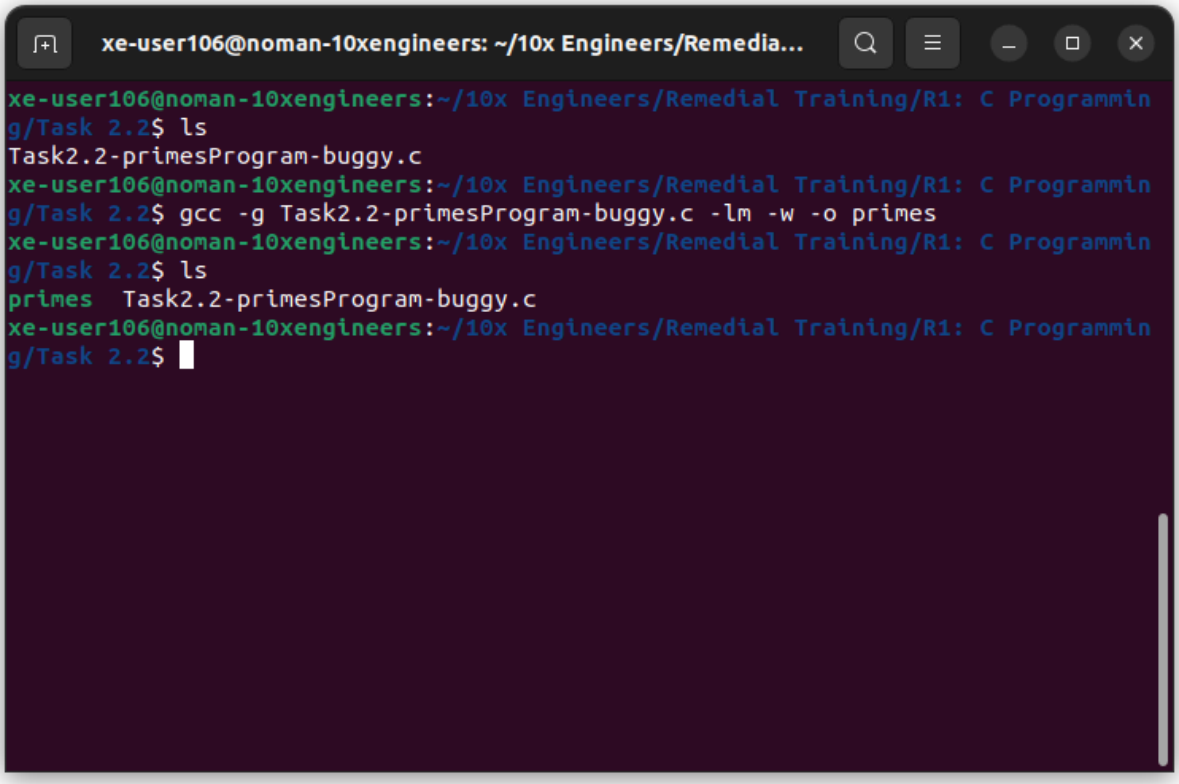
Module: R1: C Programming
Section: C Arrays, Pointers & Strings **Task:** 2.2

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:

```
xe-user106@noman-10xengineers: ~/10x Engineers/Remedial Training/R1: C Programming/Task 2.2
45    int i;
46    int total;
47    for(i=0; i<n; i++) {
48      total =+ arr[i];
49    }
50    return total;
51  }
52
53  int* getPrimes(int n) {
54    int result[n];
55    int i = 0;
56    int x = 2;
57    while(i < n) {
58 --> if(isPrime(x)) {
59        result[i] = x;
60        i++;
61        x += 2;
62      }
63    }
64    return result;
65  }
66
67  int isPrime(int x) {
68    if(x % 2 == 0) {
69      return 0;
70    }
71    for(int i=3; i<=sqrt(x); i+=2) {
/home/xe-user106/10x Engineers/Remedial Training/R1: C Programming/Task 2.2/Task2.2-primesProgram-buggy.c
(gdb) n
57        while(i < n) {
(gdb) n
58          if(isPrime(x)) {
(gdb) s
isPrime (x=2) at Task2.2-primesProgram-buggy.c:68
68        if(x % 2 == 0) {
(gdb) p x
$1 = 2
(gdb) n
69          return 0;
(gdb) n
77      }
(gdb) n
getPrimes (n=10) at Task2.2-primesProgram-buggy.c:58
58          if(isPrime(x)) {
(gdb) n
57        while(i < n) {
(gdb) n
58          if(isPrime(x)) {
(gdb) n
57        while(i < n) {
(gdb) p i
$2 = 0
(gdb) n
58          if(isPrime(x)) {
(gdb)
```

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

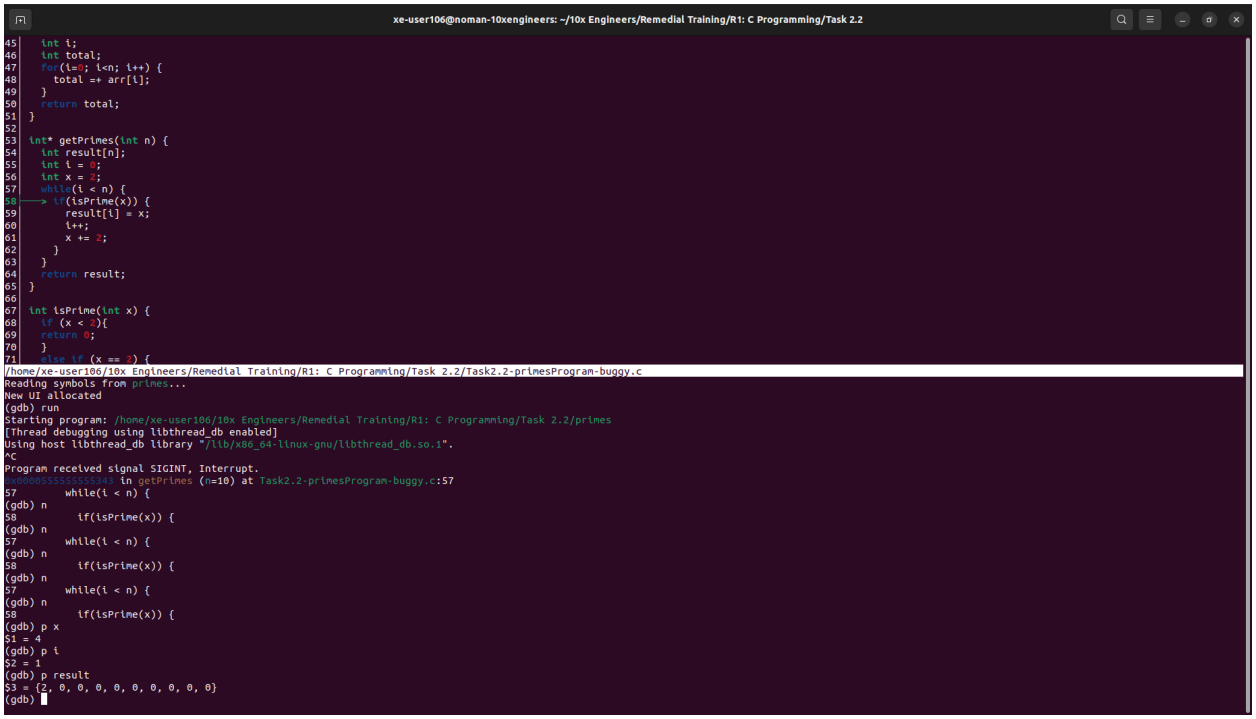
```
66
67  int isPrime(int x) {
68 --> if(x % 2 == 0) {
69      return 0;
70    }
71    for(int i=3; i<=sqrt(x); i+=2) {
72      if(x % i == 0) {
73        return 0;
74      }
75    }
76    return 1;
77  }
/home/xe-user106/10x Engineers/Remedial Training/R1: C Programming/Task 2.2/Task2.2-primesProgram-buggy.c
(gdb) n
69          return 0;
(gdb) n
77      }
(gdb) n
getPrimes (n=10) at Task2.2-primesProgram-buggy.c:58
58          if(isPrime(x)) {
(gdb) n
57        while(i < n) {
(gdb) n
58          if(isPrime(x)) {
(gdb) n
57        while(i < n) {
(gdb) p i
$2 = 0
(gdb) n
58          if(isPrime(x)) {
(gdb) s
isPrime (x=2) at Task2.2-primesProgram-buggy.c:68
68        if(x % 2 == 0) {
(gdb) p x
$3 = 2
(gdb) p isPrime(2)
$4 = 0
(gdb) p isPrime(4)
$5 = 0
(gdb)
```

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:

```
58 -> while(i < n) {
59      if(isPrime(x)) {
60        result[i] = x;
61        i++;
62        x += 2;
63      }
64    }
65    return result;
66  }
67
68  int isPrime(int x) {
69    if (x < 2){
70    return 0;
71    }
/home/xe-user106/10x Engineers/Remedial Training/R1: C Programming/Task 2.2/Task2.2-primesProgram-buggy.c
(gdb) n
59          if(isPrime(x)) {
(gdb) n
60            result[i] = x;
(gdb) n
61            i++;
(gdb) n
62            x += 2;
(gdb) n

Hardware watchpoint 2: x

Old value = 7
New value = 9
getPrimes (n=10) at Task2.2-primesProgram-buggy.c:58
58        while(i < n) {
(gdb) n
59          if(isPrime(x)) {
(gdb) n
58        while(i < n) {
(gdb) n
59          if(isPrime(x)) {
(gdb) n
58        while(i < n) {
(gdb) p x
$1 = 9
(gdb)
```

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.

```
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Task 2.2$ ./primes
Segmentation fault (core dumped)
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Task 2.2$
```

```c
int* getPrimes(int n) {
  int *result = (int*)malloc(n * sizeof(int)); // Dynamically allocate memory
  if (result == NULL) {
    printf("Memory allocation failed\n");
    exit(1);
  }

  result[0] = 2;
  int i = 1;
  int x = 3;
  while(i < n) {
    if(isPrime(x)) {
      result[i] = x;
      i++;
    }
    x += 2;
  }
  return result;
}

int isPrime(int x) {
  if (x < 2){
  return 0;
```

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

```
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Task 2.2$ vim Task2.2-primesProgram-buggy.c
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Task 2.2$ !gcc
gcc -g Task2.2-primesProgram-buggy.c -lm -w -o primes
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Task 2.2$ ./primes
The sum of the first 10 primes is 29
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Task 2.2$
```

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

```
xe-user106@noman-10xengineers: ~/10x Engineers/Remedial Training/R1: C Programming/Task 2.2
26   * integer is prime and false otherwise.
27   */
28  int isPrime(int x);
29
30  int main(int argc, char **argv) {
31
32    int n = 10; //default to the first 10 primes
33    if(argc = 2) {
34      atoi(argv[2]);
35    }
36    int *primes = getPrimes(n);
37
38    int s = sum(primes, n);
39--> printf("The sum of the first %d primes is %d\n", n, s);
40
41    return 0;
42  }
43
44  int sum(int *arr, int n) {
45    int i;
46    int total;
47    for(i=0; i<n; i++) {
48      total =+ arr[i];
49    }
50    return total;
51  }
52
/home/xe-user106/10x Engineers/Remedial Training/R1: C Programming/Task 2.2/Task2.2-primesProgram-buggy.c
New UI allocated
(gdb) run
Starting program: /home/xe-user106/10x Engineers/Remedial Training/R1: C Programming/Task 2.2/primes
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
The sum of the first 10 primes is 29
[Inferior 1 (process 30320) exited normally]
(gdb) b main
Breakpoint 1 at 0x555555555 1dc: file Task2.2-primesProgram-buggy.c, line 32.
(gdb) run
Starting program: /home/xe-user106/10x Engineers/Remedial Training/R1: C Programming/Task 2.2/primes
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main (argc=1, argv=0x7fffffffde48) at Task2.2-primesProgram-buggy.c:32
32        int n = 10; //default to the first 10 primes
(gdb) n
33        if(argc = 2) {
(gdb) n
36        int *primes = getPrimes(n);
(gdb) n
38        int s = sum(primes, n);
(gdb) n
39        printf("The sum of the first %d primes is %d\n", n, s);
(gdb) p *primes@10
$1 = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29}
(gdb)
```

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.

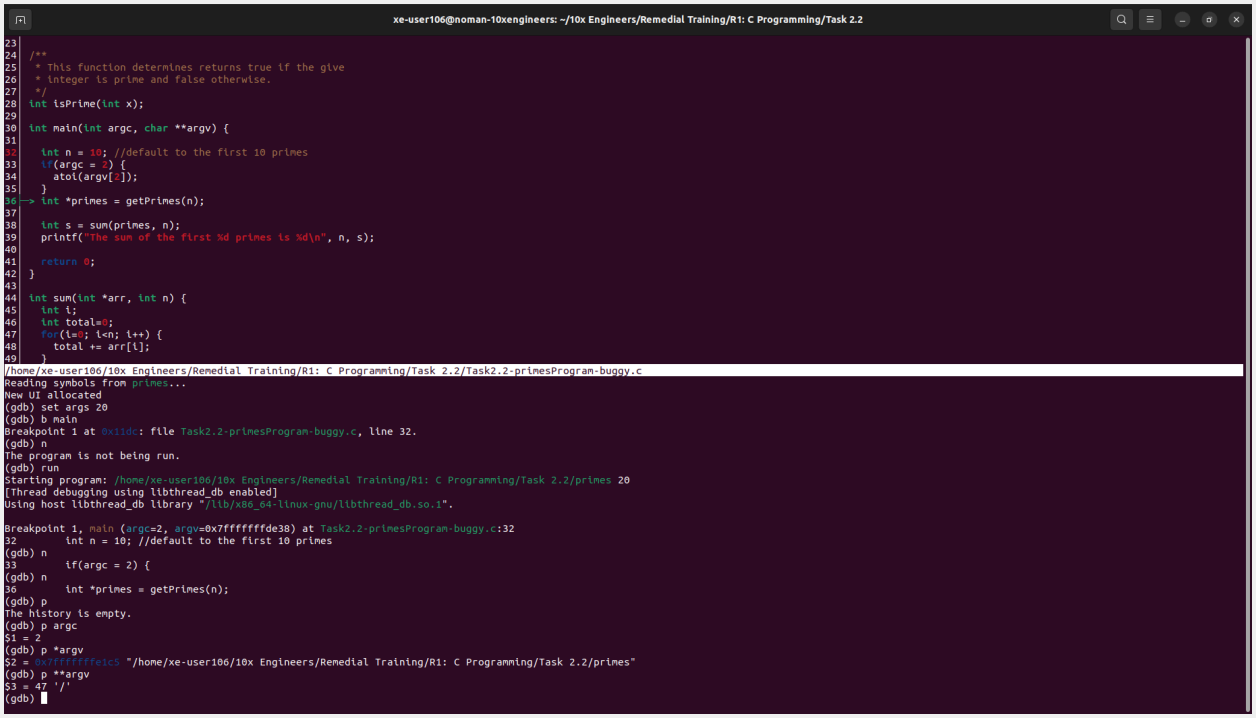
```
xe-user106@noman-10xengineers: ~/10x Engineers/Remedial Training/R1: C Programming/Task 2.2
35    }
36    int *primes = getPrimes(n);
37
38    int s = sum(primes, n);
39    printf("The sum of the first %d primes is %d\n", n, s);
40
41    return 0;
42  }
43
44  int sum(int *arr, int n) {
45    int i;
46    int total;
47    for(i=0; i<n; i++) {
48  ---> total =+ arr[i];
49    }
50    return total;
51  }
52
53  int* getPrimes(int n) {
54    int *result = (int*)malloc(n * sizeof(int)); // Dynamically allocate memory
55    if (result == NULL) {
56      printf("Memory allocation failed\n");
57      exit(1);
58    }
59
60    result[0] = 2;
61    int i = 1;
/home/xe-user106/10x Engineers/Remedial Training/R1: C Programming/Task 2.2/Task2.2-primesProgram-buggy.c
Old value = 21845
New value = 2
sum (arr=0x555555555592a0, n=10) at Task2.2-primesProgram-buggy.c:47
47        for(i=0; i<n; i++) {
(gdb) n
48          total =+ arr[i];
(gdb) n

Hardware watchpoint 3: total

Old value = 2
New value = 3
sum (arr=0x555555555592a0, n=10) at Task2.2-primesProgram-buggy.c:47
47        for(i=0; i<n; i++) {
(gdb) n
48          total =+ arr[i];
(gdb) n

Hardware watchpoint 3: total

Old value = 3
New value = 5
sum (arr=0x555555555592a0, n=10) at Task2.2-primesProgram-buggy.c:47
47        for(i=0; i<n; i++) {
(gdb) n
48          total =+ arr[i];
(gdb)
```

It will take the following correction to avoid any garbage values:

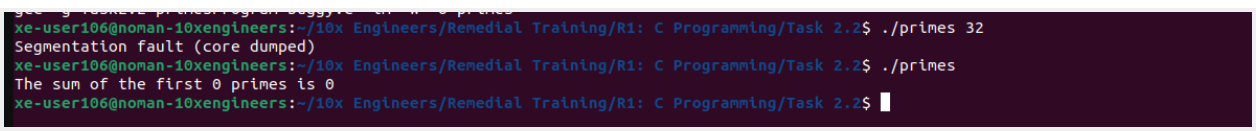
```
int sum(int *arr, int n) {
      int i;
      int total = 0;
      for(i=0; i<n; i++) {
      total += arr[i];
}
      return total;
}
```

```
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Task 2.2$ !gcc
gcc -g Task2.2-primesProgram-buggy.c -lm -w -o primes
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Task 2.2$ ./primes
The sum of the first 10 primes is 129
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Task 2.2$ ./primes 30
The sum of the first 10 primes is 129
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Task 2.2$ ./primes 32
The sum of the first 10 primes is 129
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Task 2.2$
```

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]);
}
```

```
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Task 2.2$ cgdb primes
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Task 2.2$ !gcc
gcc -g Task2.2-primesProgram-buggy.c -lm -w -o primes
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Task 2.2$ ./primes
The sum of the first 10 primes is 129
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Task 2.2$ ./primes 20
The sum of the first 20 primes is 639
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Task 2.2$ ./primes 30
The sum of the first 30 primes is 1593
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Task 2.2$ ./primes 35
The sum of the first 35 primes is 2276
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Task 2.2$ ./primes 50
The sum of the first 50 primes is 5117
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Task 2.2$
```

The program is now completely bug free and working perfectly.