**Module: R1: C Programming**

**Section:** C Arrays, Pointers & Strings **Task:** 2.2

**Task 2.2**

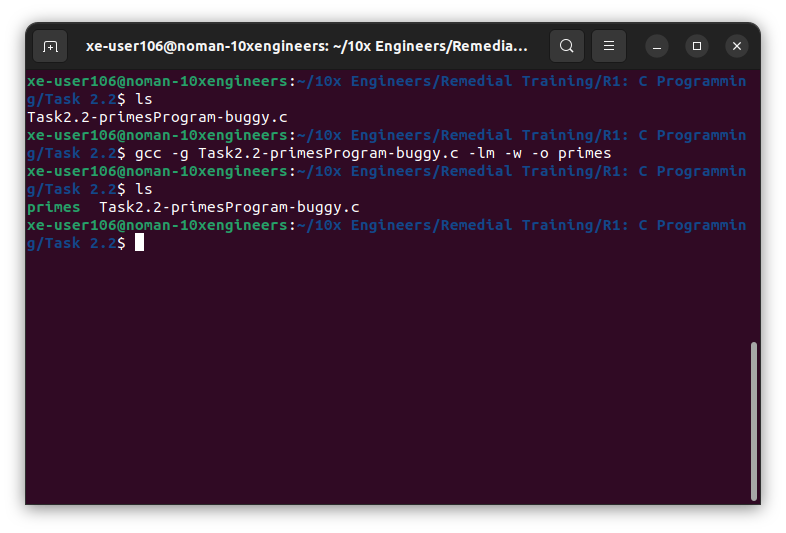
**GDB**

**Prime Numbers:‬**

‭First of all, we need to compile the given program using gcc:‬

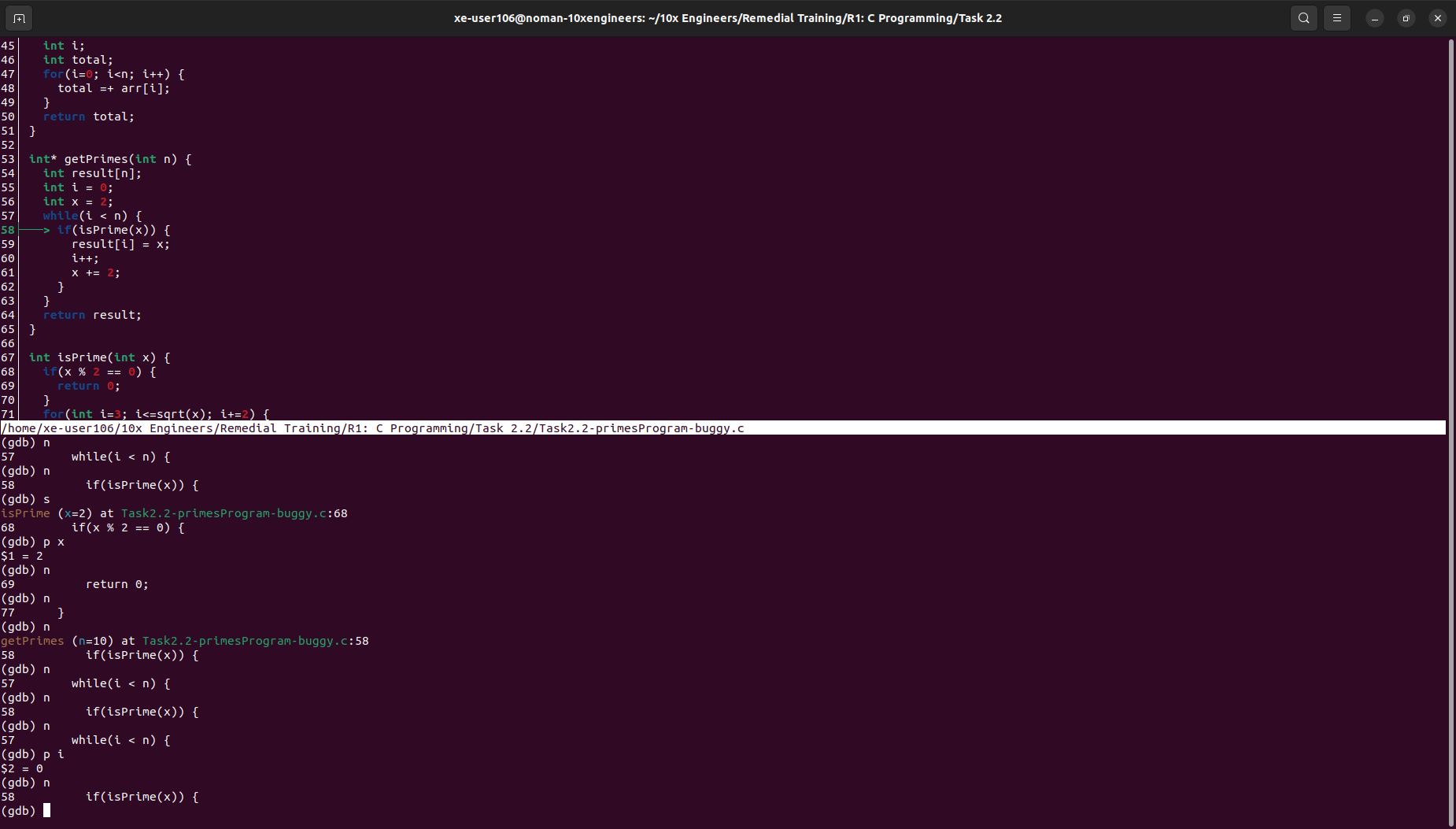
‭

*gcc -g Task2.2-primesProgram-buggy.c -lm -w -o primes*

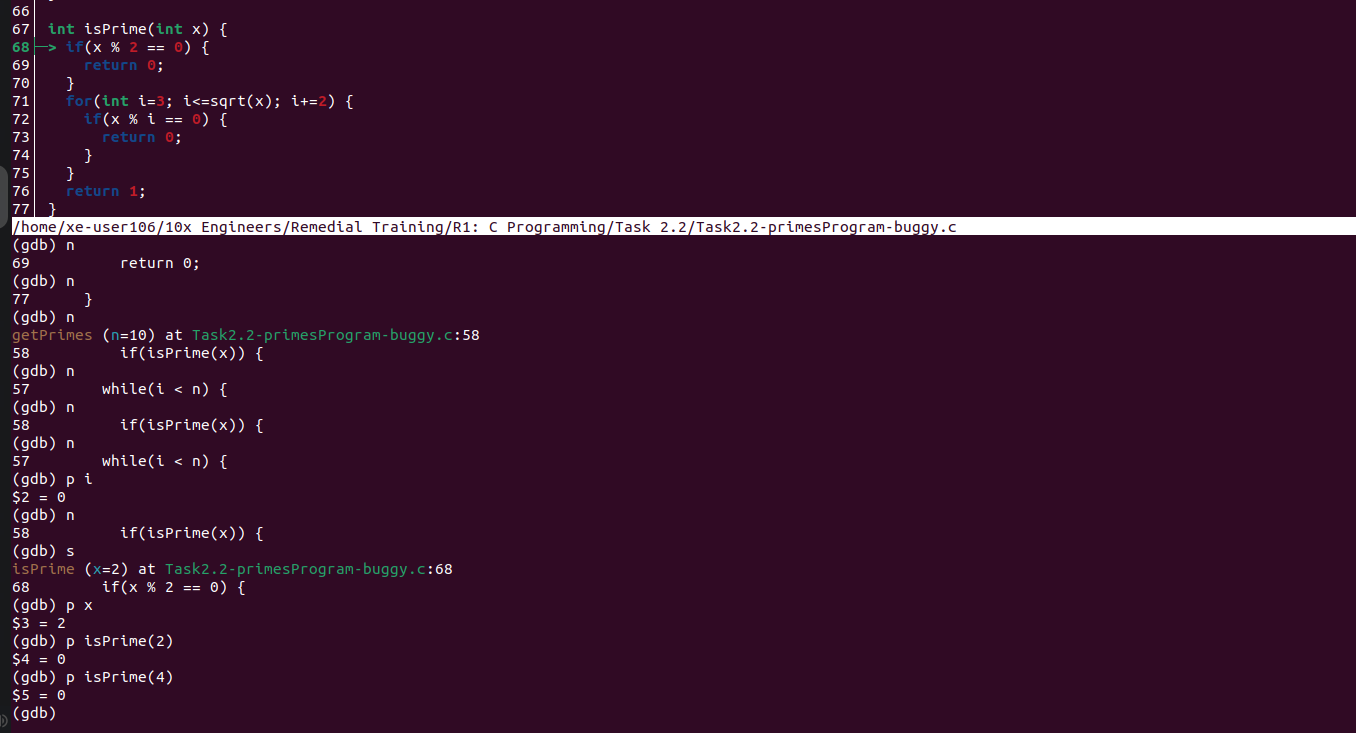
****

Opened the ‬‭cgdb‬‭ to debug our program. The program appears to be stuck inside‬

**‭getPrimes( )‬** ‭function. The value of x denotes that we are stuck at first iteration of the loop:



Stepped into **isPrime** function to investigate the issue:



‭The function is returning false to discard even numbers, however, 2 itself is a prime‬ ‭number. In order to accommodate this, we will need to modify our logic a bit. Here’s the‬

‭revised logic:

‭int isPrime(int x) {‬

i‭f (x < 2){‬

‭return 0;‬

‭}‬

‭ else if (x == 2){‬

‭ return 1;‬

‭}‬

‭else if(x % 2 == 0) {‬

‭return 0;‬

‭}‬

‭for(int i=3; i<=sqrt(x); i+=2) {‬

‭if(x % i == 0) {‬

‭return 0;‬

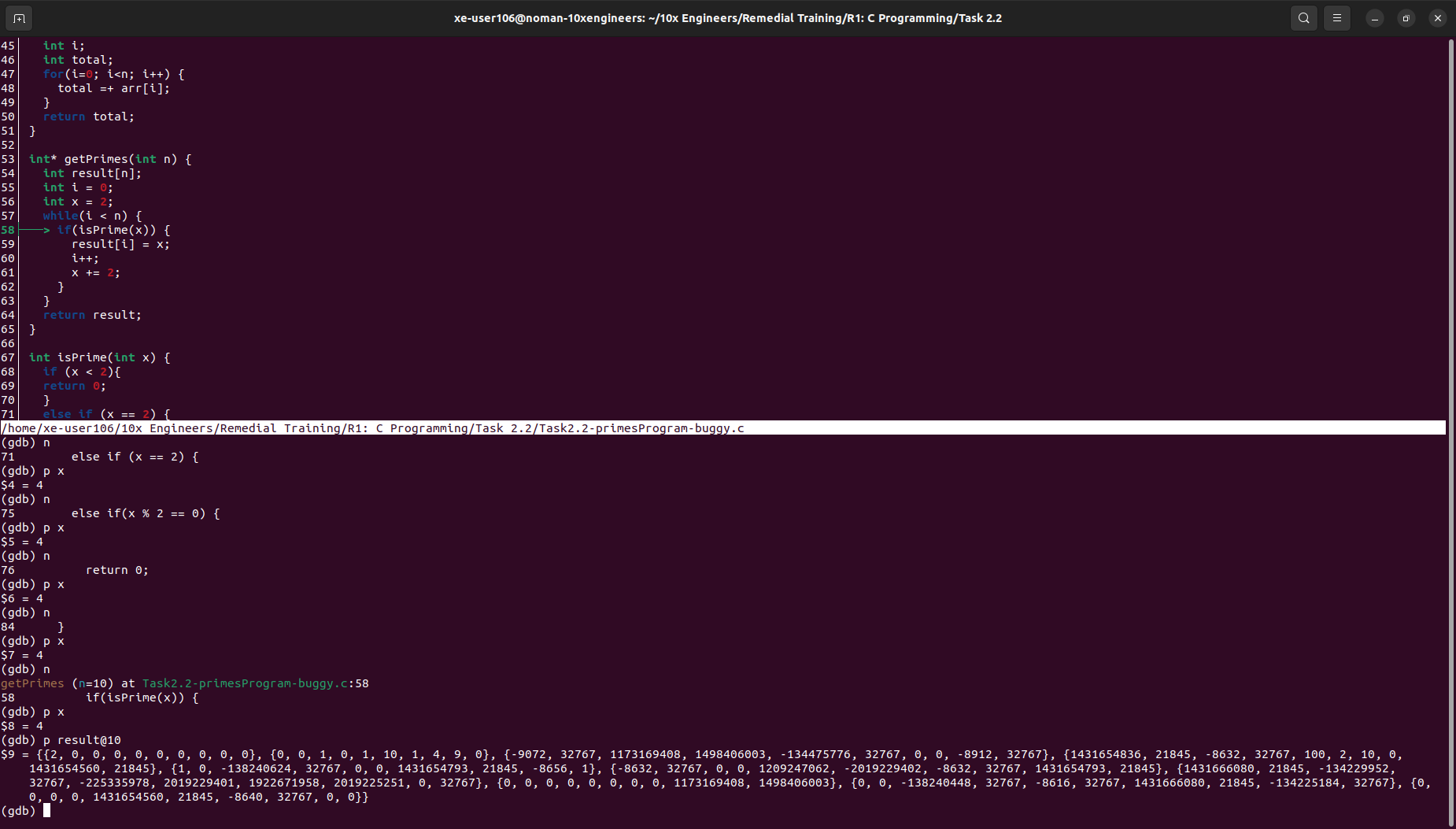
‭}‬

‭}‬

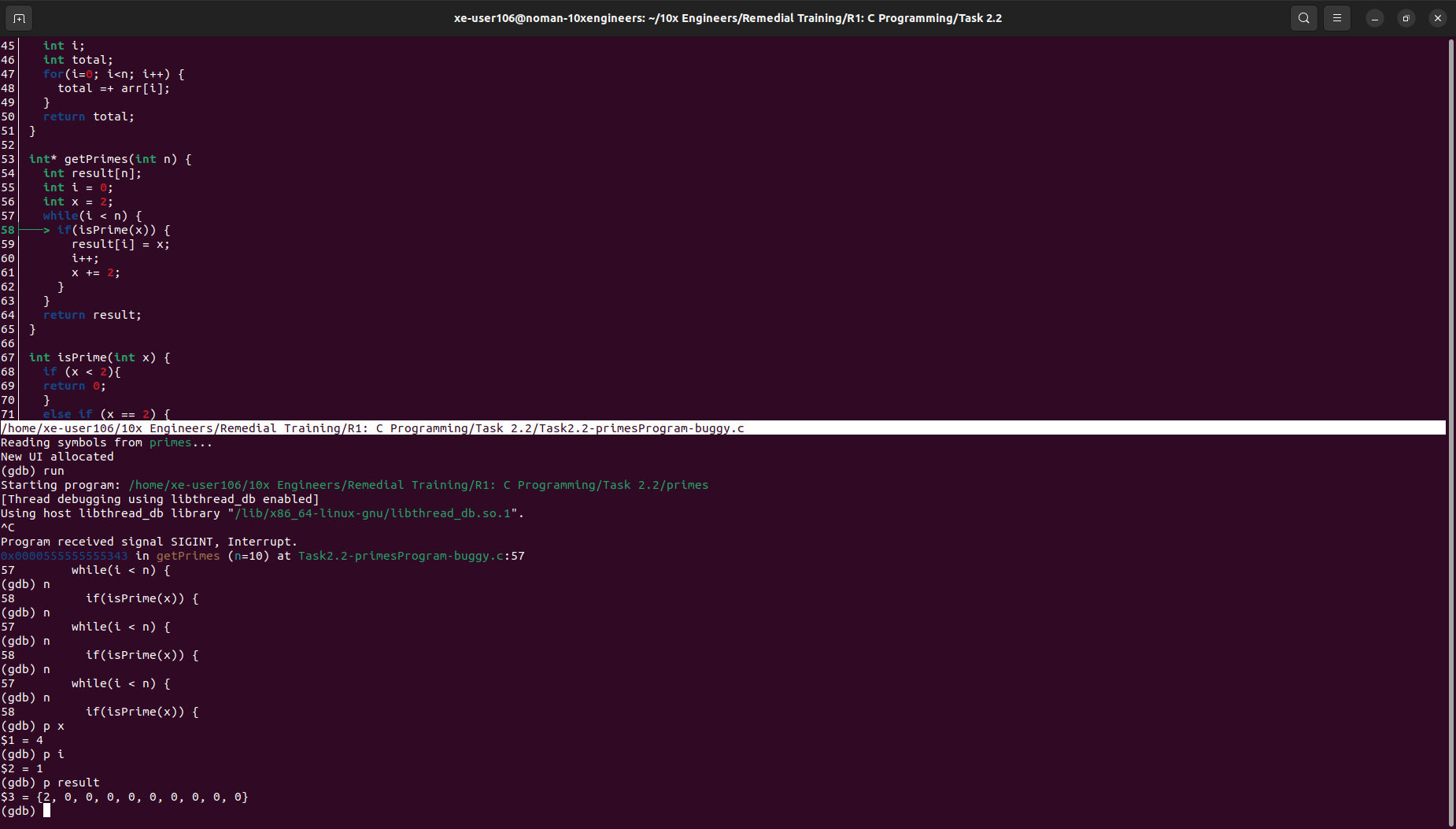
‭return 1;‬

‭}‬

‭Recompiled the program and the problem still exists. Opened the debugger again. The‬ ‭program appears to be stuck inside while loop.



Stepped into‬ **‭isPrime( )** ‬‭function again. The first prime number 2 has been stored‬ ‭successfully. And the number x has been incremented to 4 now. Number 3 has been‬ ‭skipped:‬



‭

Added a break at ‬‭getPrimes.‬‭The first time we entered the loop. In order to‬ accommodate for 3, we need to alter our logic and start x from 3:‬

‭

int\* getPrimes(int n) {‬

‭ int result[n];‬

‭result[0]=2;‬

‭int i = 1;‬

‭int x = 3;‬

‭while(i < n) {‬

‭ if(isPrime(x)) {‬

‭result[i] = x;‬

‭i++;‬

‭x += 2;‬

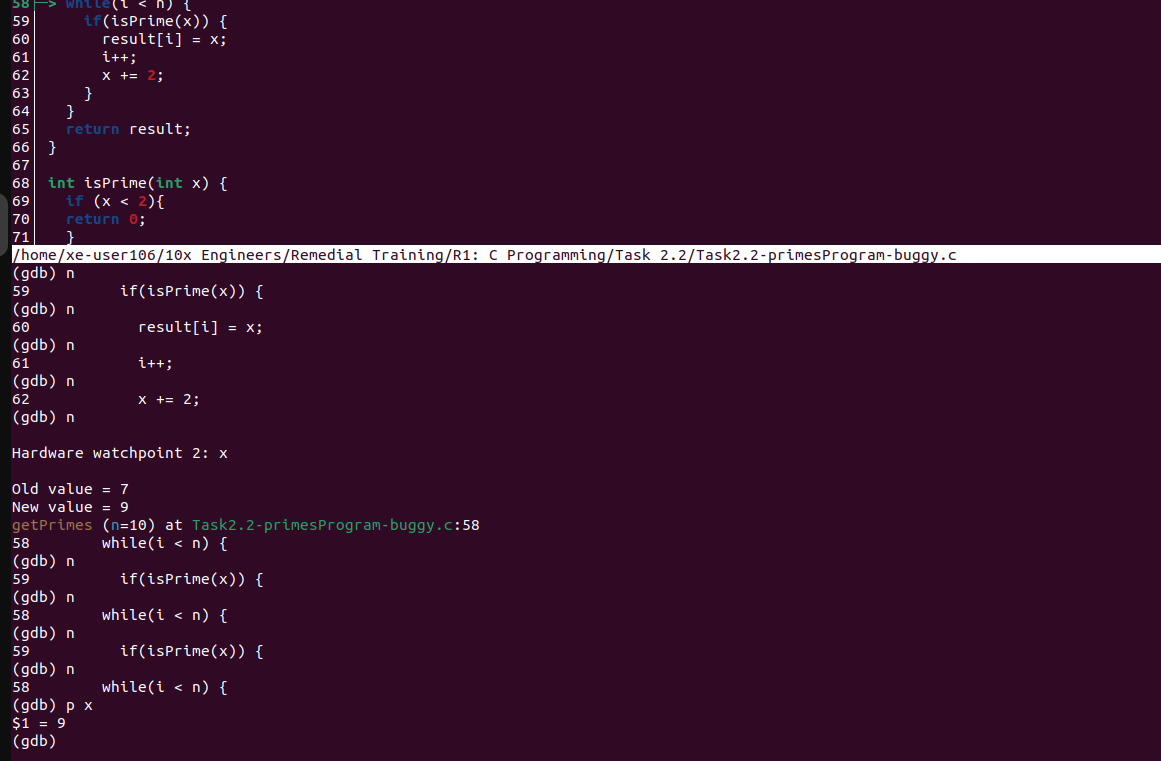
‭ }‬

‭ }‬

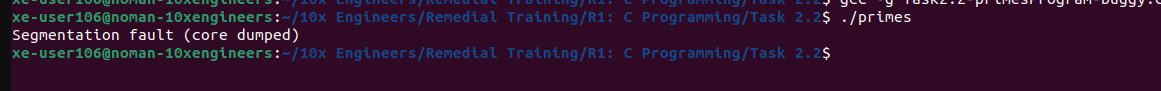
‭return result;‬

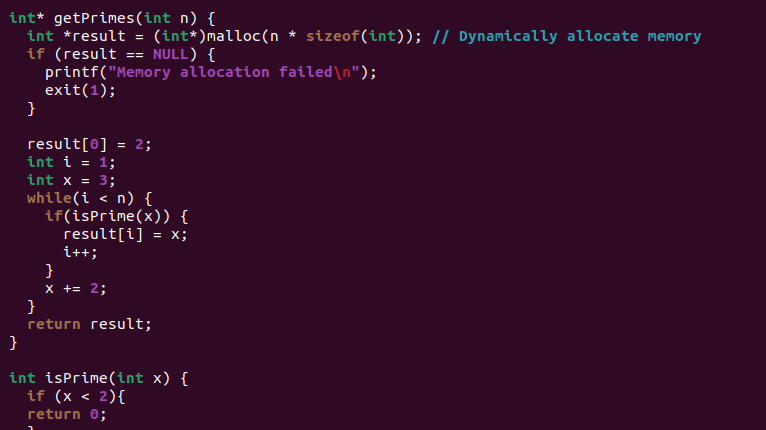
‭}‬

Recompiling again gives the same error again. Debugging again, the programs gets‬ stuck when x=9:

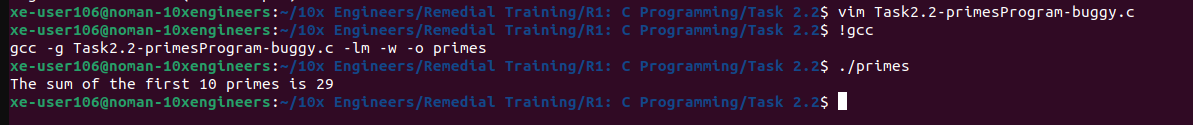
****

Moved the x increment‬‭ **x+=2‬** ‭outside the while loop which resulted in segmentation‬ ‭fault. The program is accessing memory which is null so we need to first allocate‬ memory for our program to access. We’ll be dynamically allocate this memory.

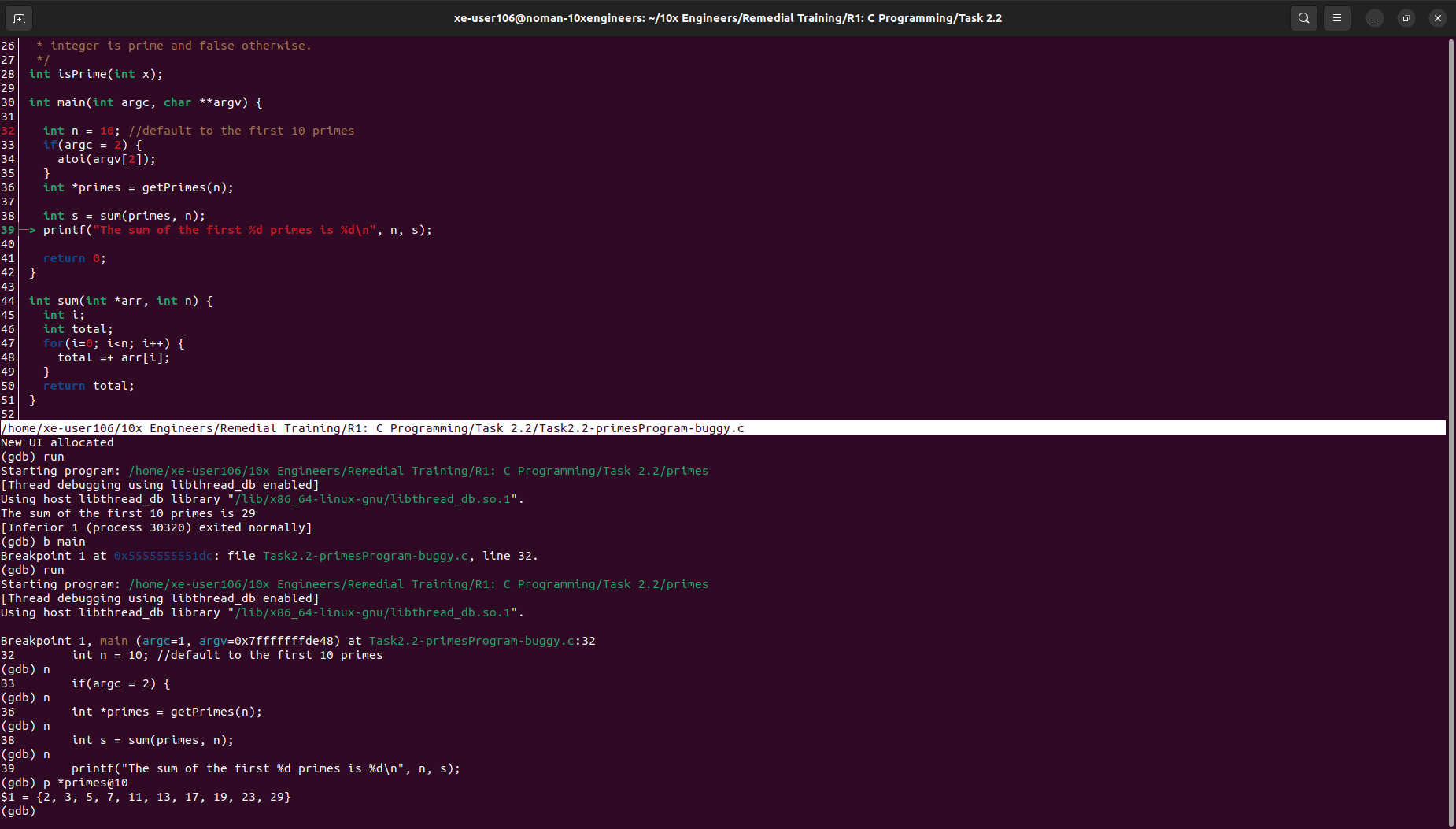
****



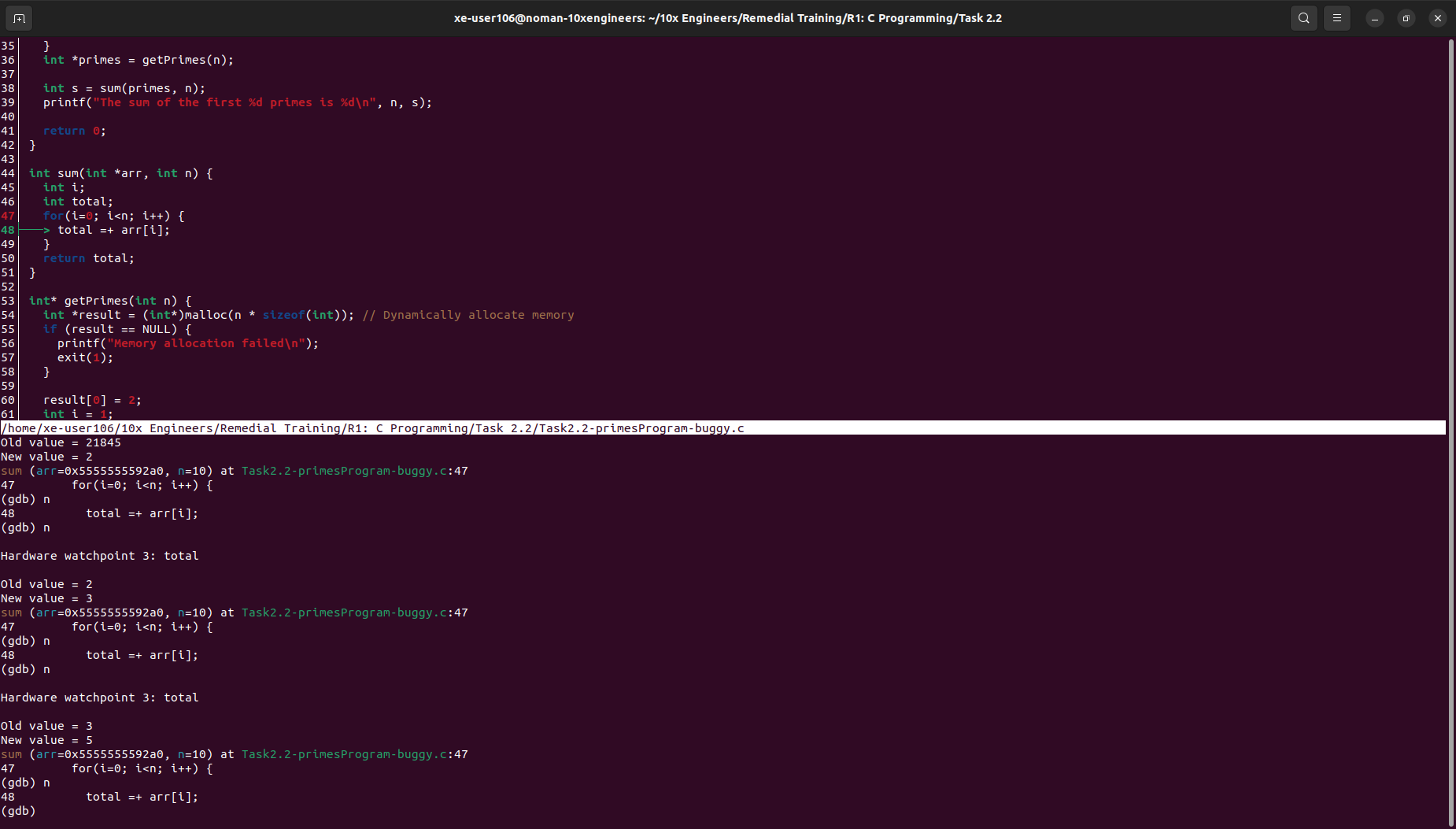
‭Recompiled the program and it has finally worked. The program is now working but the‬ ‭output is not right this time.



Opened with debugger and added a break at main to investigate further. The prime‬ ‭numbers have been calculated correctly:



The sum has not been calculated right. Added a break at sum this time. Instead of‬ ‭updating the total, the program is assigning the prime number to the total in **sum( )**‬ ‭function.



It will take the following correction to avoid any garbage values:  
int sum(int \*arr, int n) {‬

‭int i;‬

‭int total = 0;‬

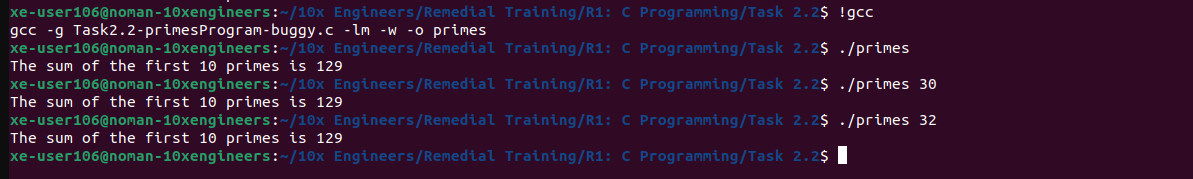
‭for(i=0; i<n; i++) {‬

‭total += arr[i];‬

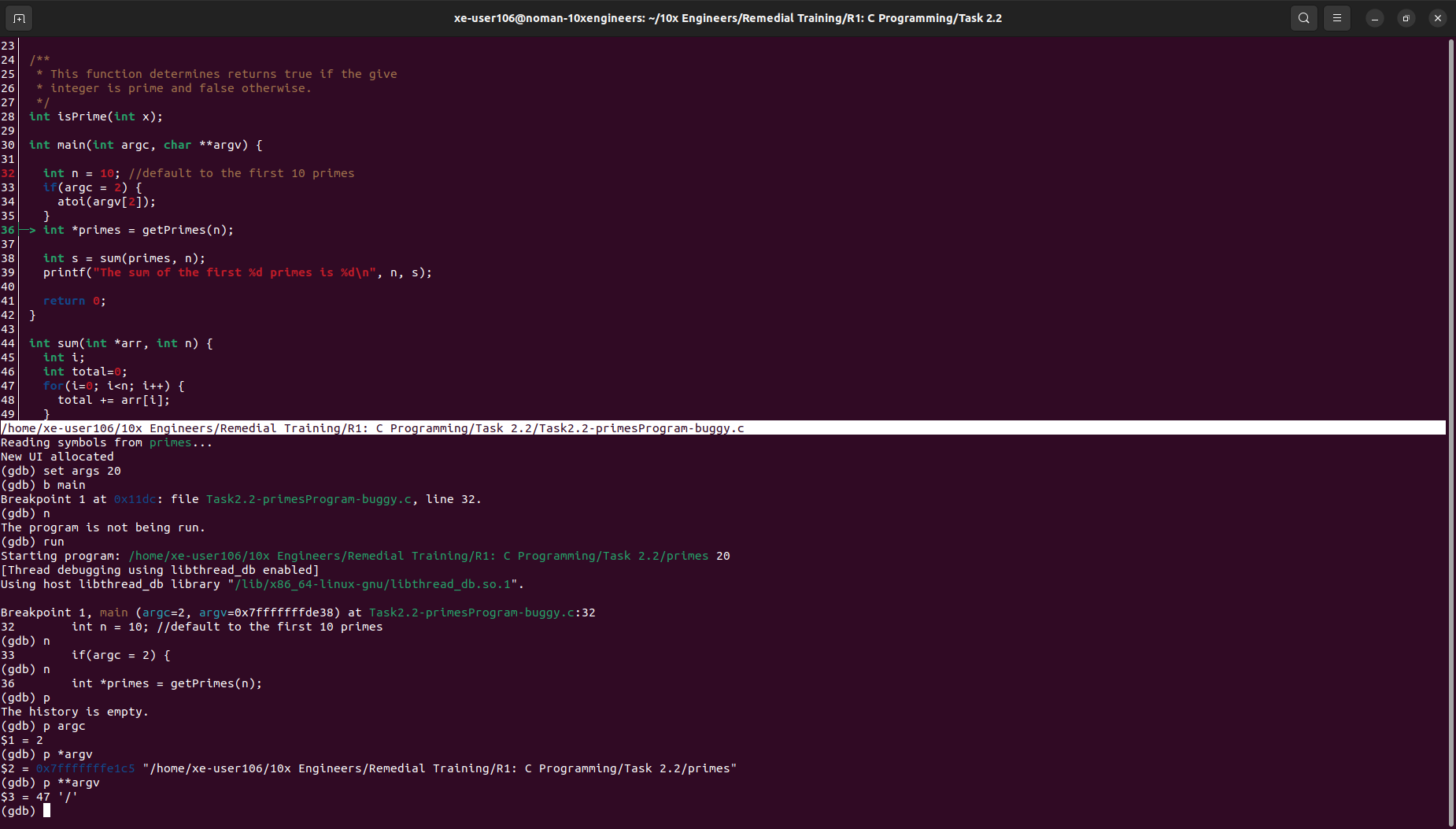
‭}‬

‭return total;‬

‭}‬



The program now only gives the sum for first 10 prime numbers and don’t consider any‬ ‭argument from the user. The function is calling‬‭ **atoi ( )** ‬‭but doesn’t do anything with the‬ ‭value it returns. Let’s fix that!‬

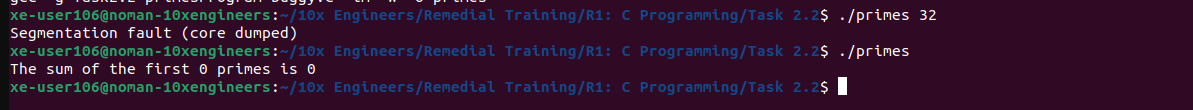


int n = 10; //default to the first 10 primes‬

‭if(argc = 2) {‬

‭n = atoi(argv[2]);‬

‭}‬



After recompiling, segmentation fault has occurred. The issue happened to be‬ due to‬‭ **argv[2]**‬ ‭which was actually accessing a null memory **(0x0)**, the solution is as‬ follows:

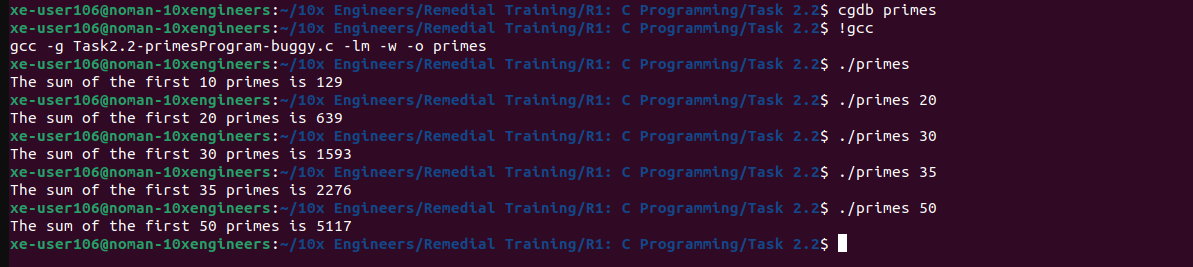
int main(int argc, char \*\*argv) {‬

i‭nt n = 10; //default to the first 10 primes‬

‭if(argc == 2) {‬

‭n = atoi(argv[1]);‬

‭}‬



The program is now completely bug free and working perfectly.‬