Developing Soft and Parallel Programming Skills Using Project-Based Learning

Spring 2019, The Raspberry Five

Jacques Agbenu, Marcus Aqui, Bonnie Atelsek, Zach Benator, Jay Patel

Planning and Scheduling

We used slack to assign to assign the sperate tasks in assignment 4. It was decided that Marcus would do the Arm Programming task, Jay would update the GitHub and create the table that would outline all the tasks that we would do, Zach would complete the parallel programming task, and Bonnie would shoot, edit and upload the team video to YouTube. We agreed to meet on 03/26 to go over the ARM programming and the Parallel programming.

We met in the library in a reserved study room. Some of us were late because of classes and distance from the campus. During the wait for the members of the group to assemble, the parallel programming questions were finished by Jacques, Bonnie and Zach. Because there was not a tv in the study room, Jacques had to reserve a pc stand in the curve at 6 o'clock, so we could use our raspberry pi to go over the programming. One of us had a test to study for so they had to leave before 6 o'clock to study for it.

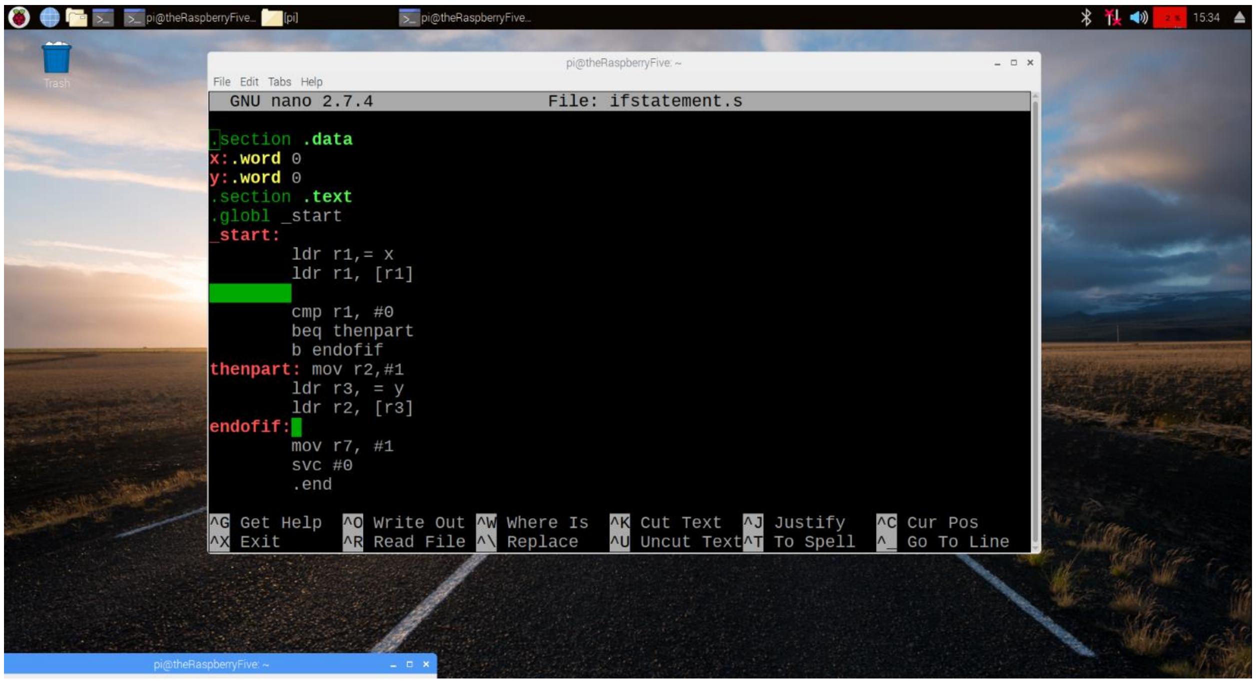
In the curve we set up the raspberry pi and went over the ARM programming and the Parallel Programming. We went over the Parallel programming that Zach oversaw first. There were some issues that needed to be solved in a few sections of the programming but other than that, there didn't seem to be an issue with the programming. Zach hadn't finished the report but was going to finish it by the time the main report was due or before. We then looked at the ARM programming that Marcus oversaw. He finished his report and we looked at his logic that he wrote for the questions.

Teamwork Basics

There was a bit of a lack of communication with this assignment. It wasn't the fault of anybody; Spring break was a very unwelcome distraction that broke the flow of the team. The two members of the team who were working on the programming were not able to transfer the raspberry pi between themselves resulting in one of the two to do their part of the programming later than was intended. Mostly everything was done before the final report was due and the parts that weren't due are being worked on. Everyone had a job to do and they worked on them to their fullest.

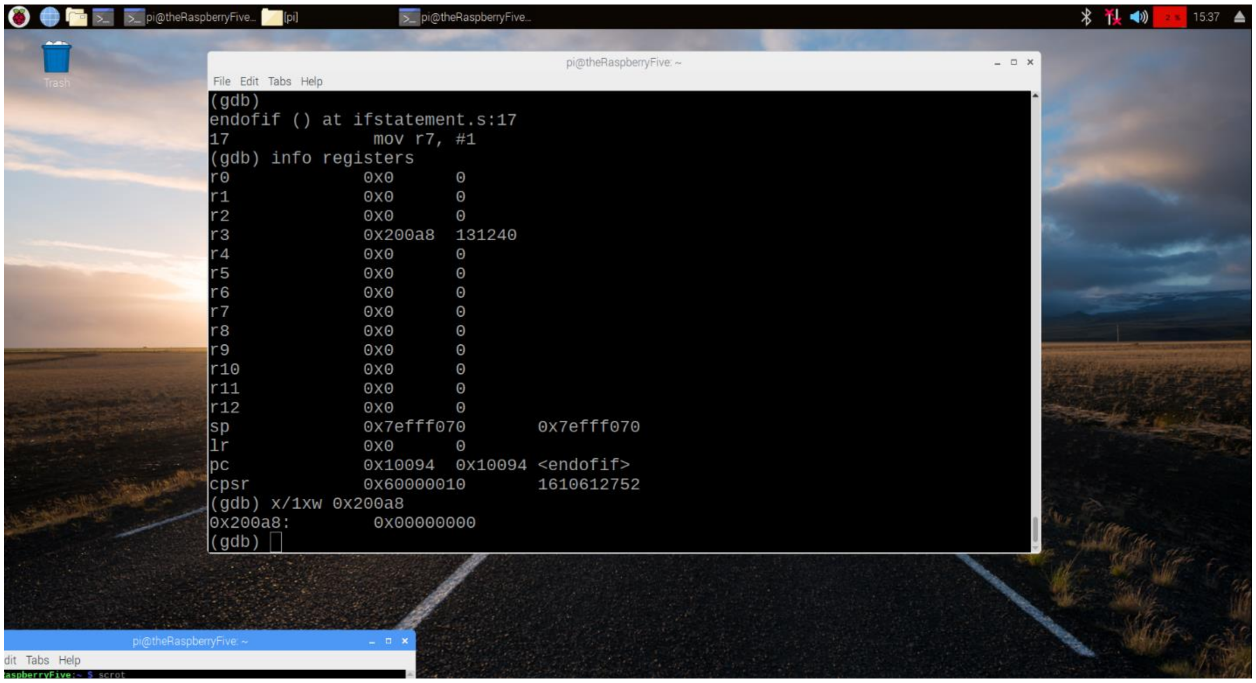
ARM Assembly Programming

Marcus completed the ARM Programming over the break and we looked at it when we met up at the library. We started out looking at the first problem of the code which called for a simple if statement that looked at the value of x and gave an output whether x was equal to zero or not. He decoded that logic into the ARM assembly language.



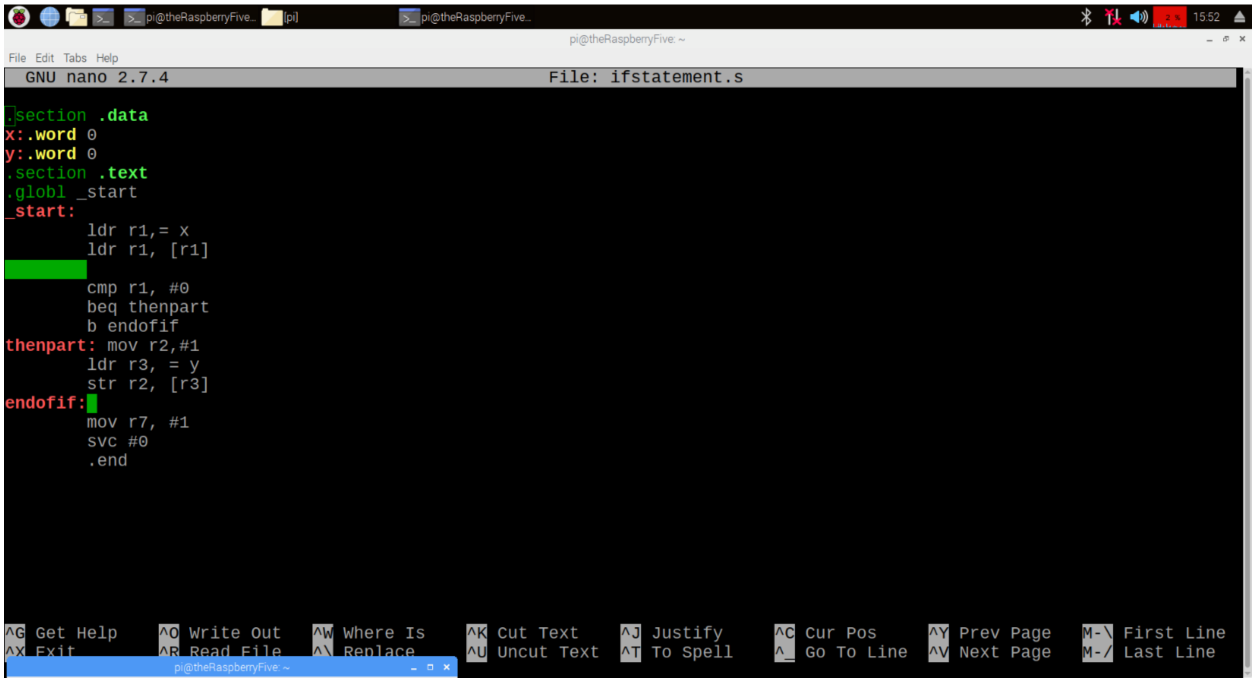
```
GNU nano 2.7.4                    File: ifstatement.s

.section .data
x:.word 0
y:.word 0
.section .text
.globl _start
_start:
        ldr r1,= x
        ldr r1, [r1]

        cmp r1, #0
        beq thenpart
        b endofif
thenpart: mov r2,#1
        ldr r3, = y
        ldr r2, [r3]
endofif:
        mov r7, #1
        svc #0
        .end

^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
^X Exit      ^R Read File ^\ Replace   ^U Uncut Text^T To Spell  ^_ Go To Line
```

He found an issue with a bit of the code as he kept on getting an error that made the result 0 when is should have been one.



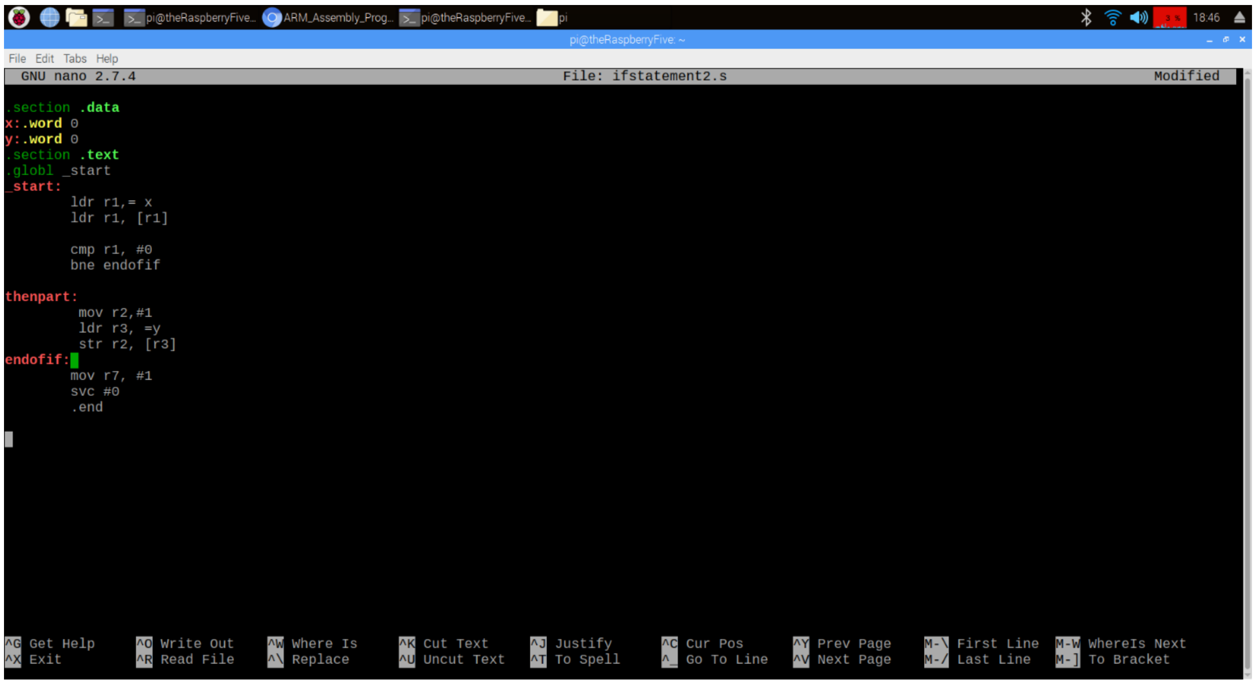
```
(gdb)
endofif () at ifstatement.s:17
17              mov r7, #1
(gdb) info registers
r0             0x0      0
r1             0x0      0
r2             0x0      0
r3             0x200a8  131240
r4             0x0      0
r5             0x0      0
r6             0x0      0
r7             0x0      0
r8             0x0      0
r9             0x0      0
r10            0x0      0
r11            0x0      0
r12            0x0      0
sp             0x7efff070       0x7efff070
lr             0x0      0
pc             0x10094  0x10094 <endofif>
cpsr           0x60000010       1610612752
(gdb) x/1xw 0x200a8
0x200a8:        0x00000000
(gdb)
```

So, he decided to change the ldr in the second part of the thenpart to a str instead and that seemed to fix the issue.



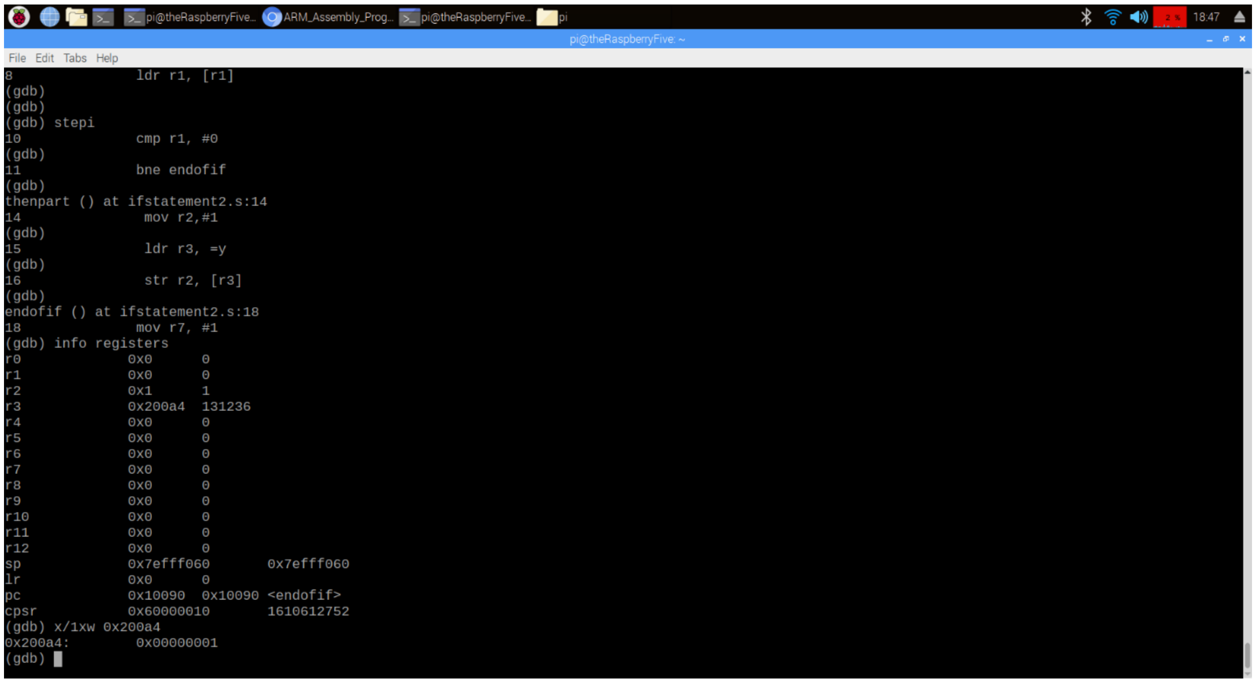
```
  GNU nano 2.7.4                              File: ifstatement.s

.section .data
x:.word 0
y:.word 0
.section .text
.globl _start
_start:
        ldr r1,= x
        ldr r1, [r1]

        cmp r1, #0
        beq thenpart
        b endofif
thenpart: mov r2,#1
        ldr r3, = y
        str r2, [r3]
endofif:
        mov r7, #1
        svc #0
        .end
```

For the second part of the code he had to get rid of the second b in the start section and change the beq to a bne. When we looked at the code when we gathered we saw that he left the thenpart where it was, and the code kept on running like It normally should. It didn't look right since in that case it didn't count for that if it was not equal then it should have gone to the endofif section instead of the thenpart section. We changed that to make it more logically sound.



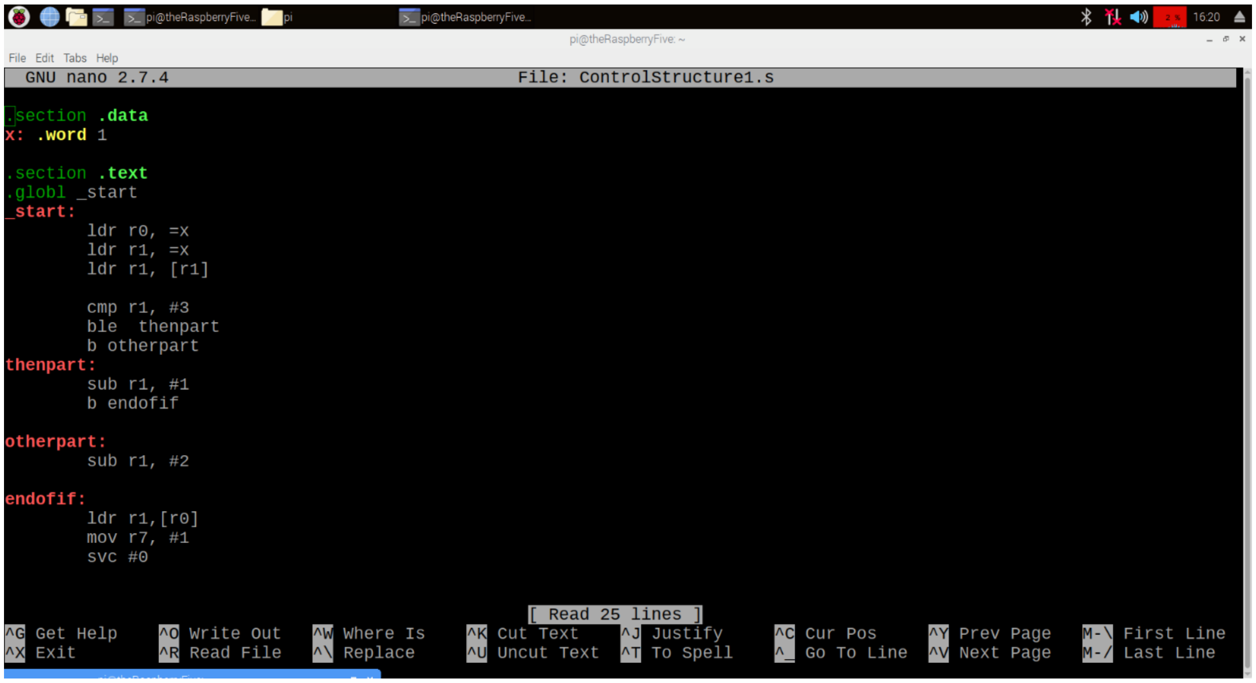
```
GNU nano 2.7.4                  File: ifstatement2.s                  Modified

.section .data
x:.word 0
y:.word 0
.section .text
.globl _start
_start:
        ldr r1,= x
        ldr r1, [r1]

        cmp r1, #0
        bne endofif

thenpart:
         mov r2,#1
         ldr r3, =y
         str r2, [r3]
endofif:
        mov r7, #1
        svc #0
        .end

^G Get Help   ^O Write Out   ^W Where Is   ^K Cut Text    ^J Justify    ^C Cur Pos     ^Y Prev Page   M-\ First Line   M-W WhereIs Next
^X Exit       ^R Read File   ^\ Replace    ^U Uncut Text  ^T To Spell   ^_ Go To Line  ^V Next Page   M-/ Last Line    M-] To Bracket
```

The results for that ne were still the same as if we hadn't made it logically sound.

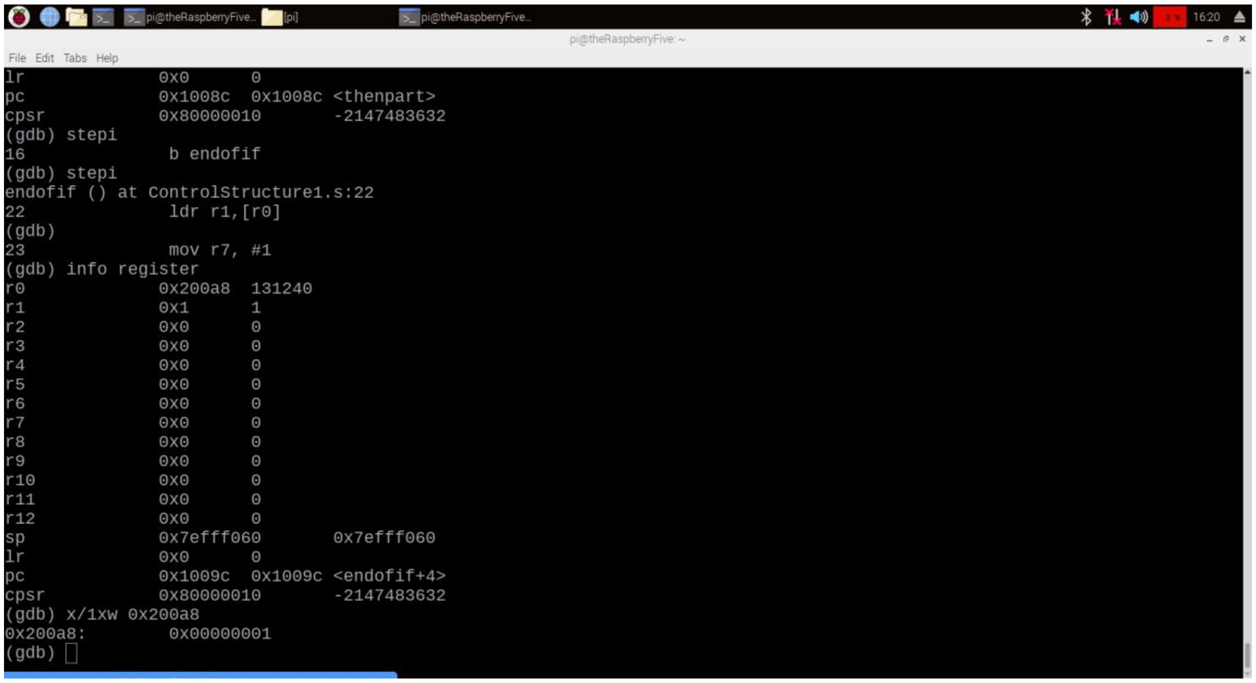


```
8               ldr r1, [r1]
(gdb)
(gdb)
(gdb) stepi
10              cmp r1, #0
(gdb)
11              bne endofif
(gdb)
thenpart () at ifstatement2.s:14
14               mov r2,#1
(gdb)
15               ldr r3, =y
(gdb)
16               str r2, [r3]
(gdb)
endofif () at ifstatement2.s:18
18              mov r7, #1
(gdb) info registers
r0             0x0      0
r1             0x0      0
r2             0x1      1
r3             0x200a4  131236
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             0x7efff060       0x7efff060
lr             0x0      0
pc             0x10090  0x10090 <endofif>
cpsr           0x60000010       1610612752
(gdb) x/1xw 0x200a4
0x200a4:        0x00000001
(gdb)
```

For the final part of the ARM programming, Marcus had to compare the value of x against 3. If it was less than or equal to 3, then x = x-1. If x was greater than 3 , then the new value of x is x - 2. The program ran as is should but since Marcus used code from the previous parts of the assignment, there was an error since there was a ldr in the end where there needed to be a str. The output was defiantly not what it should have been.

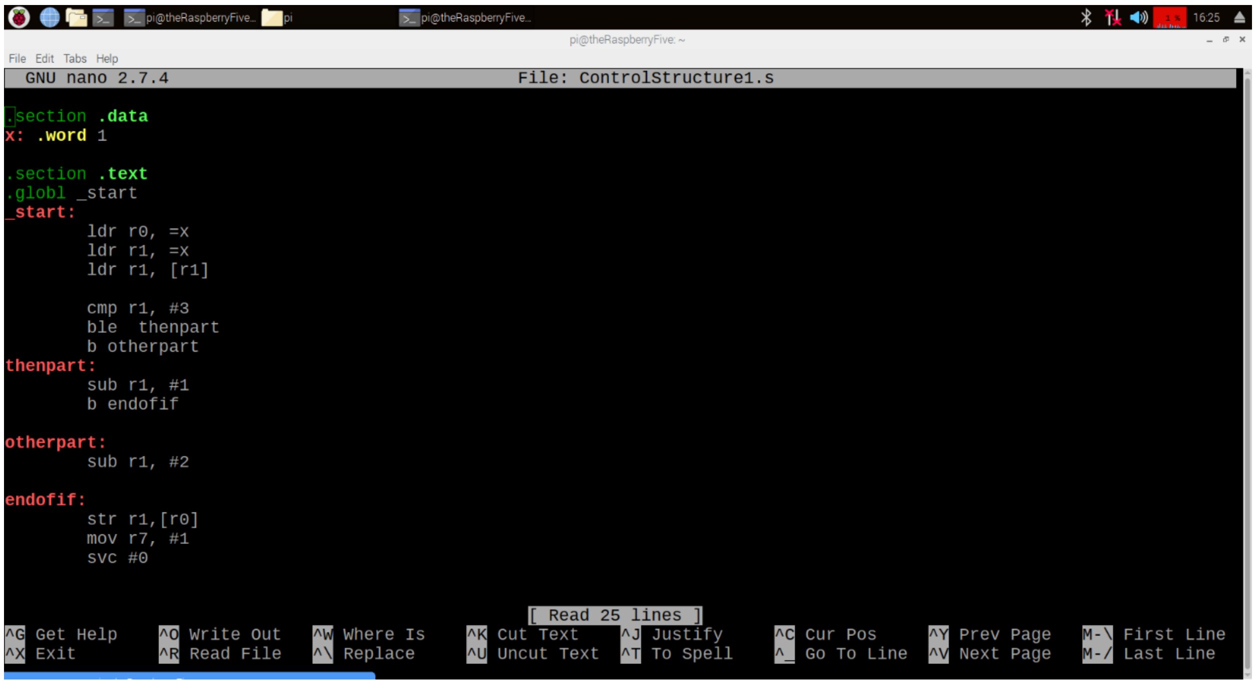


```
 GNU nano 2.7.4                              File: ControlStructure1.s

.section .data
x: .word 1

.section .text
.globl _start
_start:
        ldr r0, =x
        ldr r1, =x
        ldr r1, [r1]

        cmp r1, #3
        ble  thenpart
        b otherpart
thenpart:
        sub r1, #1
        b endofif

otherpart:
        sub r1, #2

endofif:
        ldr r1,[r0]
        mov r7, #1
        svc #0

                                            [ Read 25 lines ]
^G Get Help     ^O Write Out    ^W Where Is     ^K Cut Text     ^J Justify      ^C Cur Pos      ^Y Prev Page    M-\ First Line
^X Exit         ^R Read File    ^\ Replace      ^U Uncut Text   ^T To Spell     ^_ Go To Line   ^V Next Page    M-/ Last Line
```

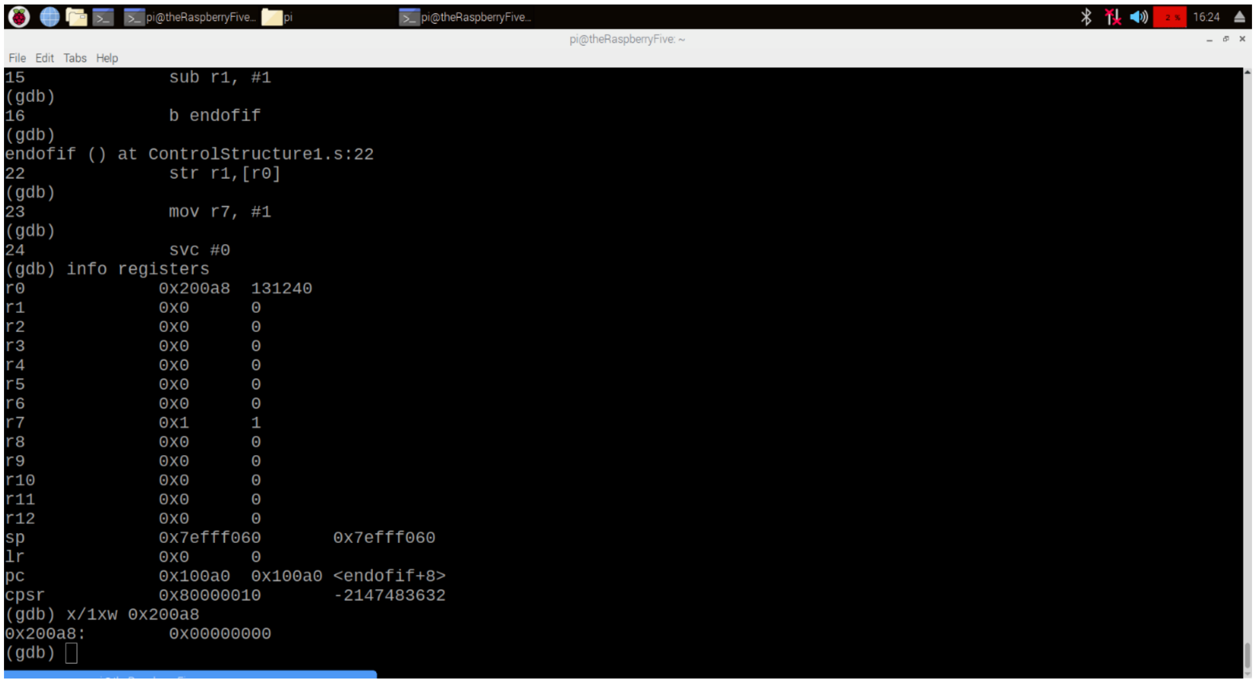


```
lr             0x0      0
pc             0x1008c  0x1008c <thenpart>
cpsr           0x80000010       -2147483632
(gdb) stepi
16              b endofif
(gdb) stepi
endofif () at ControlStructure1.s:22
22              ldr r1,[r0]
(gdb)
23              mov r7, #1
(gdb) info register
r0             0x200a8  131240
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             0x7efff060       0x7efff060
lr             0x0      0
pc             0x1009c  0x1009c <endofif+4>
cpsr           0x80000010       -2147483632
(gdb) x/1xw 0x200a8
0x200a8:        0x00000001
(gdb) 
```

He was able to fix the code by changing that ldr to a str and got it to work like it should have.



```
  GNU nano 2.7.4                                    File: ControlStructure1.s

.section .data
x: .word 1

.section .text
.globl _start
_start:
        ldr r0, =x
        ldr r1, =x
        ldr r1, [r1]

        cmp r1, #3
        ble  thenpart
        b otherpart
thenpart:
        sub r1, #1
        b endofif

otherpart:
        sub r1, #2

endofif:
        str r1,[r0]
        mov r7, #1
        svc #0

                                        [ Read 25 lines ]
^G Get Help    ^O Write Out   ^W Where Is    ^K Cut Text    ^J Justify     ^C Cur Pos     ^Y Prev Page   M-\ First Line
^X Exit        ^R Read File   ^\ Replace     ^U Uncut Text  ^T To Spell    ^_ Go To Line  ^V Next Page   M-/ Last Line
```

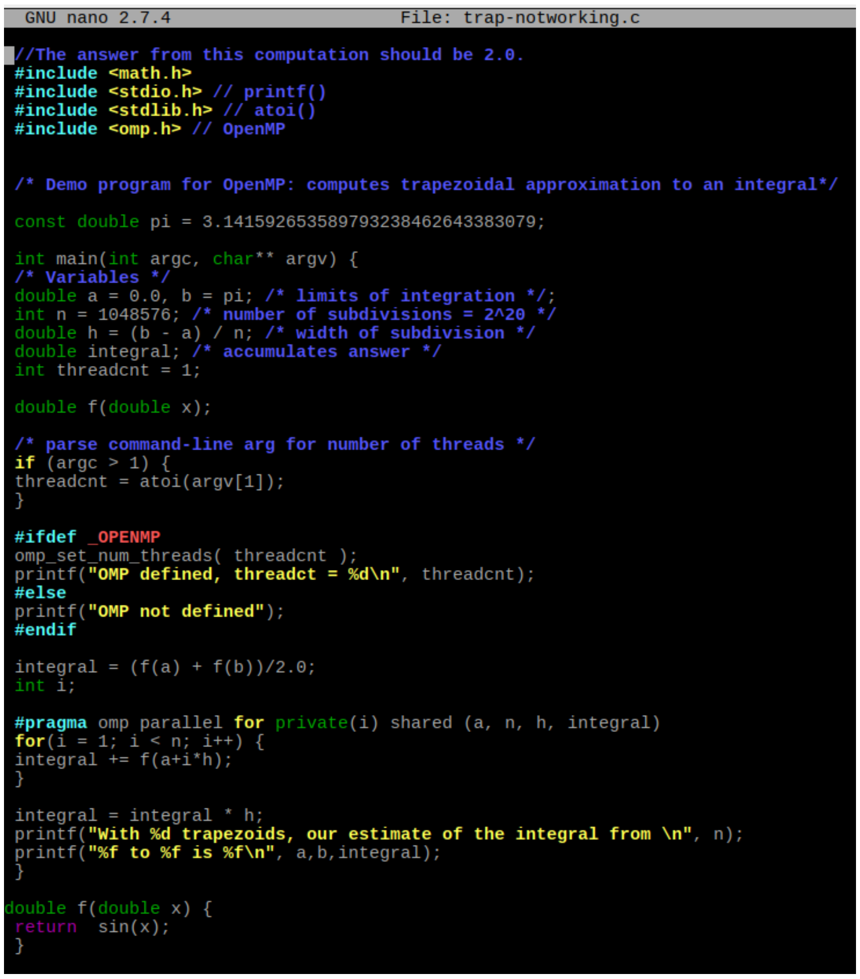


```
15              sub r1, #1
(gdb)
16              b endofif
(gdb)
endofif () at ControlStructure1.s:22
22              str r1,[r0]
(gdb)
23              mov r7, #1
(gdb)
24              svc #0
(gdb) info registers
r0             0x200a8  131240
r1             0x0      0
r2             0x0      0
r3             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             0x7efff060       0x7efff060
lr             0x0      0
pc             0x100a0  0x100a0 <endofif+8>
cpsr           0x80000010       -2147483632
(gdb) x/1xw 0x200a8
0x200a8:        0x00000000
(gdb) 
```

*unable to edit due to late entry, copied verbatim from group member Zach

Assignment 4 Parallel Programming

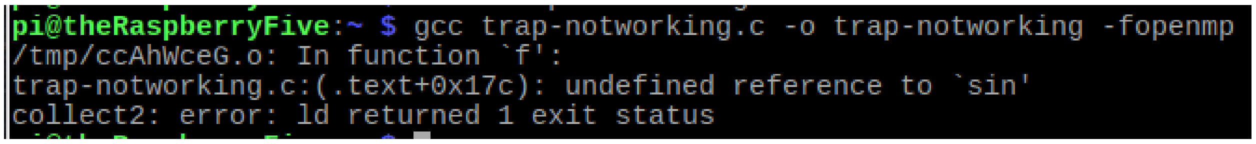
2: Integration Using the Trapezoidal Rule



```
GNU nano 2.7.4                    File: trap-notworking.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 = 2^20 */
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);
}
```

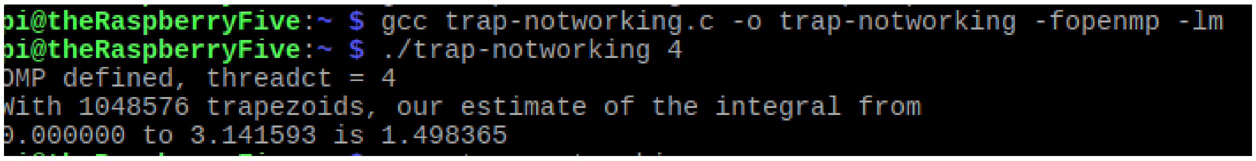
This is trap-notworking.c, which attempts to compute an approximation of $\int_0^{\pi} \sin(x)dx$ using parallel processing.

The following compile command fails because “sin(x)” is not recognized,



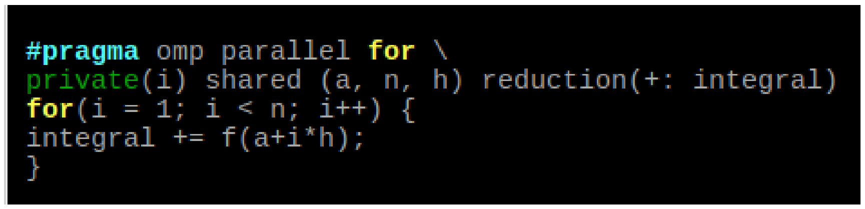
```
pi@theRaspberryFive:~ $ gcc trap-notworking.c -o trap-notworking -fopenmp
/tmp/ccAhWceG.o: In function `f':
trap-notworking.c:(.text+0x17c): undefined reference to `sin'
collect2: error: ld returned 1 exit status
```

but adding “-lm” to include the math library when compiling solves the problem and allows for trap-notworking.c to be compiled successfully.



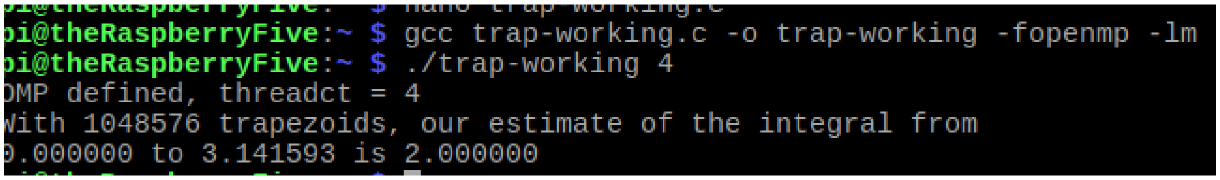
```
pi@theRaspberryFive:~ $ gcc trap-notworking.c -o trap-notworking -fopenmp -lm
pi@theRaspberryFive:~ $ ./trap-notworking 4
OMP defined, threadct = 4
With 1048576 trapezoids, our estimate of the integral from
0.000000 to 3.141593 is 1.498365
```

However, the program still gives an incorrect result. We need to add a reduction clause for the accumulator variable (integral) for the summation across parallel threads to work correctly.



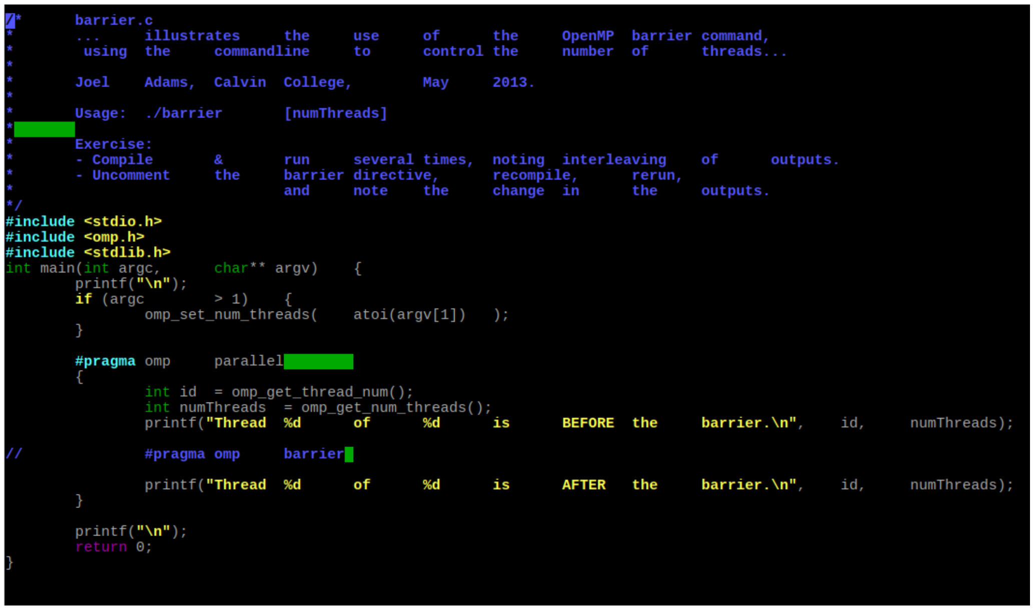
```
#pragma omp parallel for \
private(i) shared (a, n, h) reduction(+: integral)
for(i = 1; i < n; i++) {
integral += f(a+i*h);
}
```

Now the program calculates the correct result.



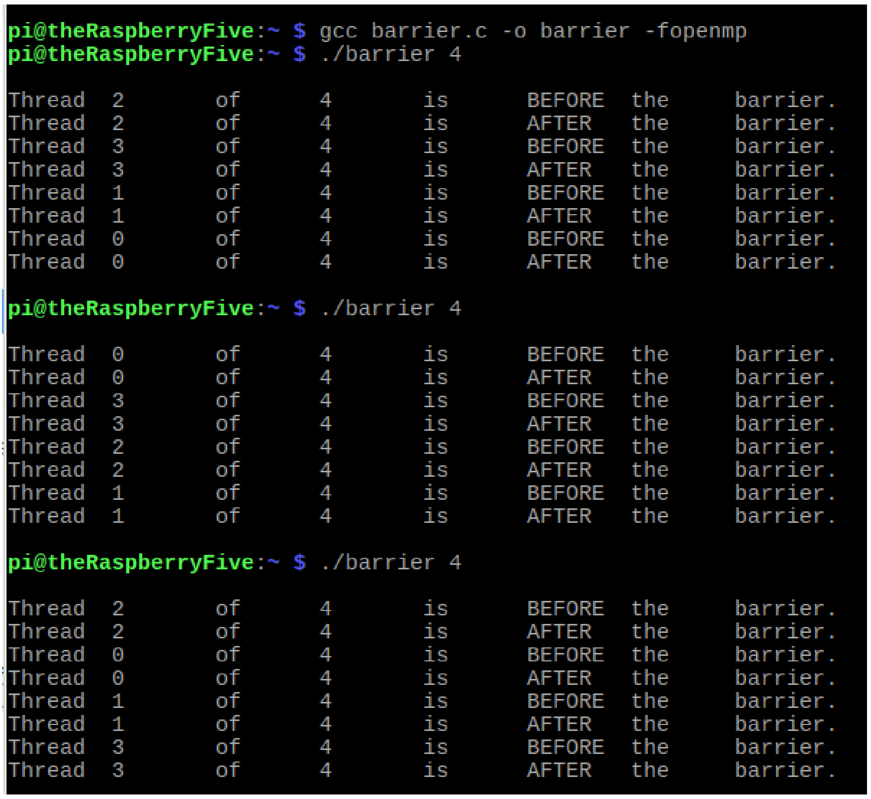
```
pi@theRaspberryFive:~ $ gcc trap-working.c -o trap-working -fopenmp -lm
pi@theRaspberryFive:~ $ ./trap-working 4
OMP defined, threadct = 4
With 1048576 trapezoids, our estimate of the integral from
0.000000 to 3.141593 is 2.000000
```

3: Synchronization with a Barrier



```
/*      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;
}
```

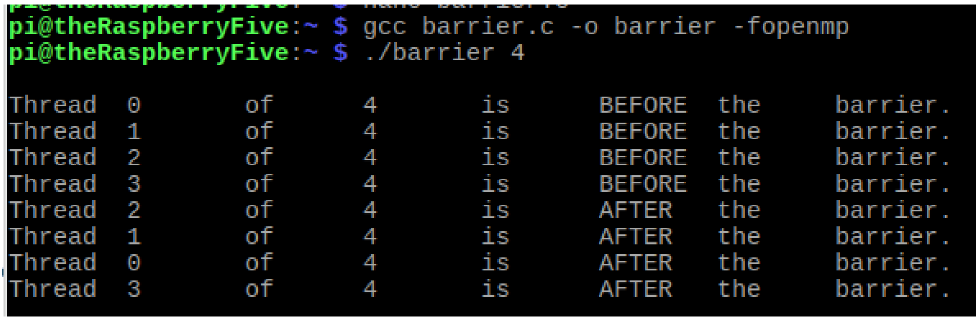
This program uses the barrier pattern (commented out in this image) to stop threads from continuing until they all reach the same point.



```
pi@theRaspberryFive:~ $ gcc barrier.c -o barrier -fopenmp
pi@theRaspberryFive:~ $ ./barrier 4

Thread  2       of      4       is      BEFORE  the     barrier.
Thread  2       of      4       is      AFTER   the     barrier.
Thread  3       of      4       is      BEFORE  the     barrier.
Thread  3       of      4       is      AFTER   the     barrier.
Thread  1       of      4       is      BEFORE  the     barrier.
Thread  1       of      4       is      AFTER   the     barrier.
Thread  0       of      4       is      BEFORE  the     barrier.
Thread  0       of      4       is      AFTER   the     barrier.

pi@theRaspberryFive:~ $ ./barrier 4

Thread  0       of      4       is      BEFORE  the     barrier.
Thread  0       of      4       is      AFTER   the     barrier.
Thread  3       of      4       is      BEFORE  the     barrier.
Thread  3       of      4       is      AFTER   the     barrier.
Thread  2       of      4       is      BEFORE  the     barrier.
Thread  2       of      4       is      AFTER   the     barrier.
Thread  1       of      4       is      BEFORE  the     barrier.
Thread  1       of      4       is      AFTER   the     barrier.

pi@theRaspberryFive:~ $ ./barrier 4

Thread  2       of      4       is      BEFORE  the     barrier.
Thread  2       of      4       is      AFTER   the     barrier.
Thread  0       of      4       is      BEFORE  the     barrier.
Thread  0       of      4       is      AFTER   the     barrier.
Thread  1       of      4       is      BEFORE  the     barrier.
Thread  1       of      4       is      AFTER   the     barrier.
Thread  3       of      4       is      BEFORE  the     barrier.
Thread  3       of      4       is      AFTER   the     barrier.
```

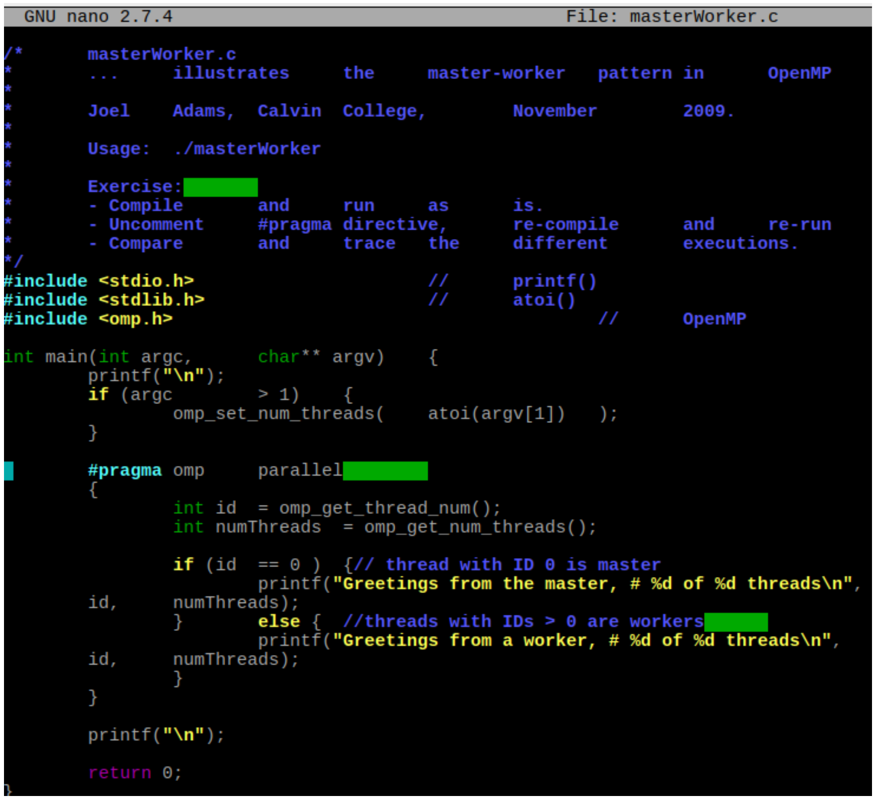
With the barrier commented out, each thread executes all its statements before the following thread begins.

But with the barrier active (no longer commented out), each thread stops at the barrier until all threads have reached it, and then the threads continue and execute the statement after the barrier.



```
pi@theRaspberryFive:~ $ gcc barrier.c -o barrier -fopenmp
pi@theRaspberryFive:~ $ ./barrier 4

Thread  0       of      4       is      BEFORE  the     barrier.
Thread  1       of      4       is      BEFORE  the     barrier.
Thread  2       of      4       is      BEFORE  the     barrier.
Thread  3       of      4       is      BEFORE  the     barrier.
Thread  2       of      4       is      AFTER   the     barrier.
Thread  1       of      4       is      AFTER   the     barrier.
Thread  0       of      4       is      AFTER   the     barrier.
Thread  3       of      4       is      AFTER   the     barrier.
```

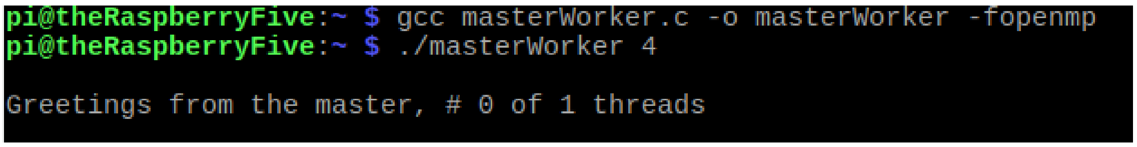
4: Program Structure: The Master-Worker Implementation Strategy



```
  GNU nano 2.7.4                                  File: masterWorker.c

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

        printf("\n");

        return 0;
}
```

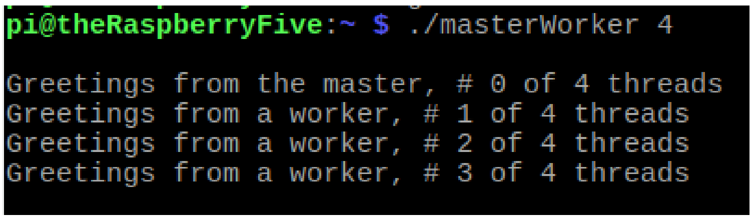
This program uses the master-worker pattern to allow one thread to execute one segment of code while the other three execute a different segment of code.

With the “#pragma omp parallel” commented out, only the “master’s” statements are executed,



```
pi@theRaspberryFive:~ $ gcc masterWorker.c -o masterWorker -fopenmp
pi@theRaspberryFive:~ $ ./masterWorker 4

Greetings from the master, # 0 of 1 threads
```

But with the #pragma included in the code, the first thread executes the “master” statement while the other 3 execute the “worker” statement.



```
pi@theRaspberryFive:~ $ ./masterWorker 4

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

Appendix

Slack Link: https://raspberryfive.slack.com/messages/CFQUGDXMX/details/

Username: scholarpanther3@gmail.com
Password: GetTh1s1010
(if needed to log into the slack account)

GitHub Link: https://github.com/theRaspberryFive

YouTube Link: https://www.youtube.com/watch?v=JipuB4sV3VQ