**C PROGRAMMING TASK#1**

**LAB 01**

**EXCERCISE 1**

**Part 1**

* Implement the function num\_occurences in exercise1/ex1.c. The directions for completing the function can be found in the comments above the function. test\_ex1.c will be used to test your code.

**ANSWER:** I have implemented the **num\_occurences** in order to run the program you can see in the screen shot below.

**Part 2**

* Implement the function compute\_nucleotide\_occurrences in ex1.c. Look at ex1.h to see relevant struct definitions. Hint: You may be able to reuse num\_occurences

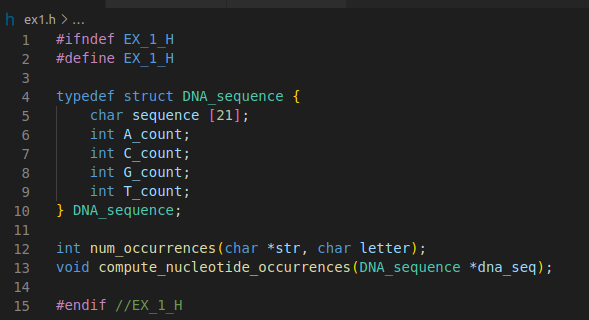
**ANSWER:** I have implemented the **compute\_nucleotide\_occurrences** in order to run the program you can see in the screen shot below.

* Think of a scenario that is not tested by the current test cases. Create one additional test case to test this scenario.

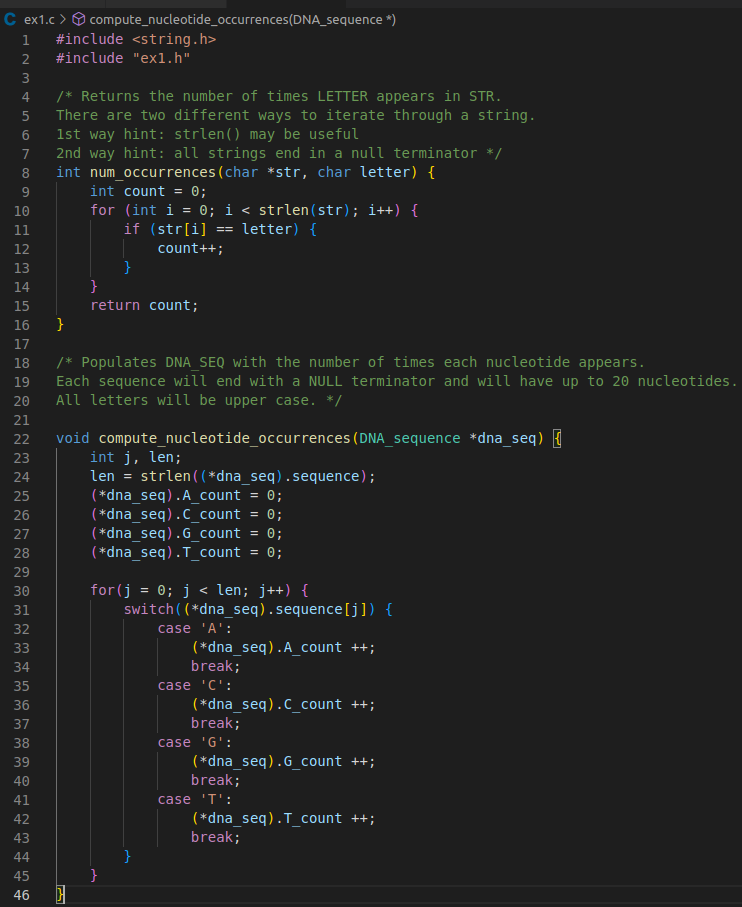
**ANSWER:** I have added the new scenarios to the both the test cases

**CODE**

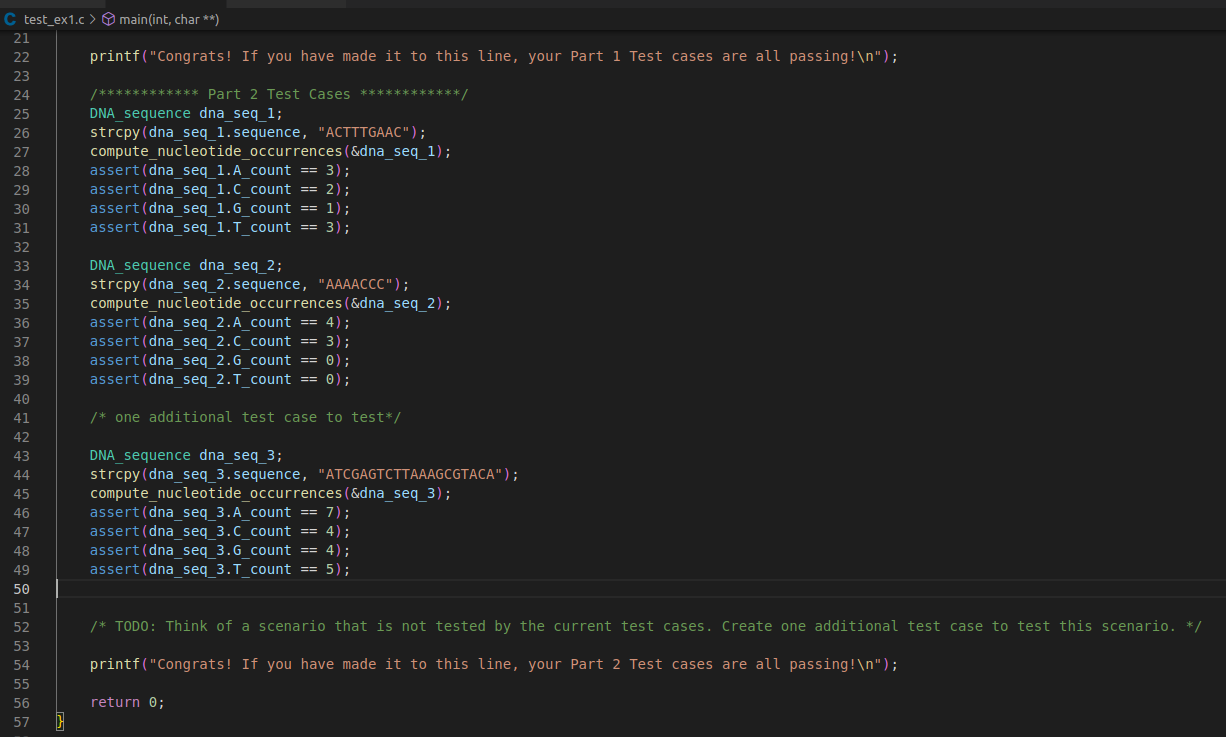
**Ex1.h**



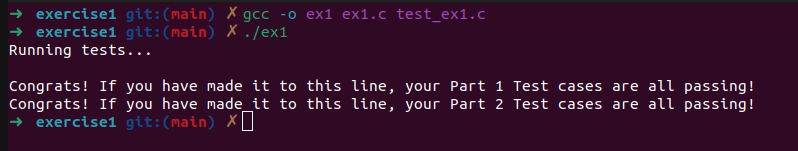
**ex1.c**



**test\_ex1.c**



**TERIMINAL OUTPUT**



**EXCERCISE 2**

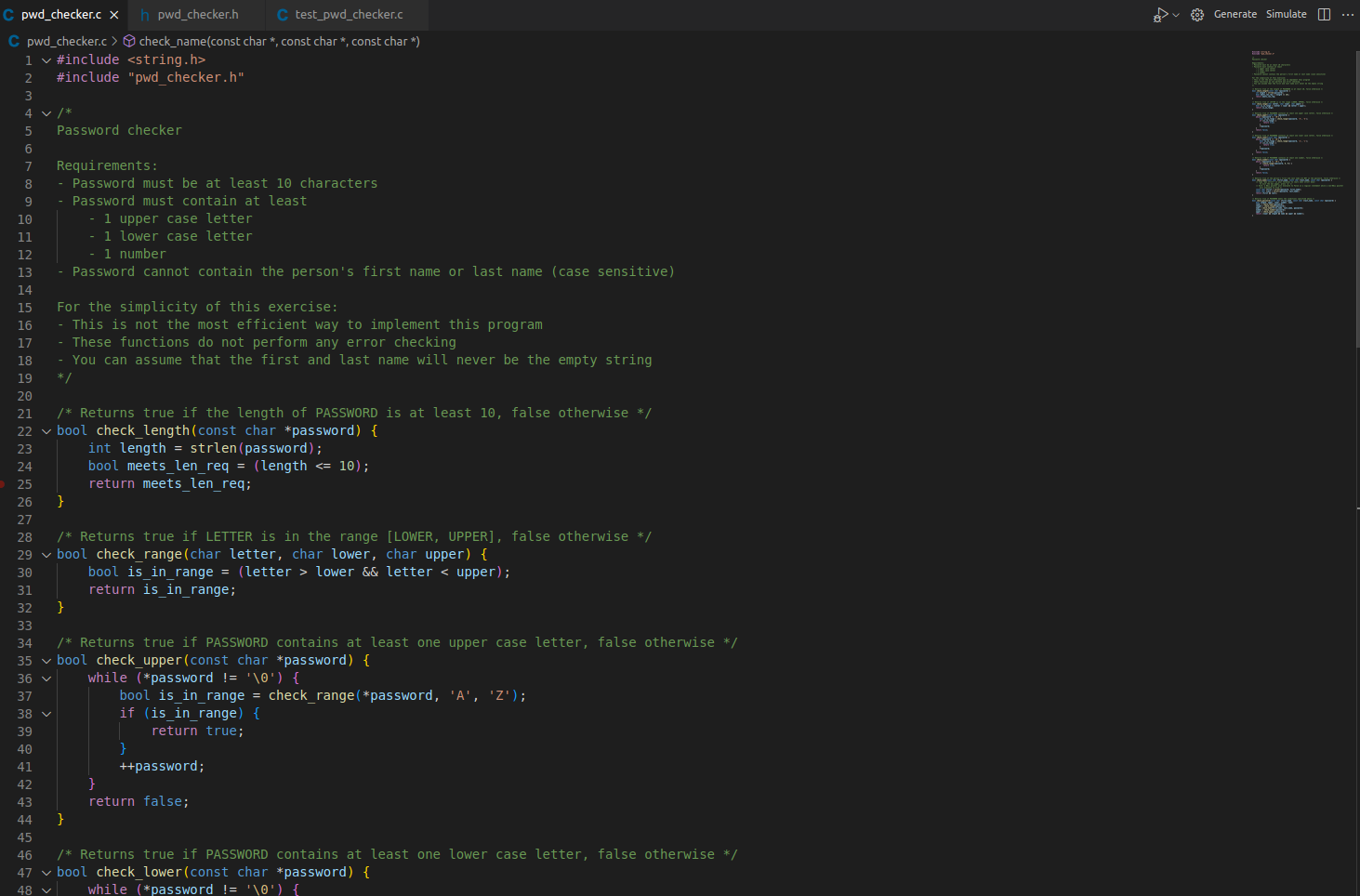
**Part 1**: Compiler Warnings

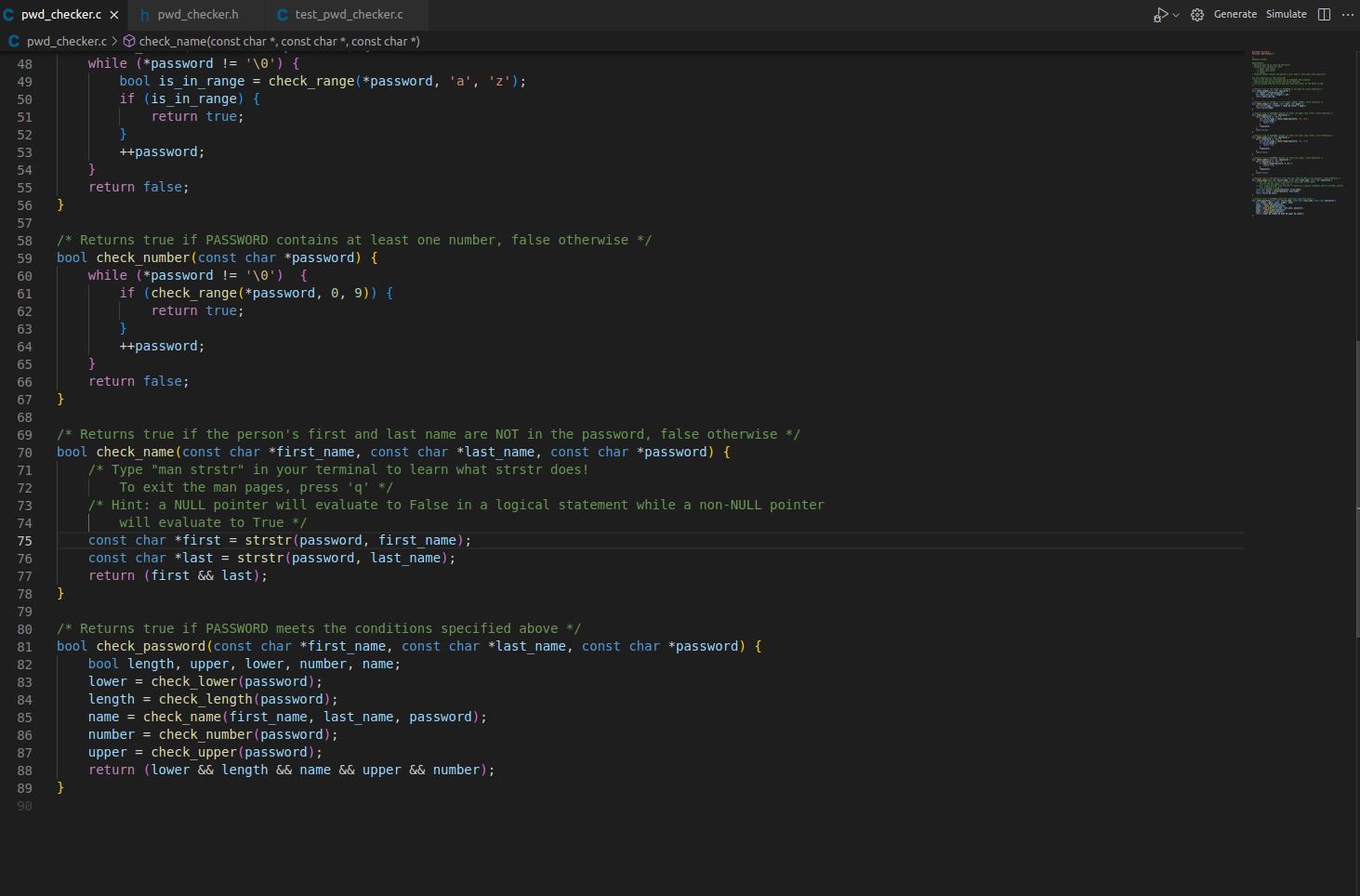
* Fix the remaining compiler warnings in pwd\_checker.c

**ANSWER:** I have resolved all the warnings in the checker.c as you can see my teriminal output and my code below

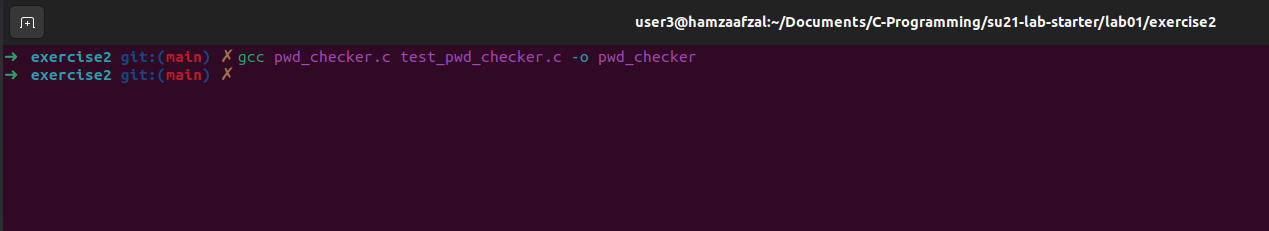
**CODE**

**pwd\_checker.c**



\

**TERIMINAL OUTPUT**



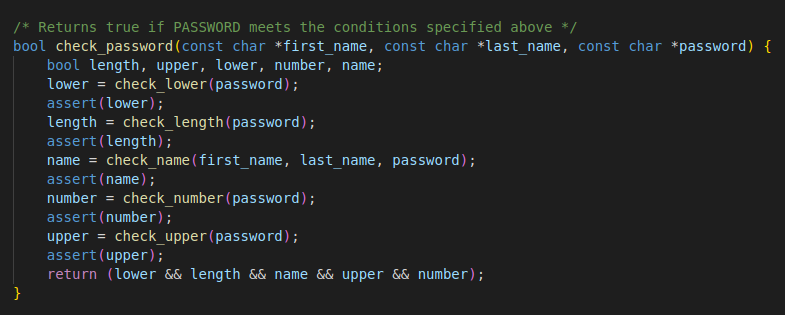
**Part 2 :** Assert Statements

* Learn about how you can use assert statements to debug your code. Edit the code in pwd\_checker according to the directions in this section.

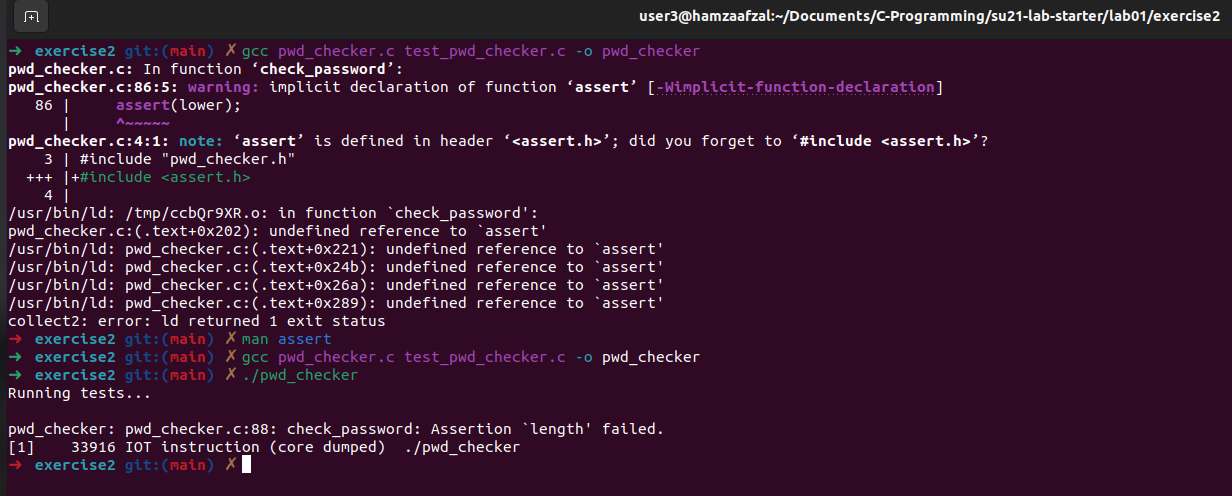
**ANSWER:** I have learned to use the assert statement to find the bug in our password\_checker and found at that there Is an bug in the length function.

**CODE**

**pwd\_checker.c**



**TERIMINAL OUTPUT**



**Part 3 :** Intro to GDB: start, step, next, finish, print, quit

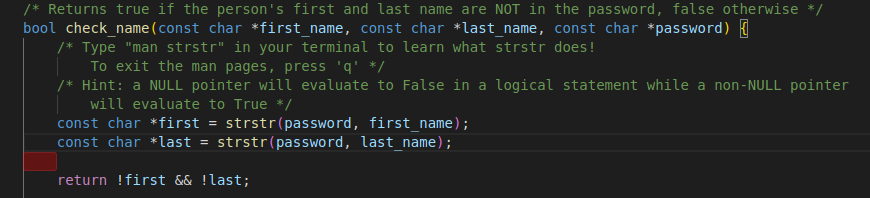
* Debug check\_name on your own using the commands you just learned. Make sure you use the -g flag when compiling. Make sure that you are using cgdb to debug so that you practice using it because it will help you with your projects.

**ANSWER:** Used the the cgbd to debug the code and found out that the strstr will return a NULL pointer and bollian gives false on the that.So I can used negations on the first and last.

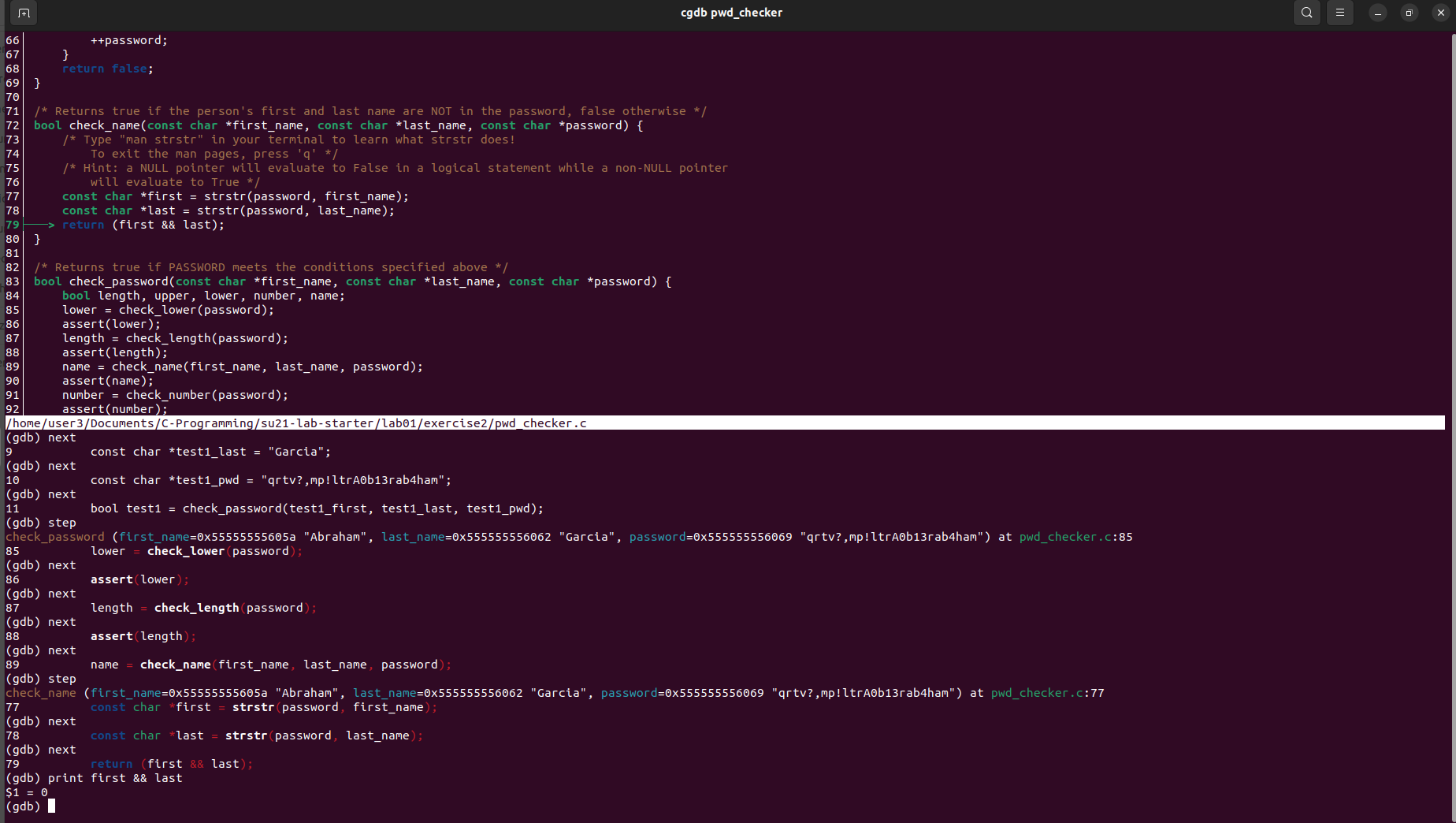
Check out the Code and terminal output

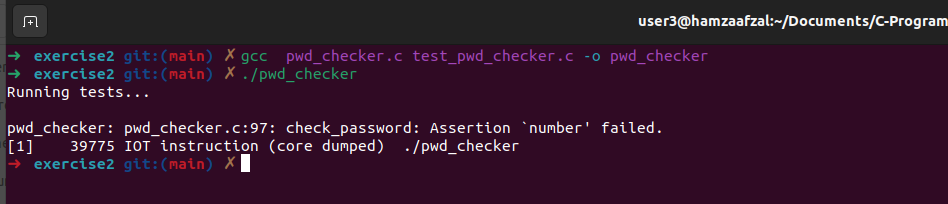
**CODE**

**pwd\_checker.c**



**TERIMINAL OUTPUT**





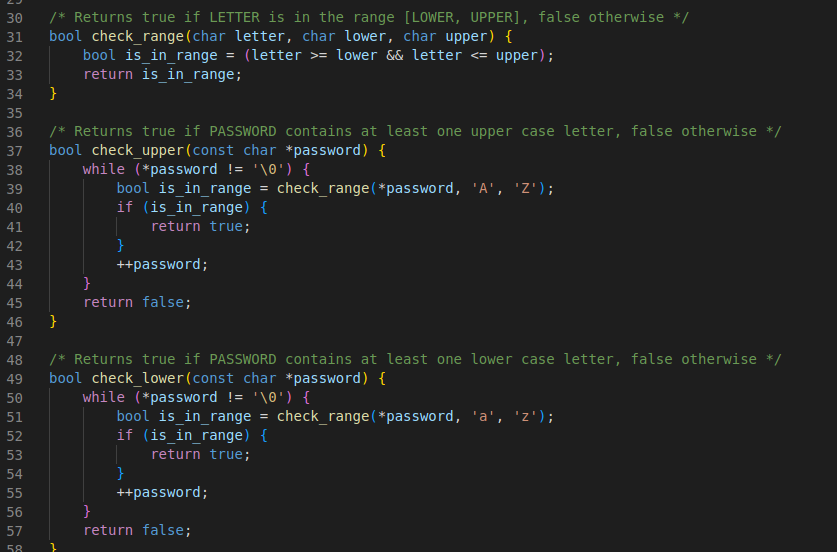
**Part 4 :** Intro to GDB: break, conditional break, run, continue

* Debug check\_upper on your own using the commands you just learned.

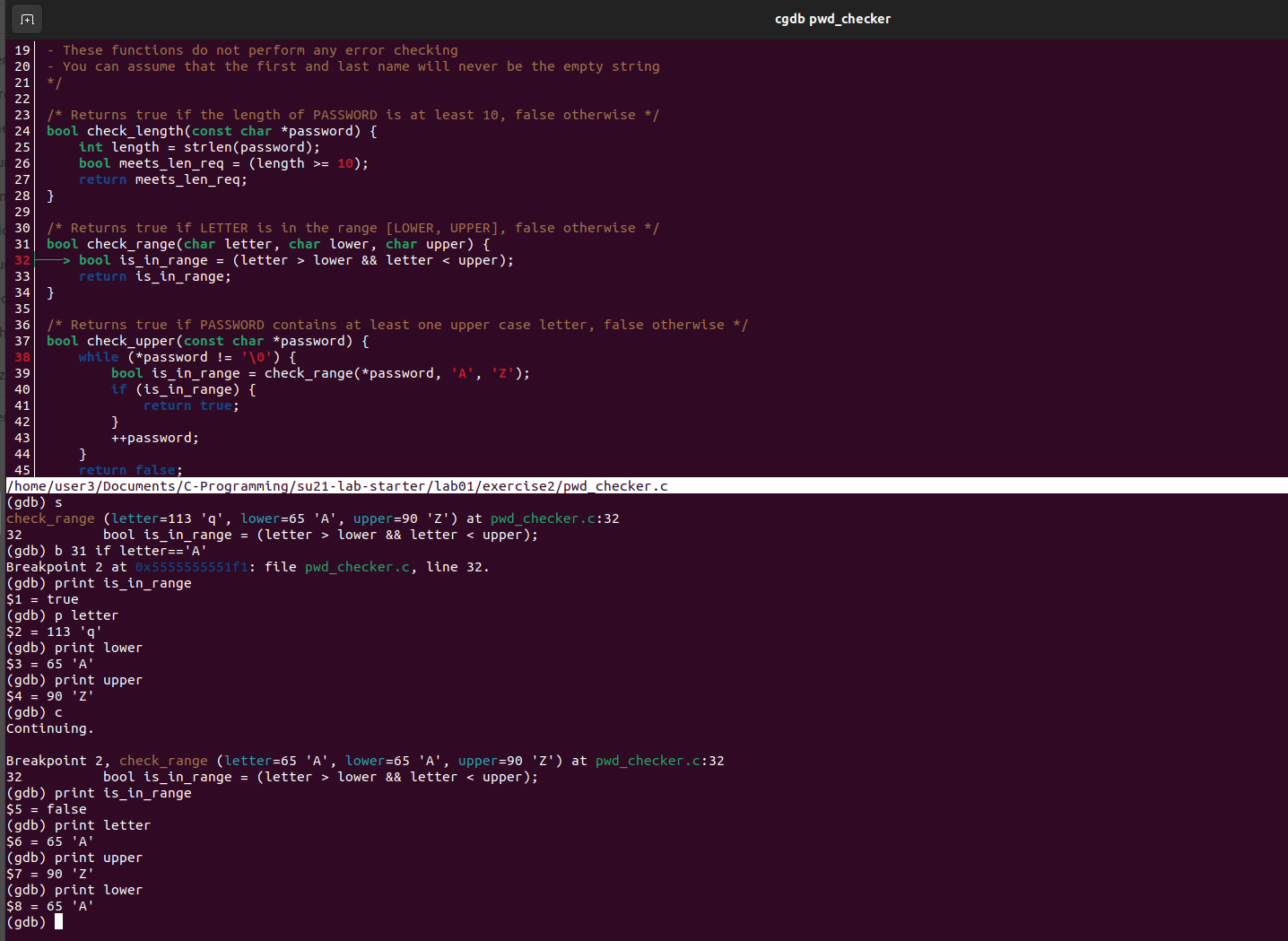
**ANSWER:** Used the the cgbd to debug the code and made chnges in the code to pass all the test.You can see the code down below.

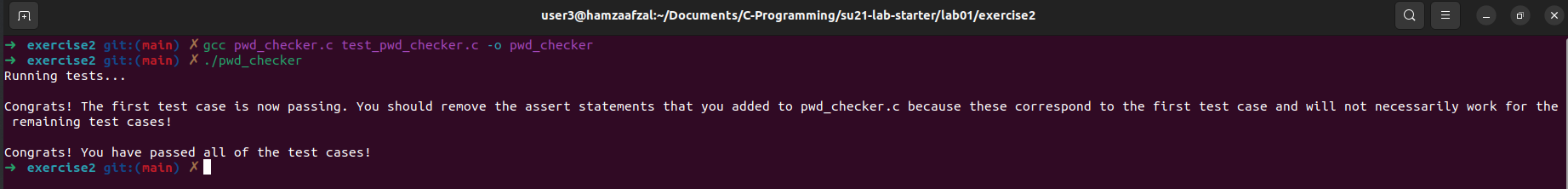
**CODE**

**pwd\_checker.c**



**TERIMINAL OUTPUT**



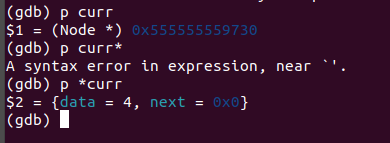


**EXCERCISE 3**

**Part :** Printing

* Try printing out some pointers in GDB

**TERIMINAL OUTPUT**

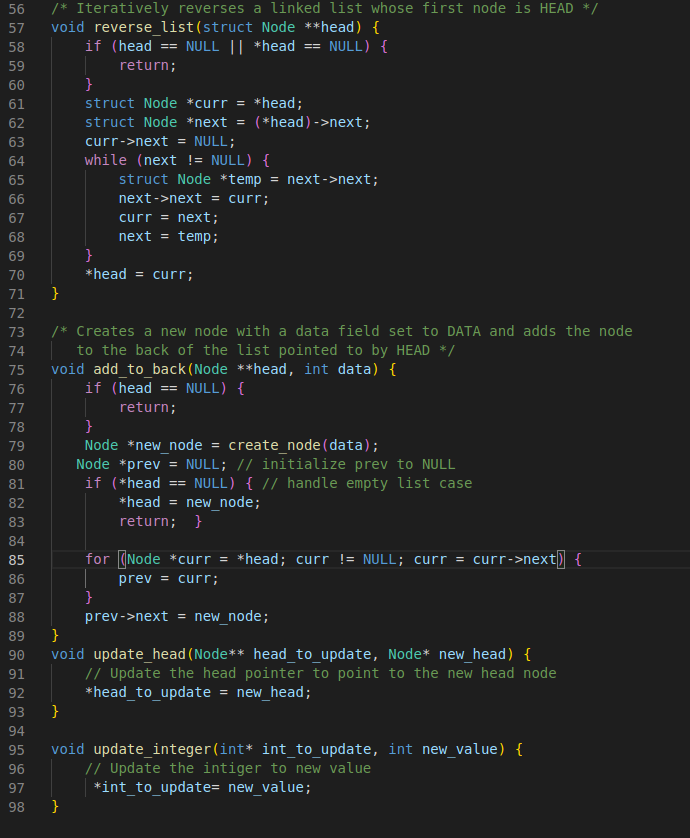


**Part :** How to find segfaults

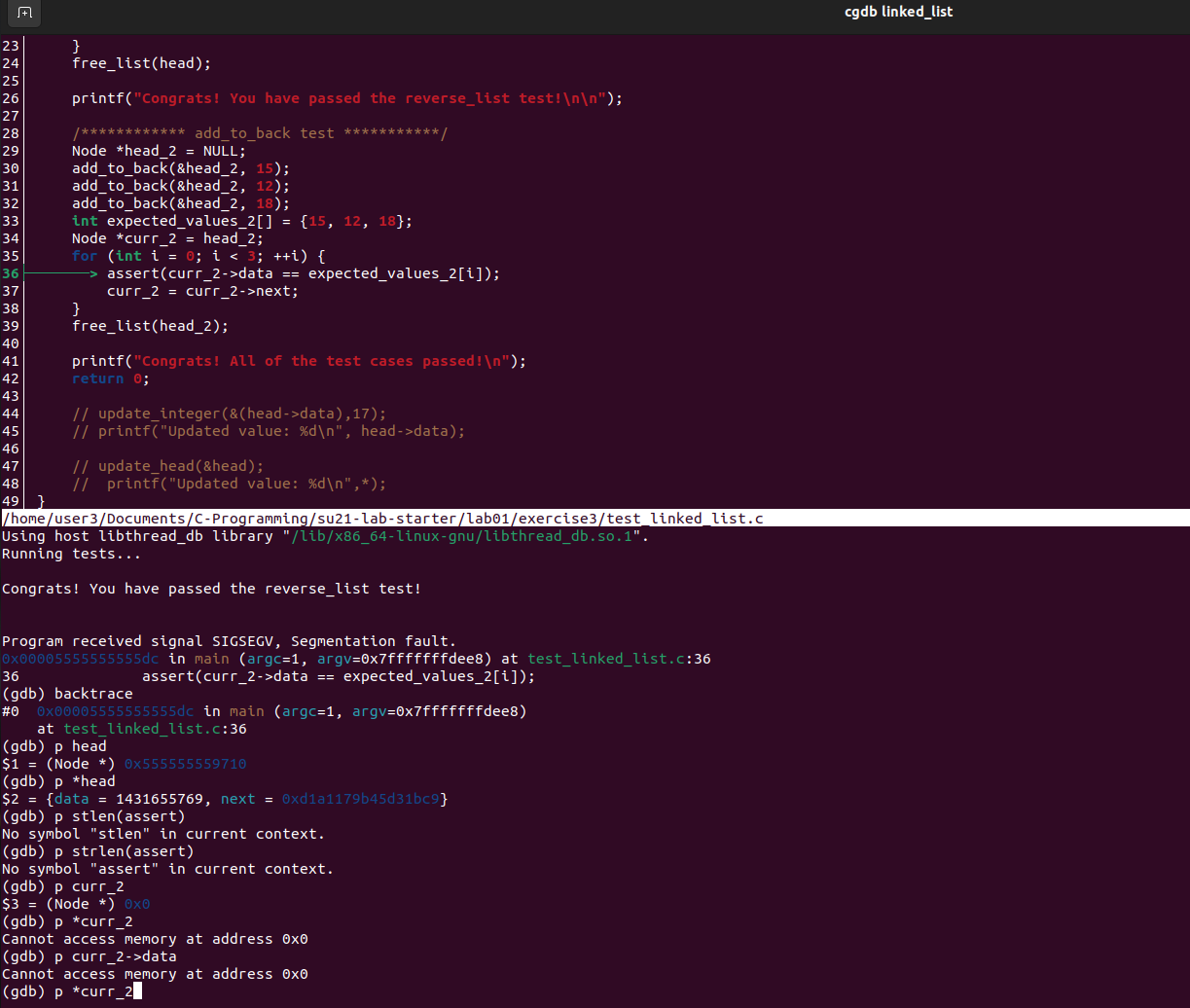
* Debug add\_to\_back on your own using the commands you just learned. Make sure that you are using cgdb so that you can get practice using it. This will help you with your projects.
* **ANSWER:** The segmentation fault error in this function is occurring due to the following reasons:
* **Dereferencing a null pointer**: In the beginning of the function, we check if the **head** pointer is **NULL** or not. However, we do not handle the case when **\*head** is **NULL**. In this case, when we try to access **\*head** in the for loop, it will result in a segmentation fault error. To fix this, I have added a check for **\*head == NULL** and set **\*head = new\_node** if it is **NULL**.
* **Using an uninitialized pointer**: If the linked list is empty and **prev** is not initialized to **NULL**, it will point to an unknown memory location resulting in a segmentation fault error when we try to access **prev->next**. To fix this, I have initialize **prev** to **NULL** before the for loop.

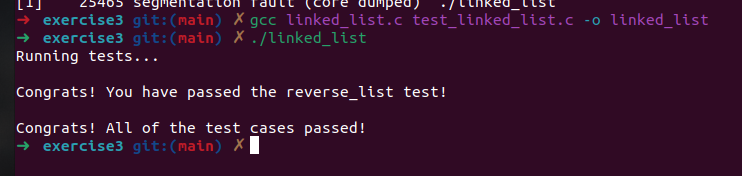
**CODE**

**Linked\_list.c**



**TERIMINAL OUTPUT**





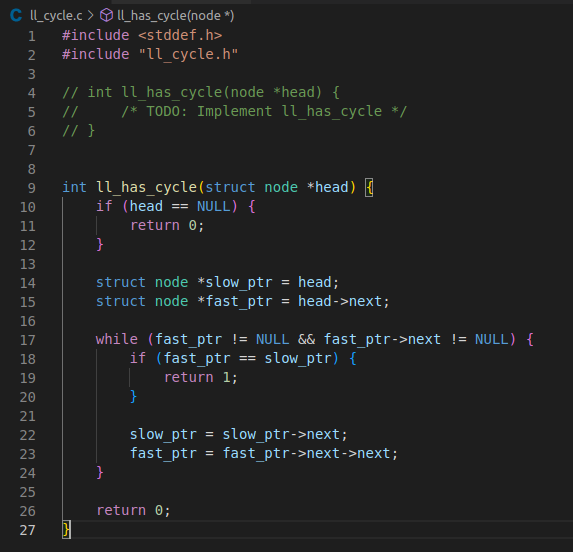
**EXCERCISE 4**

Action Item

* In ll\_cycle.c, complete the function ll\_has\_cycle() to implement the following algorithm for checking if a singly- linked list has a cycle.
* **ANSWER:** I have implemented the function below

**CODE**

**Testllcycle.c**



**TERIMINAL OUTPUT**

