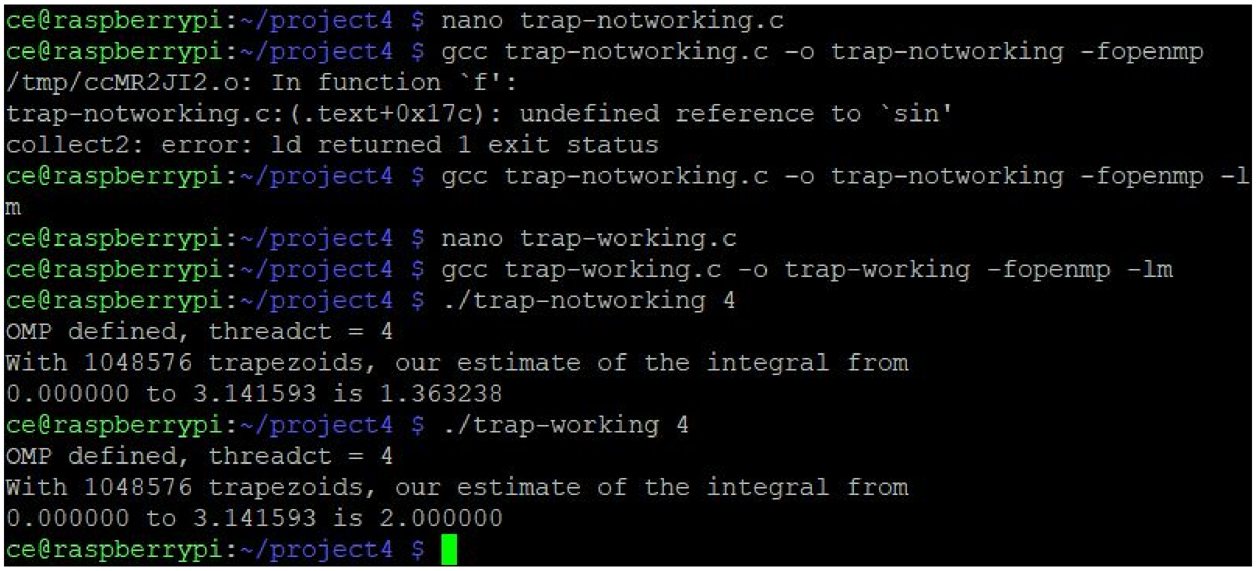
2. Next, I created trap-working. It is the same as trap-working, except the variable holding the sum isn’t purely shared between each thread. Instead, it is a reduction. Each thread gets its own copy of the variable that it uses to update the main one to avoid a race condition.



```
//The answer from this computation should be 2.0.

#include <math.h>
#include <stdio.h> // printf()
#include <stdlib.h> // atoi()
#include <omp.h> // OpenMP
/* Demo program for OpenMP: computes trapezoidal approximation to an integral*/
const double pi = 3.141592653589793238462643383079;


int main(int argc, char** argv) {

   /* Variables */
   double a = 0.0, b = pi; /* limits of integration */;
   int n = 1048576; /* number of subdivisions = 2^**/
   double h = (b - a) / n; /* width of subdivision */
   double integral; /* accumulates answer */
   int threadcnt = 1;

   double f(double x);

   /* parse command-line arg for number of threads */

   if (argc > 1) {
      threadcnt = atoi(argv[1]);
   }

   #ifdef _OPENMP
      omp_set_num_threads( threadcnt );
      printf("OMP defined, threadct = %d\n", threadcnt);

   #else
      printf("OMP not defined");

   #endif

   integral = (f(a) + f(b))/2.0;
   int i;

   #pragma omp parallel for private(i) shared (a, n, h) reduction(+:integral)
      for(i = 1; i < n; i++) {
         integral += f(a+i*h);

      }

   integral = integral * h;
   printf("With %d trapezoids, our estimate of the integral from \n", n);
   printf("%f to %f is %f\n", a,b,integral);
}

double f(double x) {
   return sin(x);
}
```

trap-working.c

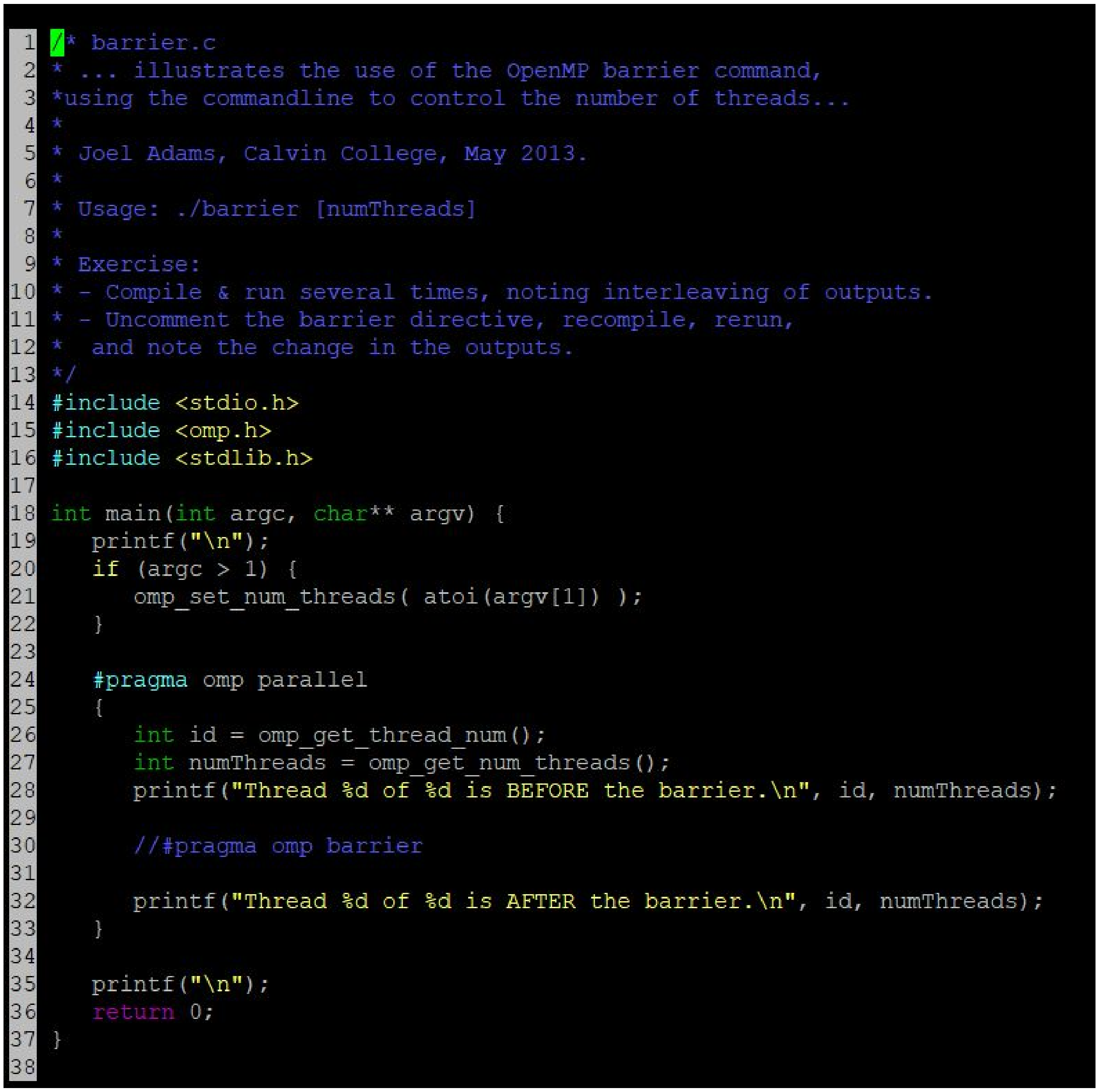
3. After creating both a working and non-working version of this program, I compiled both and ran them. The working one gave the expecting result of 2 whereas the non-working one gave a different result.



```
ce@raspberrypi:~/project4 $ nano trap-notworking.c
ce@raspberrypi:~/project4 $ gcc trap-notworking.c -o trap-notworking -fopenmp
/tmp/ccMR2JI2.o: In function `f':
trap-notworking.c:(.text+0x17c): undefined reference to `sin'
collect2: error: ld returned 1 exit status
ce@raspberrypi:~/project4 $ gcc trap-notworking.c -o trap-notworking -fopenmp -l
m
ce@raspberrypi:~/project4 $ nano trap-working.c
ce@raspberrypi:~/project4 $ gcc trap-working.c -o trap-working -fopenmp -lm
ce@raspberrypi:~/project4 $ ./trap-notworking 4
OMP defined, threadct = 4
With 1048576 trapezoids, our estimate of the integral from
0.000000 to 3.141593 is 1.363238
ce@raspberrypi:~/project4 $ ./trap-working 4
OMP defined, threadct = 4
With 1048576 trapezoids, our estimate of the integral from
0.000000 to 3.141593 is 2.000000
ce@raspberrypi:~/project4 $
```

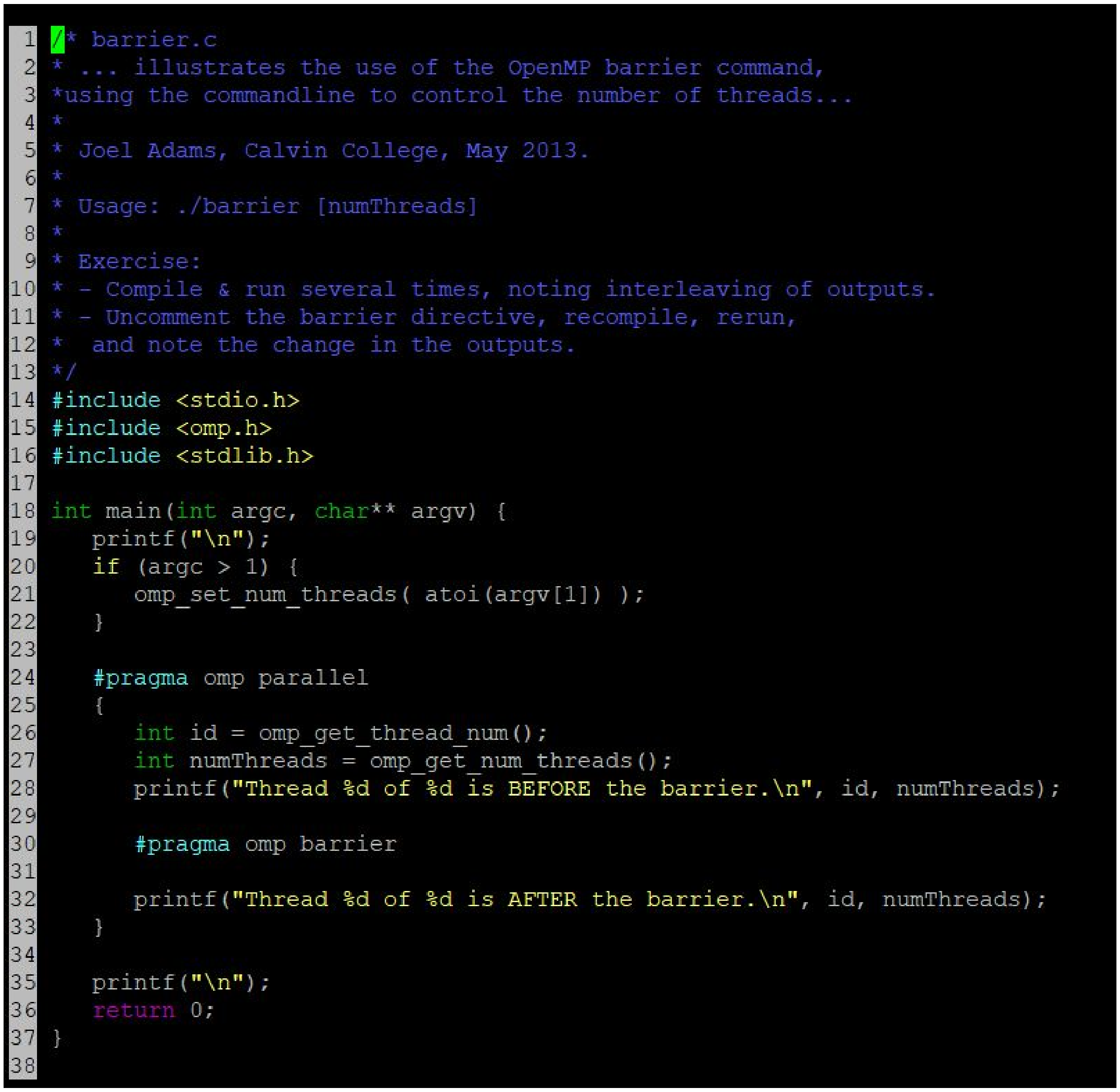
Results of trap-(not)/working

4. My next task was to create a program called barrier.c. This program displays the function of the barrier pattern, which is a pattern that forces each thread to stop and wait until all threads have reached a certain step in the program. This is useful when dependency is a concern. I first created it with the barrier function disabled and ran it. I then enabled the barrier function and ran it once again. With barrier disabled, each thread completed the entire operation at once. With barrier enabled, each thread stopped at the barrier until every thread had reached it.



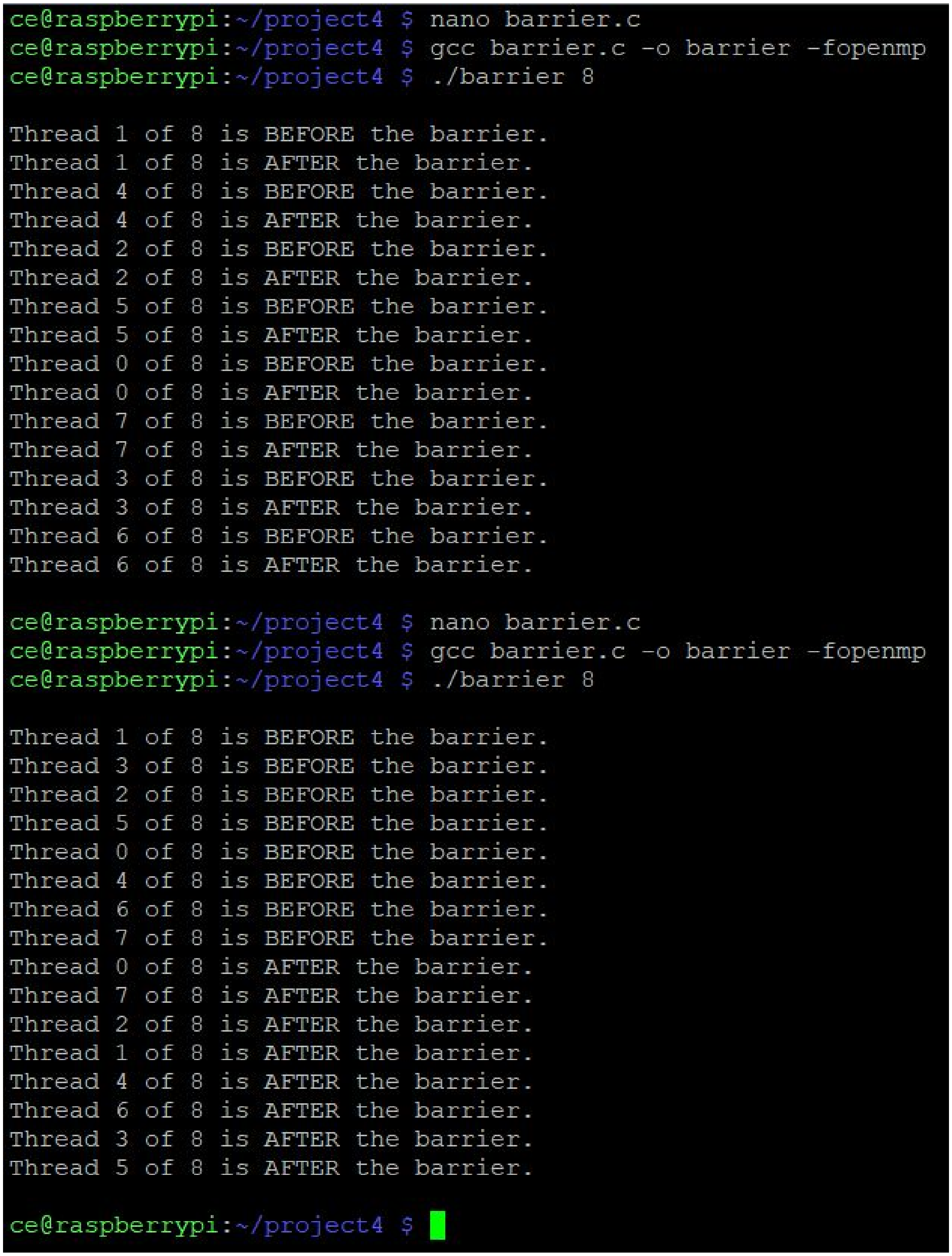
```c
/* barrier.c
* ... illustrates the use of the OpenMP barrier command,
*using the commandline to control the number of threads...
*
* Joel Adams, Calvin College, May 2013.
*
* Usage: ./barrier [numThreads]
*
* Exercise:
* - Compile & run several times, noting interleaving of outputs.
* - Uncomment the barrier directive, recompile, rerun,
*   and note the change in the outputs.
*/
#include <stdio.h>
#include <omp.h>
#include <stdlib.h>

int main(int argc, char** argv) {
   printf("\n");
   if (argc > 1) {
      omp_set_num_threads( atoi(argv[1]) );
   }

   #pragma omp parallel
   {
      int id = omp_get_thread_num();
      int numThreads = omp_get_num_threads();
      printf("Thread %d of %d is BEFORE the barrier.\n", id, numThreads);

      //#pragma omp barrier

      printf("Thread %d of %d is AFTER the barrier.\n", id, numThreads);
   }

   printf("\n");
   return 0;
}
```

barrier.c with barrier functionality disabled



```
/* barrier.c
* ... illustrates the use of the OpenMP barrier command,
*using the commandline to control the number of threads...
*
* Joel Adams, Calvin College, May 2013.
*
* Usage: ./barrier [numThreads]
*
* Exercise:
* - Compile & run several times, noting interleaving of outputs.
* - Uncomment the barrier directive, recompile, rerun,
*  and note the change in the outputs.
*/
#include <stdio.h>
#include <omp.h>
#include <stdlib.h>

int main(int argc, char** argv) {
   printf("\n");
   if (argc > 1) {
      omp_set_num_threads( atoi(argv[1]) );
   }

   #pragma omp parallel
   {
      int id = omp_get_thread_num();
      int numThreads = omp_get_num_threads();
      printf("Thread %d of %d is BEFORE the barrier.\n", id, numThreads);

      #pragma omp barrier

      printf("Thread %d of %d is AFTER the barrier.\n", id, numThreads);
   }

   printf("\n");
   return 0;
}

```

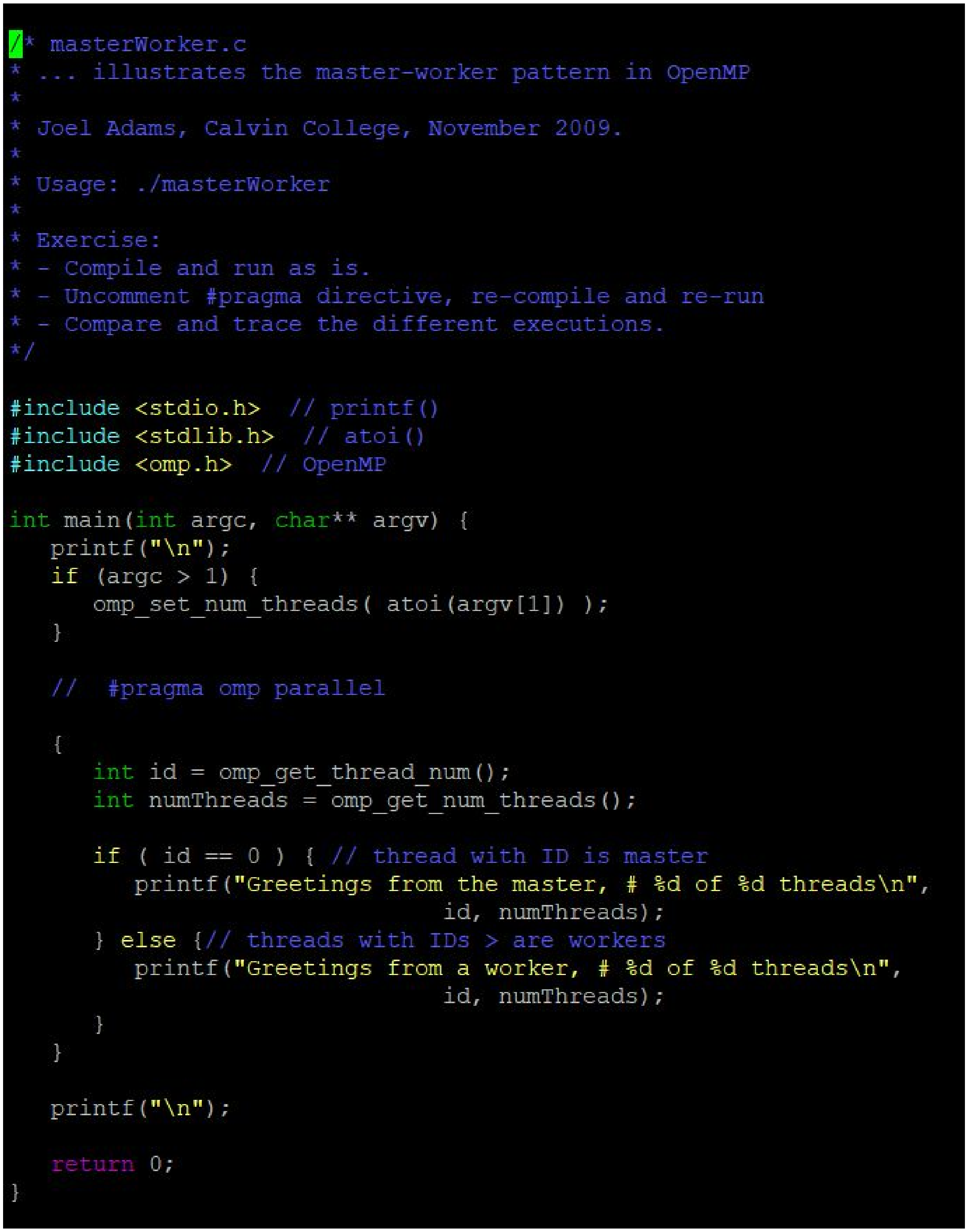
barrier.c with barrier function enabled



```
ce@raspberrypi:~/project4 $ nano barrier.c
ce@raspberrypi:~/project4 $ gcc barrier.c -o barrier -fopenmp
ce@raspberrypi:~/project4 $ ./barrier 8

Thread 1 of 8 is BEFORE the barrier.
Thread 1 of 8 is AFTER the barrier.
Thread 4 of 8 is BEFORE the barrier.
Thread 4 of 8 is AFTER the barrier.
Thread 2 of 8 is BEFORE the barrier.
Thread 2 of 8 is AFTER the barrier.
Thread 5 of 8 is BEFORE the barrier.
Thread 5 of 8 is AFTER the barrier.
Thread 0 of 8 is BEFORE the barrier.
Thread 0 of 8 is AFTER the barrier.
Thread 7 of 8 is BEFORE the barrier.
Thread 7 of 8 is AFTER the barrier.
Thread 3 of 8 is BEFORE the barrier.
Thread 3 of 8 is AFTER the barrier.
Thread 6 of 8 is BEFORE the barrier.
Thread 6 of 8 is AFTER the barrier.

ce@raspberrypi:~/project4 $ nano barrier.c
ce@raspberrypi:~/project4 $ gcc barrier.c -o barrier -fopenmp
ce@raspberrypi:~/project4 $ ./barrier 8

Thread 1 of 8 is BEFORE the barrier.
Thread 3 of 8 is BEFORE the barrier.
Thread 2 of 8 is BEFORE the barrier.
Thread 5 of 8 is BEFORE the barrier.
Thread 0 of 8 is BEFORE the barrier.
Thread 4 of 8 is BEFORE the barrier.
Thread 6 of 8 is BEFORE the barrier.
Thread 7 of 8 is BEFORE the barrier.
Thread 0 of 8 is AFTER the barrier.
Thread 7 of 8 is AFTER the barrier.
Thread 2 of 8 is AFTER the barrier.
Thread 1 of 8 is AFTER the barrier.
Thread 4 of 8 is AFTER the barrier.
Thread 6 of 8 is AFTER the barrier.
Thread 3 of 8 is AFTER the barrier.
Thread 5 of 8 is AFTER the barrier.

ce@raspberrypi:~/project4 $
```

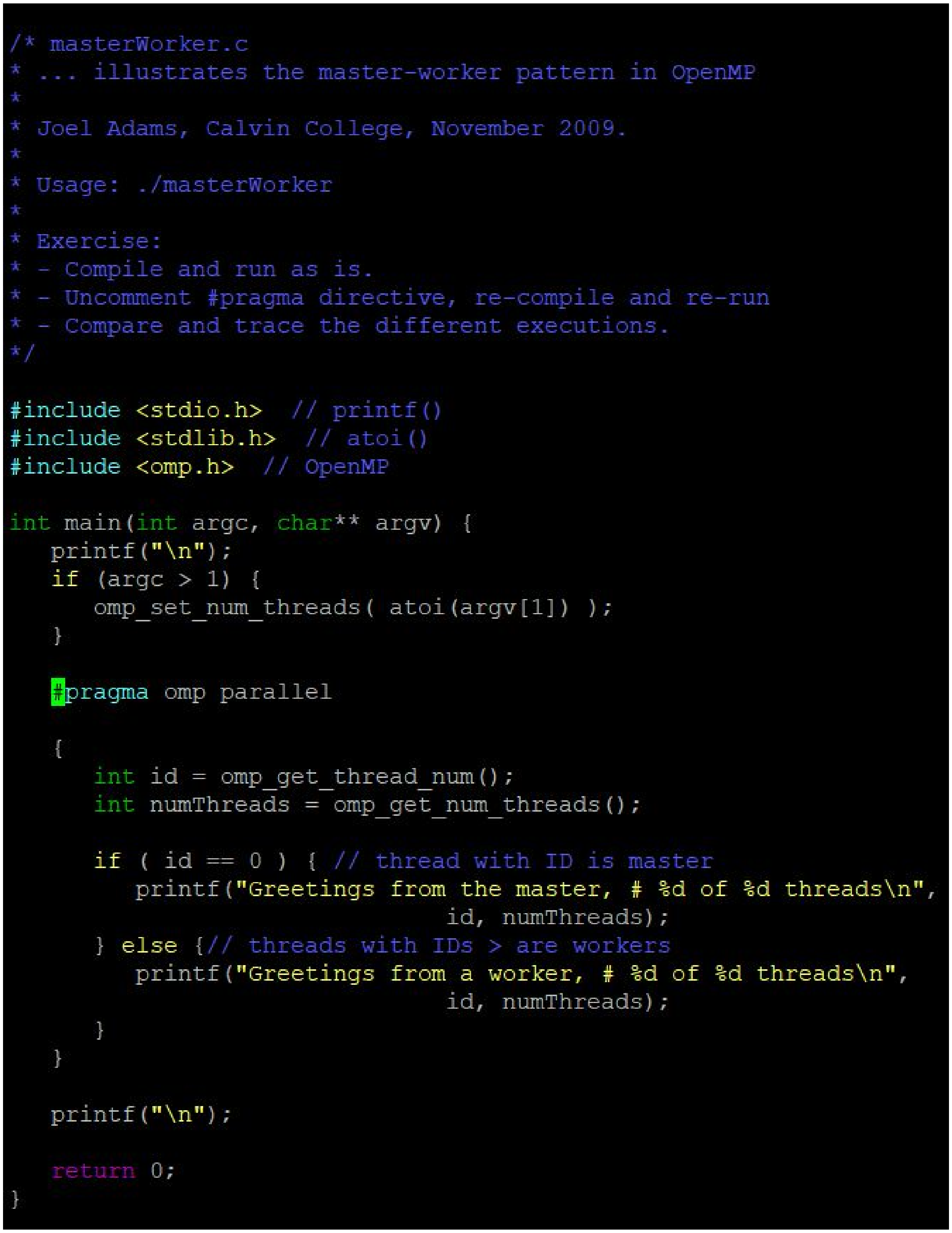
Results of barrier without and then with barrier functionality

5. Finally, I created a program called masterWorker.c. This program is designed to display a basic Master-Worker implementation. Essentially, the master thread executes one block of code while the workers execute another. First, the program is compiled and run without parallelization. In this instance, it only print “Greetings from the master, # 0 of 1 threads” since it is only a single time. Next, the program is compiled with an openmp directive. This time, we see the Master’s code block execute once and the Worker block execute 3 times.



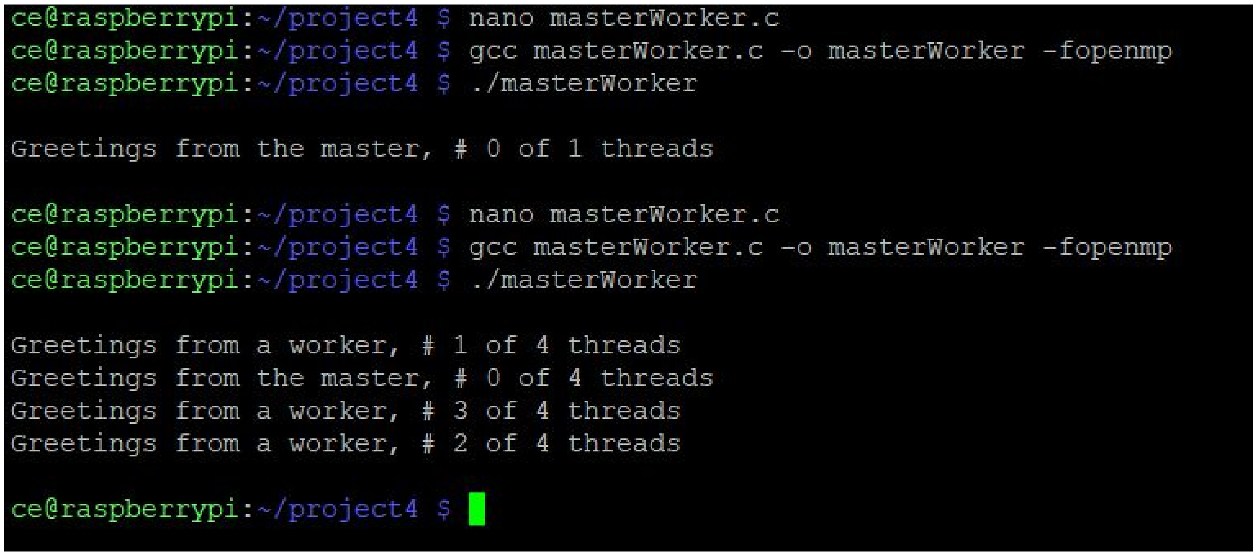
```
/* masterWorker.c
* ... illustrates the master-worker pattern in OpenMP
*
* Joel Adams, Calvin College, November 2009.
*
* Usage: ./masterWorker
*
* Exercise:
* - Compile and run as is.
* - Uncomment #pragma directive, re-compile and re-run
* - Compare and trace the different executions.
*/

#include <stdio.h>  // printf()
#include <stdlib.h>  // atoi()
#include <omp.h>  // OpenMP

int main(int argc, char** argv) {
   printf("\n");
   if (argc > 1) {
      omp_set_num_threads( atoi(argv[1]) );
   }

   //  #pragma omp parallel

   {
      int id = omp_get_thread_num();
      int numThreads = omp_get_num_threads();

      if ( id == 0 ) { // thread with ID is master
         printf("Greetings from the master, # %d of %d threads\n",
                              id, numThreads);
      } else {// threads with IDs > are workers
         printf("Greetings from a worker, # %d of %d threads\n",
                              id, numThreads);
      }
   }

   printf("\n");

   return 0;
}
```

masterWorker.c without parallelization



```
/* masterWorker.c
 * ... illustrates the master-worker pattern in OpenMP
 *
 * Joel Adams, Calvin College, November 2009.
 *
 * Usage: ./masterWorker
 *
 * Exercise:
 * - Compile and run as is.
 * - Uncomment #pragma directive, re-compile and re-run
 * - Compare and trace the different executions.
 */

#include <stdio.h>   // printf()
#include <stdlib.h>  // atoi()
#include <omp.h>  // OpenMP

int main(int argc, char** argv) {
   printf("\n");
   if (argc > 1) {
      omp_set_num_threads( atoi(argv[1]) );
   }

   #pragma omp parallel

   {
      int id = omp_get_thread_num();
      int numThreads = omp_get_num_threads();

      if ( id == 0 ) { // thread with ID is master
         printf("Greetings from the master, # %d of %d threads\n",
                              id, numThreads);
      } else {// threads with IDs > are workers
         printf("Greetings from a worker, # %d of %d threads\n",
                              id, numThreads);
      }
   }

   printf("\n");

   return 0;
}
```

masterWorker.c with parallelization

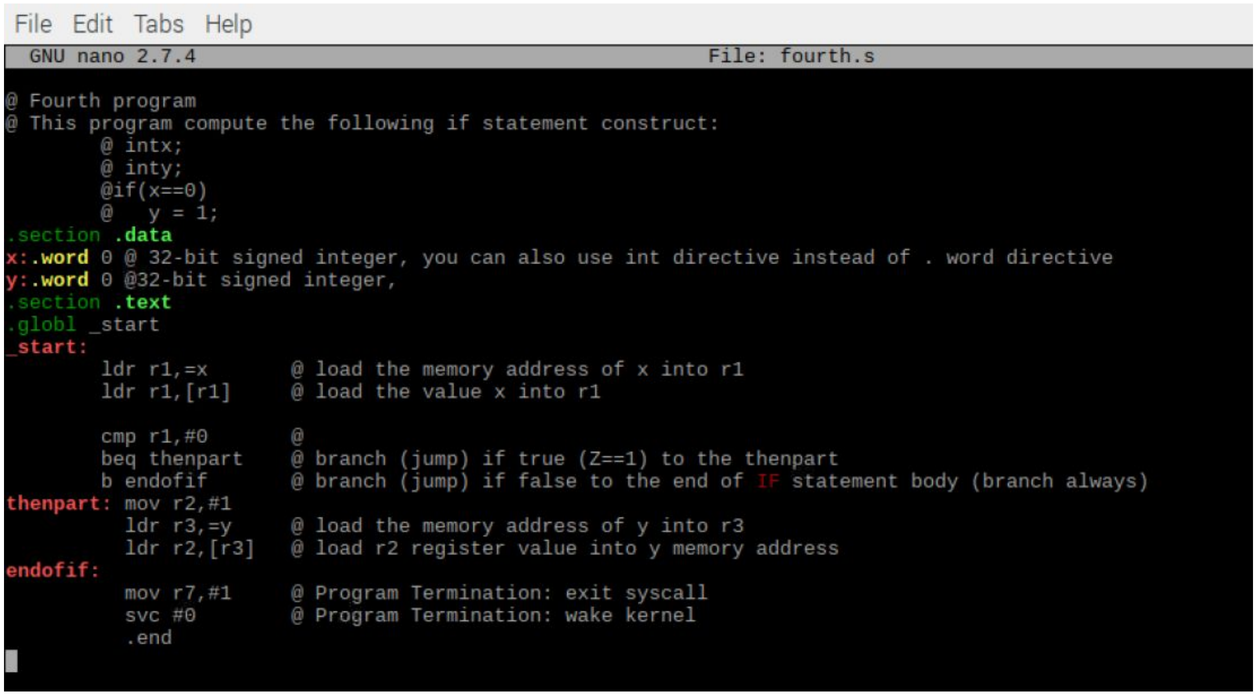


```
ce@raspberrypi:~/project4 $ nano masterWorker.c
ce@raspberrypi:~/project4 $ gcc masterWorker.c -o masterWorker -fopenmp
ce@raspberrypi:~/project4 $ ./masterWorker

Greetings from the master, # 0 of 1 threads

ce@raspberrypi:~/project4 $ nano masterWorker.c
ce@raspberrypi:~/project4 $ gcc masterWorker.c -o masterWorker -fopenmp
ce@raspberrypi:~/project4 $ ./masterWorker

Greetings from a worker, # 1 of 4 threads
Greetings from the master, # 0 of 4 threads
Greetings from a worker, # 3 of 4 threads
Greetings from a worker, # 2 of 4 threads

ce@raspberrypi:~/project4 $
```

results of running masterWorker both without and with parallelization

Task 4
ARM Programming

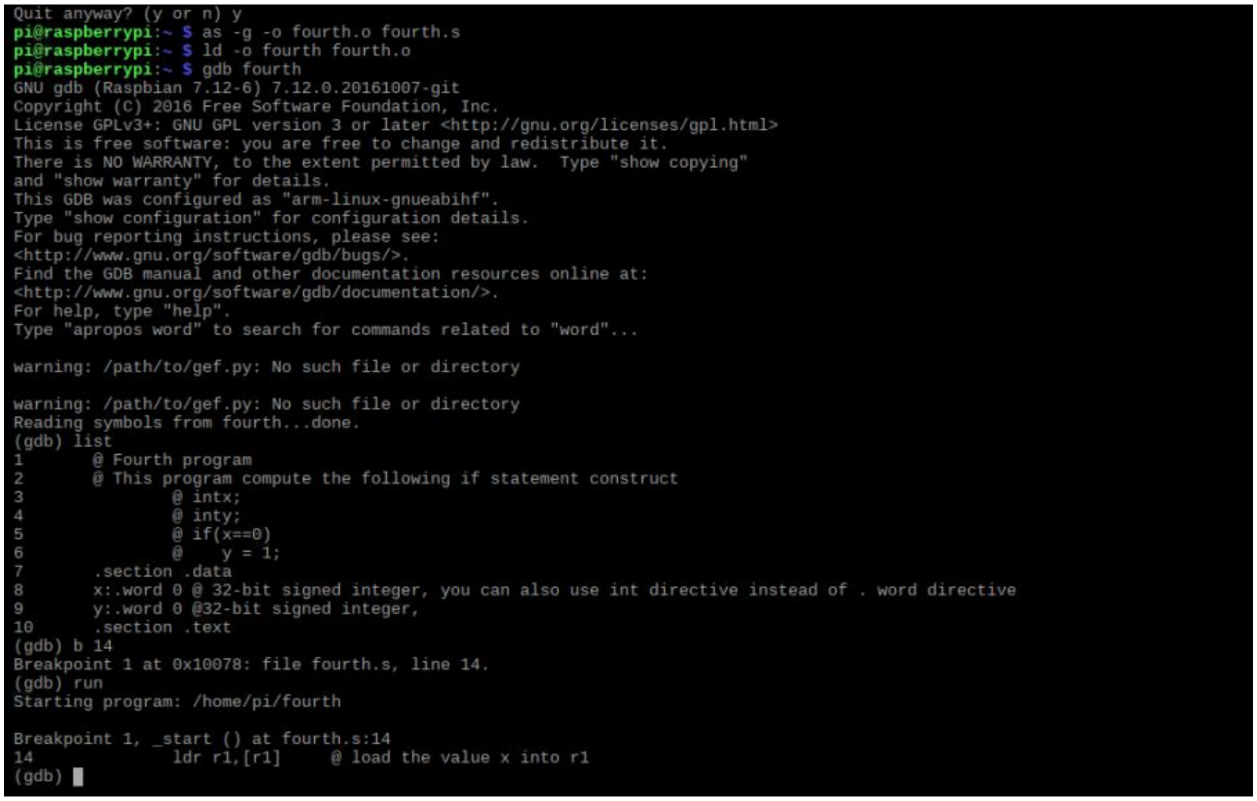
Part 1.



```
  GNU nano 2.7.4                                          File: fourth.s

@ Fourth program
@ This program compute the following if statement construct:
        @ intx;
        @ inty;
        @if(x==0)
        @   y = 1;
.section .data
x:.word 0 @ 32-bit signed integer, you can also use int directive instead of . word directive
y:.word 0 @32-bit signed integer,
.section .text
.globl _start
_start:
        ldr r1,=x       @ load the memory address of x into r1
        ldr r1,[r1]     @ load the value x into r1

        cmp r1,#0       @
        beq thenpart    @ branch (jump) if true (Z==1) to the thenpart
        b endofif       @ branch (jump) if false to the end of IF statement body (branch always)
thenpart: mov r2,#1
          ldr r3,=y     @ load the memory address of y into r3
          ldr r2,[r3]   @ load r2 register value into y memory address
endofif:
          mov r7,#1     @ Program Termination: exit syscall
          svc #0        @ Program Termination: wake kernel
          .end
```

This is the script as seen on the nano editor.

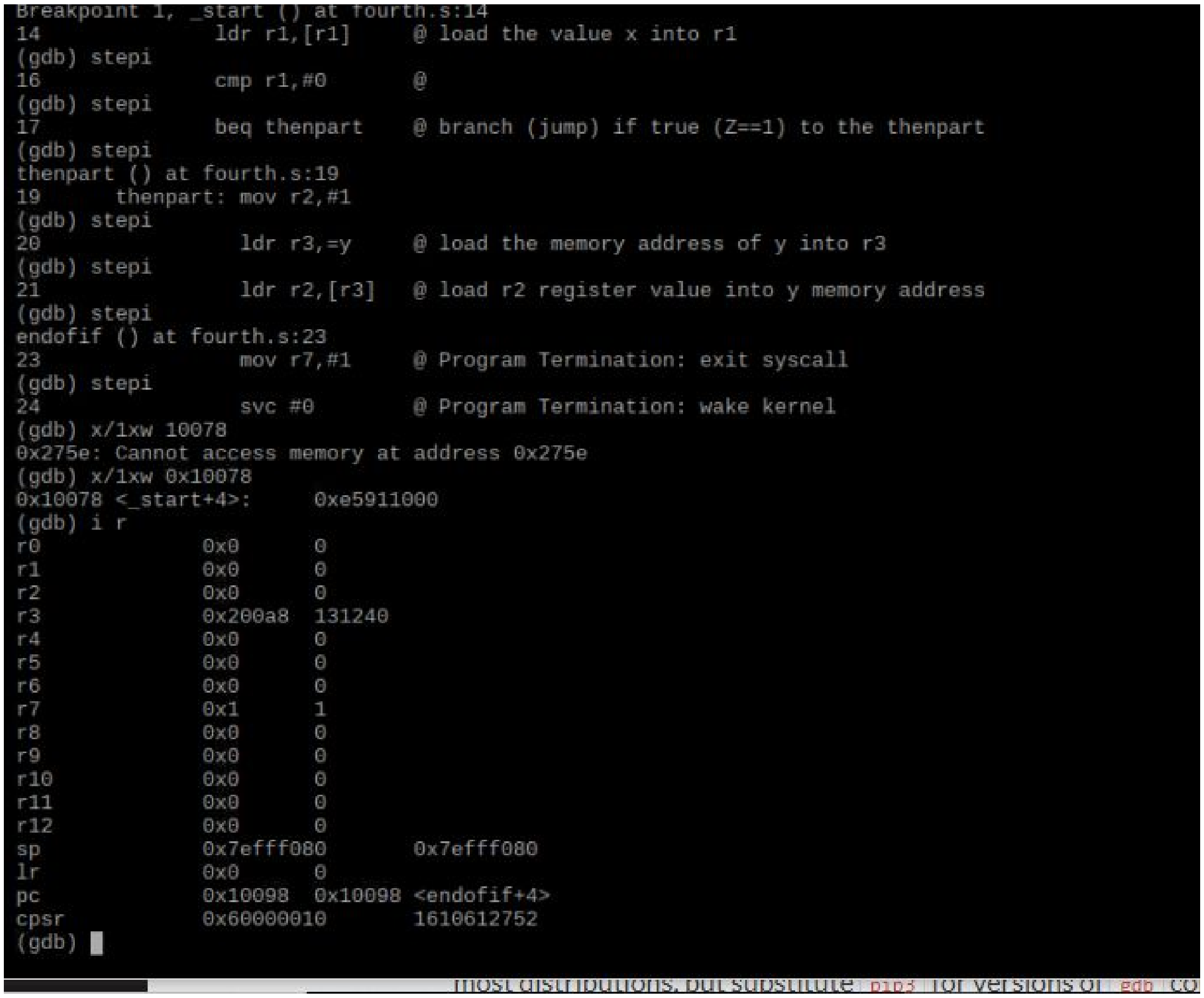
Part 1. Continued



```
Quit anyway? (y or n) y
pi@raspberrypi:~ $ as -g -o fourth.o fourth.s
pi@raspberrypi:~ $ ld -o fourth fourth.o
pi@raspberrypi:~ $ gdb fourth
GNU gdb (Raspbian 7.12-6) 7.12.0.20161007-git
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "arm-linux-gnueabihf".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...

warning: /path/to/gef.py: No such file or directory

warning: /path/to/gef.py: No such file or directory
Reading symbols from fourth...done.
(gdb) list
1       @ Fourth program
2       @ This program compute the following if statement construct
3               @ intx;
4               @ inty;
5               @ if(x==0)
6               @    y = 1;
7       .section .data
8       x:.word 0 @ 32-bit signed integer, you can also use int directive instead of . word directive
9       y:.word 0 @32-bit signed integer,
10      .section .text
(gdb) b 14
Breakpoint 1 at 0x10078: file fourth.s, line 14.
(gdb) run
Starting program: /home/pi/fourth

Breakpoint 1, _start () at fourth.s:14
14              ldr r1,[r1]     @ load the value x into r1
(gdb) 
```

The breakpoint is at line 14, and the program runs.

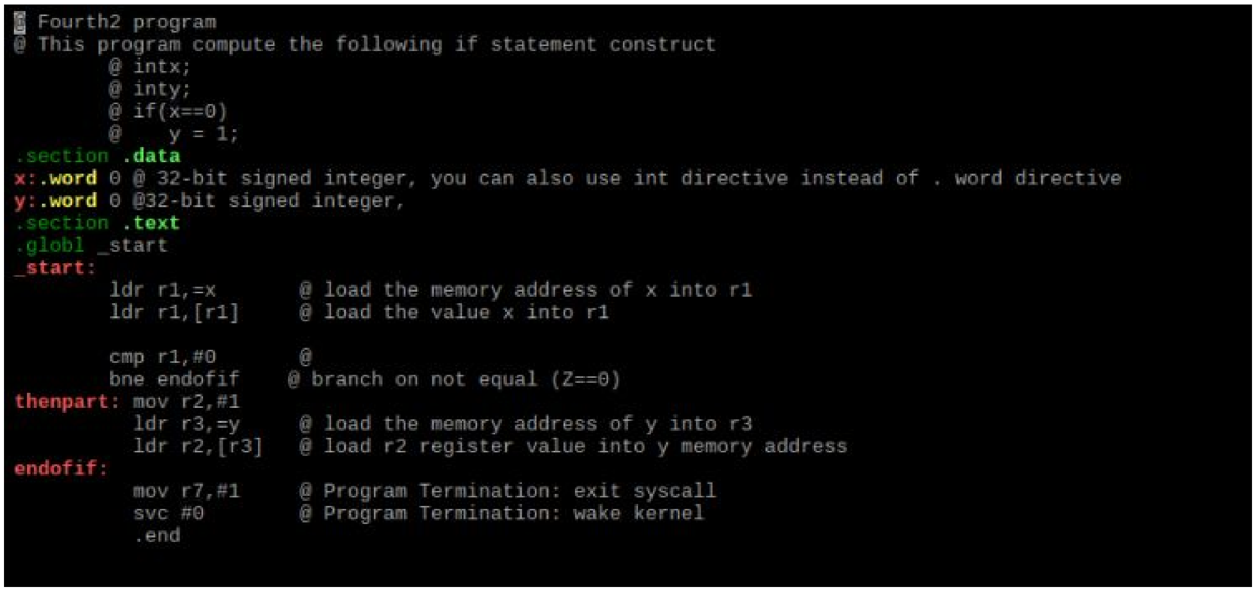
Part 1. Continued



```
Breakpoint 1, _start () at fourth.s:14
14              ldr r1,[r1]     @ load the value x into r1
(gdb) stepi
16              cmp r1,#0       @
(gdb) stepi
17              beq thenpart    @ branch (jump) if true (Z==1) to the thenpart
(gdb) stepi
thenpart () at fourth.s:19
19      thenpart: mov r2,#1
(gdb) stepi
20                ldr r3,=y     @ load the memory address of y into r3
(gdb) stepi
21                ldr r2,[r3]   @ load r2 register value into y memory address
(gdb) stepi
endofif () at fourth.s:23
23                mov r7,#1     @ Program Termination: exit syscall
(gdb) stepi
24                svc #0        @ Program Termination: wake kernel
(gdb) x/1xw 10078
0x275e: Cannot access memory at address 0x275e
(gdb) x/1xw 0x10078
0x10078 <_start+4>:     0xe5911000
(gdb) i r
r0             0x0      0
r1             0x0      0
r2             0x0      0
r3             0x200a8  131240
r4             0x0      0
r5             0x0      0
r6             0x0      0
r7             0x1      1
r8             0x0      0
r9             0x0      0
r10            0x0      0
r11            0x0      0
r12            0x0      0
sp             0x7efff080       0x7efff080
lr             0x0      0
pc             0x10098  0x10098 <endofif+4>
cpsr           0x60000010       1610612752
(gdb) 
```

The y memory location is at 0xe5911000. The cpsr value is 0x60000010 in hexadecimal, which is 0110 0000 0000 0000 0000 0000 0001 0000. The bits 31 to 28 determine if the condition will be executed. The 4 bits are 0110. The Z flag is the 30th bit. The Z value is 1 on the 0110 field.

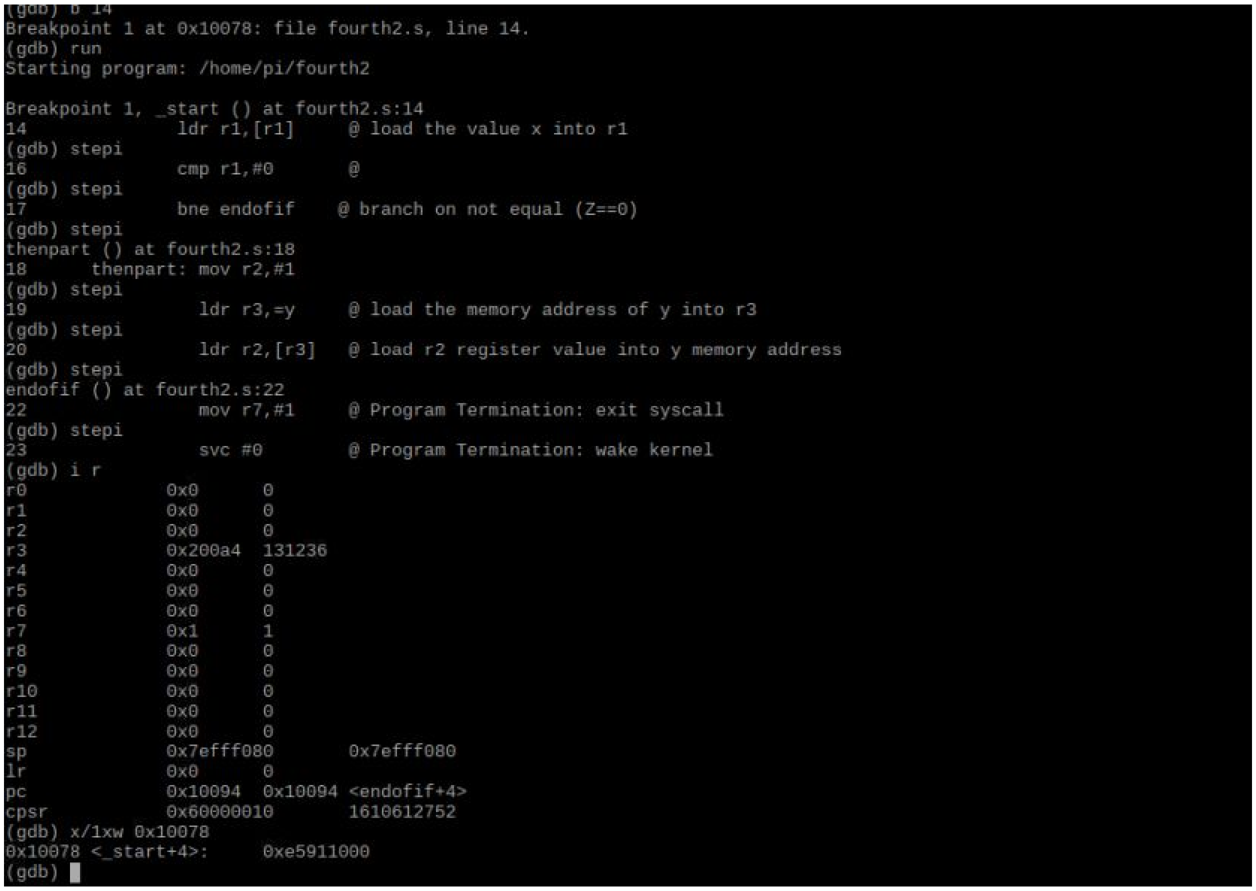
Part 2.



```
@ Fourth2 program
@ This program compute the following if statement construct
        @ intx;
        @ inty;
        @ if(x==0)
        @    y = 1;
.section .data
x:.word 0 @ 32-bit signed integer, you can also use int directive instead of . word directive
y:.word 0 @32-bit signed integer,
.section .text
.globl _start
_start:
        ldr r1,=x       @ load the memory address of x into r1
        ldr r1,[r1]     @ load the value x into r1

        cmp r1,#0       @
        bne endofif    @ branch on not equal (Z==0)
thenpart: mov r2,#1
          ldr r3,=y     @ load the memory address of y into r3
          ldr r2,[r3]   @ load r2 register value into y memory address
endofif:
          mov r7,#1     @ Program Termination: exit syscall
          svc #0        @ Program Termination: wake kernel
          .end
```

Instead of using beq and b, bne (branch on not equal) was used instead, which renders the program more efficient.

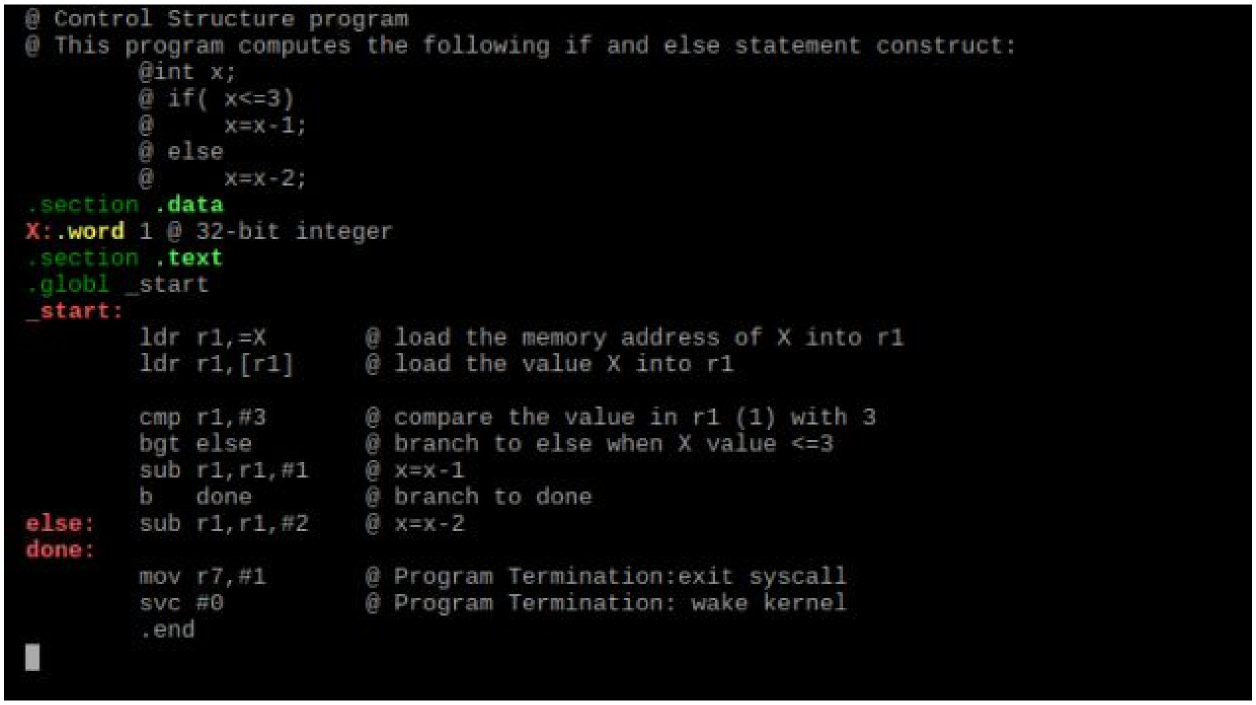
Part 2. Continued



```
(gdb) b 14
Breakpoint 1 at 0x10078: file fourth2.s, line 14.
(gdb) run
Starting program: /home/pi/fourth2

Breakpoint 1, _start () at fourth2.s:14
14              ldr r1,[r1]     @ load the value x into r1
(gdb) stepi
16              cmp r1,#0       @
(gdb) stepi
17              bne endofif    @ branch on not equal (Z==0)
(gdb) stepi
thenpart () at fourth2.s:18
18      thenpart: mov r2,#1
(gdb) stepi
19                ldr r3,=y     @ load the memory address of y into r3
(gdb) stepi
20                ldr r2,[r3]   @ load r2 register value into y memory address
(gdb) stepi
endofif () at fourth2.s:22
22                mov r7,#1     @ Program Termination: exit syscall
(gdb) stepi
23                svc #0        @ Program Termination: wake kernel
(gdb) i r
r0             0x0      0
r1             0x0      0
r2             0x0      0
r3             0x200a4  131236
r4             0x0      0
r5             0x0      0
r6             0x0      0
r7             0x1      1
r8             0x0      0
r9             0x0      0
r10            0x0      0
r11            0x0      0
r12            0x0      0
sp             0x7efff080       0x7efff080
lr             0x0      0
pc             0x10094  0x10094 <endofif+4>
cpsr           0x60000010       1610612752
(gdb) x/1xw 0x10078
0x10078 <_start+4>:     0xe5911000
(gdb)
```

The cpsr is 0x60000010, which is 0110 0000 0000 0000 0000 0000 0001 0000. The 31:28 bit field is 0110. The 30th bit corresponds to the Z flag. In this case, it is 1. The Z value is 1.

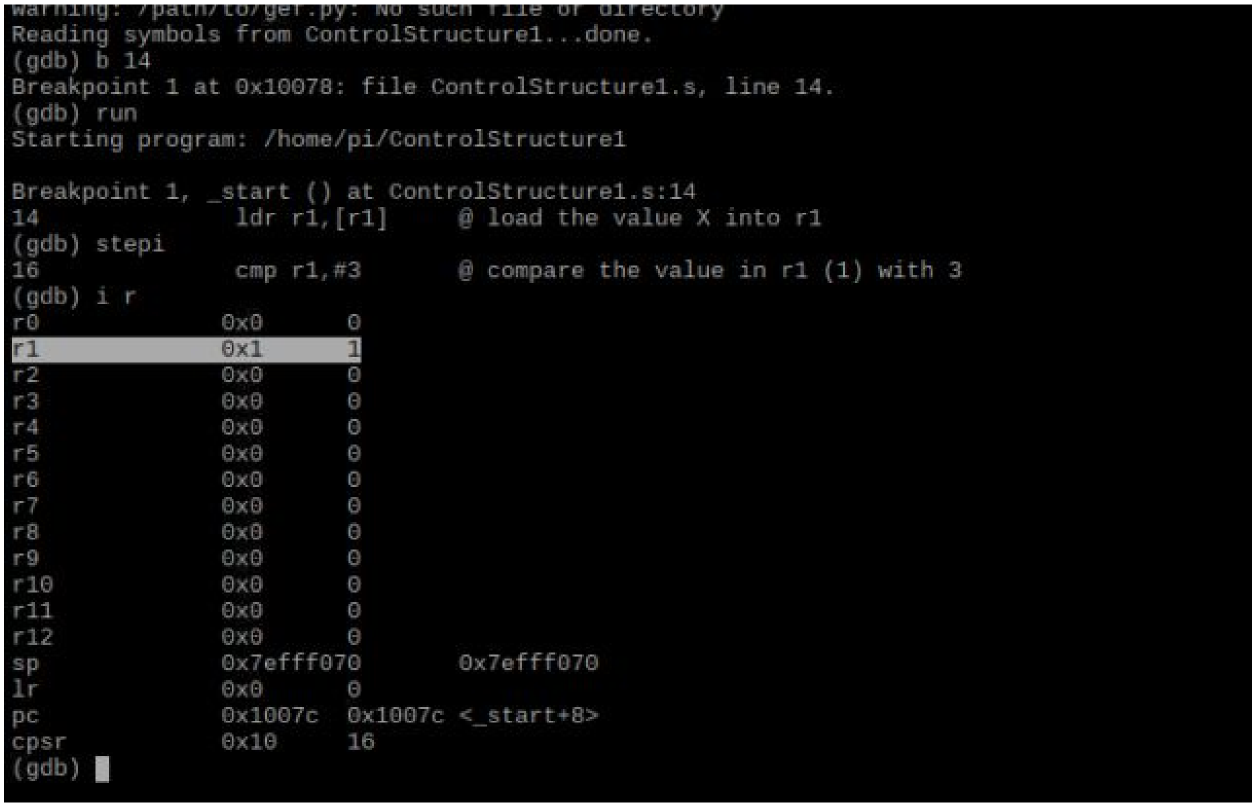
Part 3.



```
@ Control Structure program
@ This program computes the following if and else statement construct:
        @int x;
        @ if( x<=3)
        @     x=x-1;
        @ else
        @     x=x-2;
.section .data
X:.word 1 @ 32-bit integer
.section .text
.globl _start
_start:
        ldr r1,=X       @ load the memory address of X into r1
        ldr r1,[r1]     @ load the value X into r1

        cmp r1,#3       @ compare the value in r1 (1) with 3
        bgt else        @ branch to else when X value <=3
        sub r1,r1,#1    @ x=x-1
        b   done        @ branch to done
else:   sub r1,r1,#2    @ x=x-2
done:
        mov r7,#1       @ Program Termination:exit syscall
        svc #0          @ Program Termination: wake kernel
        .end
```

This is the script for the ControlStructure.s program

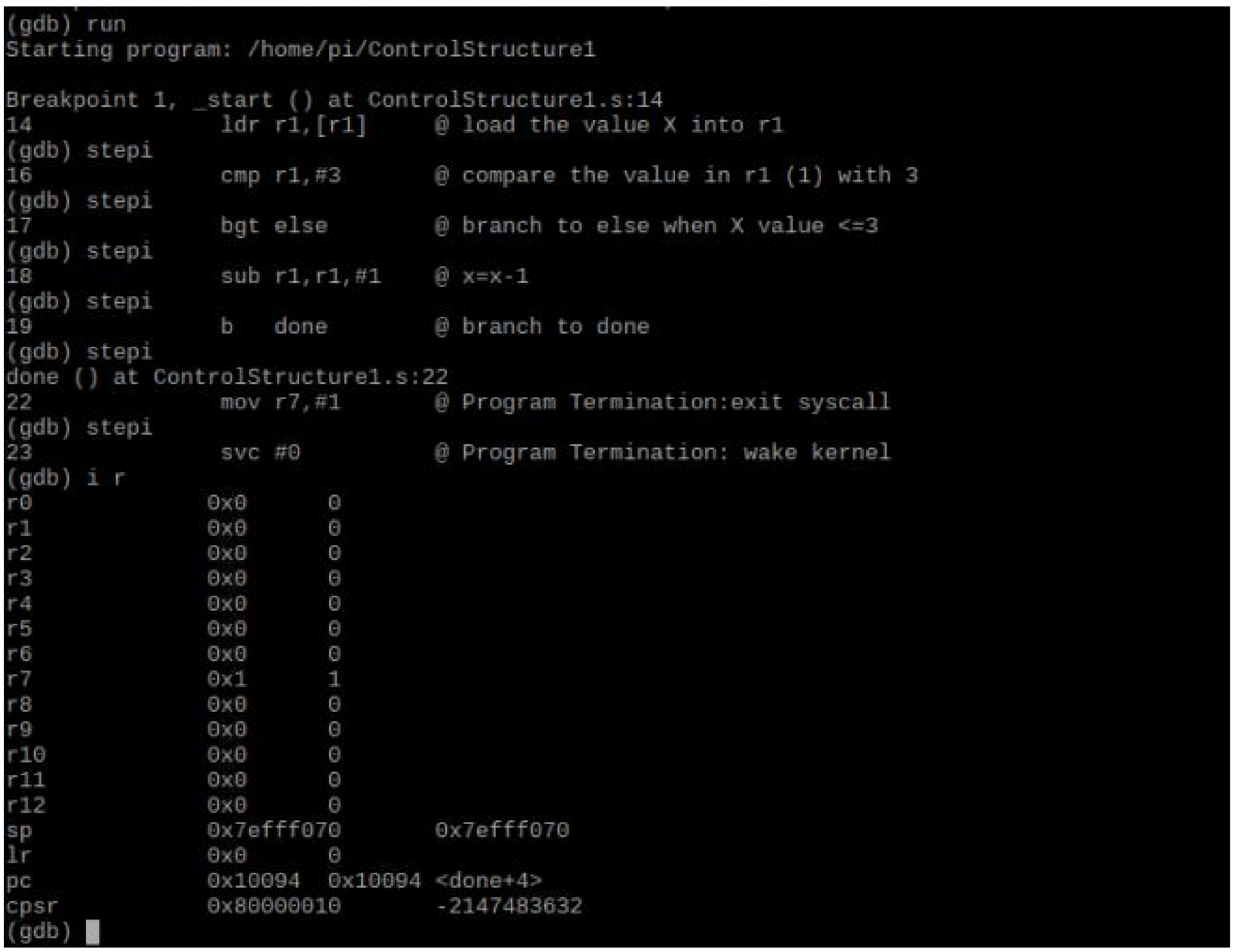
Part3. Continued



```
warning: /path/to/gef.py: No such file or directory
Reading symbols from ControlStructure1...done.
(gdb) b 14
Breakpoint 1 at 0x10078: file ControlStructure1.s, line 14.
(gdb) run
Starting program: /home/pi/ControlStructure1

Breakpoint 1, _start () at ControlStructure1.s:14
14              ldr r1,[r1]     @ load the value X into r1
(gdb) stepi
16              cmp r1,#3       @ compare the value in r1 (1) with 3
(gdb) i r
r0             0x0      0
r1             0x1      1
r2             0x0      0
r3             0x0      0
r4             0x0      0
r5             0x0      0
r6             0x0      0
r7             0x0      0
r8             0x0      0
r9             0x0      0
r10            0x0      0
r11            0x0      0
r12            0x0      0
sp             0x7efff070       0x7efff070
lr             0x0      0
pc             0x1007c  0x1007c <_start+8>
cpsr           0x10     16
(gdb) 
```

The program is run, the breakpoint is on line 14, the r1 corresponds to the value of the variable X, which is 1. The hex value is 0x1, the decimal is 1.

Part 3. Continued

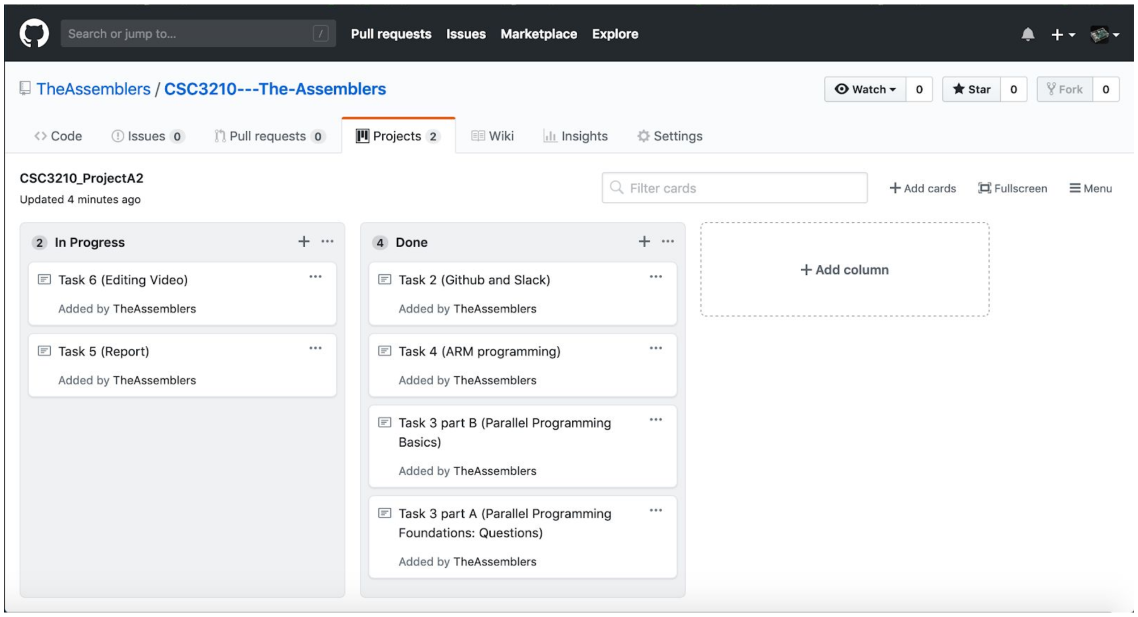


```
(gdb) run
Starting program: /home/pi/ControlStructure1

Breakpoint 1, _start () at ControlStructure1.s:14
14              ldr r1,[r1]     @ load the value X into r1
(gdb) stepi
16              cmp r1,#3       @ compare the value in r1 (1) with 3
(gdb) stepi
17              bgt else        @ branch to else when X value <=3
(gdb) stepi
18              sub r1,r1,#1    @ x=x-1
(gdb) stepi
19              b   done        @ branch to done
(gdb) stepi
done () at ControlStructure1.s:22
22              mov r7,#1       @ Program Termination:exit syscall
(gdb) stepi
23              svc #0          @ Program Termination: wake kernel
(gdb) i r
r0             0x0      0
r1             0x0      0
r2             0x0      0
r3             0x0      0
r4             0x0      0
r5             0x0      0
r6             0x0      0
r7             0x1      1
r8             0x0      0
r9             0x0      0
r10            0x0      0
r11            0x0      0
r12            0x0      0
sp             0x7efff070       0x7efff070
lr             0x0      0
pc             0x10094  0x10094 <done+4>
cpsr           0x80000010       -2147483632
(gdb) 
```

The program has run. The r1 register where X is located has hexadecimal value of 0x0 since X = 1-1 equals 0. The cpsr is 0x80000010, which is 1000 0000 0000 0000 0000 0000 0001 0000. The 31:28 bit field represents the conditions under which an instruction should be executed. In this case, it is 1000.The Z value is 0, which is the 30th bit. The result of the operation is 0.

Appendix

Youtube Channel: The Assemblers
Assignment Video: https://youtu.be/ku1L-v0E6jY
Slack: https://the-assemblers.slack.com/messages/CFSQ2GTDX/
Github: https://github.com/TheAssemblers



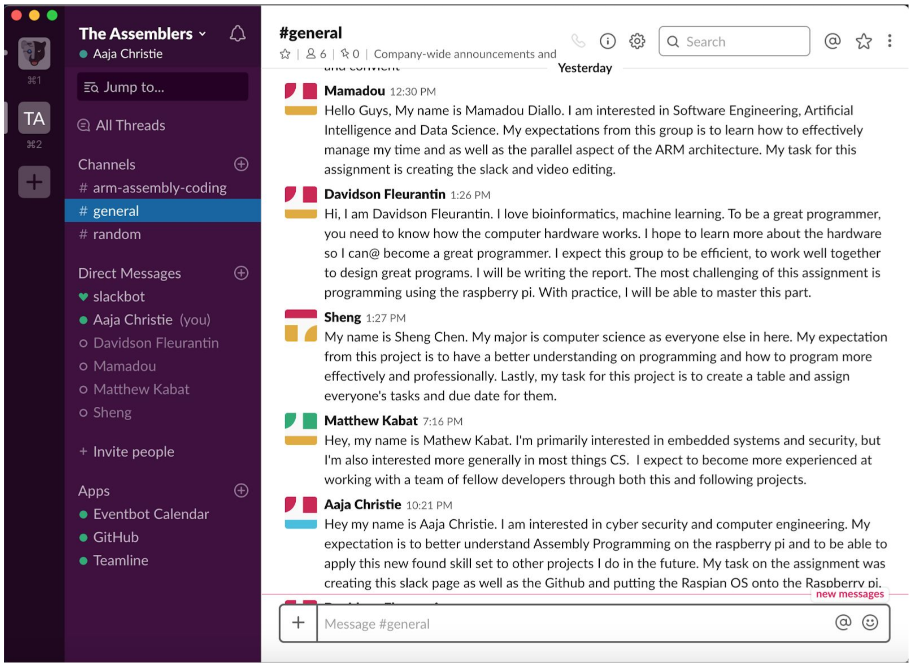
CSC3210_ProjectA2
Updated 4 minutes ago

2 In Progress

- Task 6 (Editing Video) — Added by TheAssemblers
- Task 5 (Report) — Added by TheAssemblers

4 Done

- Task 2 (Github and Slack) — Added by TheAssemblers
- Task 4 (ARM programming) — Added by TheAssemblers
- Task 3 part B (Parallel Programming Basics) — Added by TheAssemblers
- Task 3 part A (Parallel Programming Foundations: Questions) — Added by TheAssemblers



#general
6 | 0 | Company-wide announcements and

Yesterday

Mamadou 12:30 PM
Hello Guys, My name is Mamadou Diallo. I am interested in Software Engineering, Artificial Intelligence and Data Science. My expectations from this group is to learn how to effectively manage my time and as well as the parallel aspect of the ARM architecture. My task for this assignment is creating the slack and video editing.

Davidson Fleurantin 1:26 PM
Hi, I am Davidson Fleurantin. I love bioinformatics, machine learning. To be a great programmer, you need to know how the computer hardware works. I hope to learn more about the hardware so I can@ become a great programmer. I expect this group to be efficient, to work well together to design great programs. I will be writing the report. The most challenging of this assignment is programming using the raspberry pi. With practice, I will be able to master this part.

Sheng 1:27 PM
My name is Sheng Chen. My major is computer science as everyone else in here. My expectation from this project is to have a better understanding on programming and how to program more effectively and professionally. Lastly, my task for this project is to create a table and assign everyone's tasks and due date for them.

Matthew Kabat 7:16 PM
Hey, my name is Mathew Kabat. I'm primarily interested in embedded systems and security, but I'm also interested more generally in most things CS. I expect to become more experienced at working with a team of fellow developers through both this and following projects.

Aaja Christie 10:21 PM
Hey my name is Aaja Christie. I am interested in cyber security and computer engineering. My expectation is to better understand Assembly Programming on the raspberry pi and to be able to apply this new found skill set to other projects I do in the future. My task on the assignment was creating this slack page as well as the Github and putting the Raspian OS onto the Raspberry pi.