**Lab01: (** [**https://github.com/Its-Masoom/su21-lab-starter**](https://github.com/Its-Masoom/su21-lab-starter) **)**

**Exercise 1):**

**Part 1):**

Here is my code for num\_occurences function

int num\_occurrences(char \*str, char letter) {

int count = 0;

while (\*str != '\0') {

if (\*str == letter) {

count++;

}

str++;

}

return count;

}

This is the additional test case I have added

int num\_o = num\_occurrences(str, 'o');

assert(num\_o == 2);

**Part 2):**

Here is my code for compute\_nucleotide\_occurences

void compute\_nucleotide\_occurrences(DNA\_sequence \*dna\_seq) {

int i = 0;

int count\_A = 0, count\_C = 0, count\_G = 0, count\_T = 0;

while(dna\_seq->sequence[i] != '\0') {

switch(dna\_seq->sequence[i]) {

case 'A':

count\_A++;

break;

case 'C':

count\_C++;

break;

case 'G':

count\_G++;

break;

case 'T':

count\_T++;

break;

default:

break;

}

i++;

}

dna\_seq->A\_count = count\_A;

dna\_seq->C\_count = count\_C;

dna\_seq->G\_count = count\_G;

dna\_seq->T\_count = count\_T;

return;

}

This is the additional test case I have added

DNA\_sequence dna\_seq\_3;

strcpy(dna\_seq\_3.sequence, "TTCTGACTGCAACGGGCAAT");

compute\_nucleotide\_occurrences(&dna\_seq\_3);

assert(dna\_seq\_3.A\_count == 5);

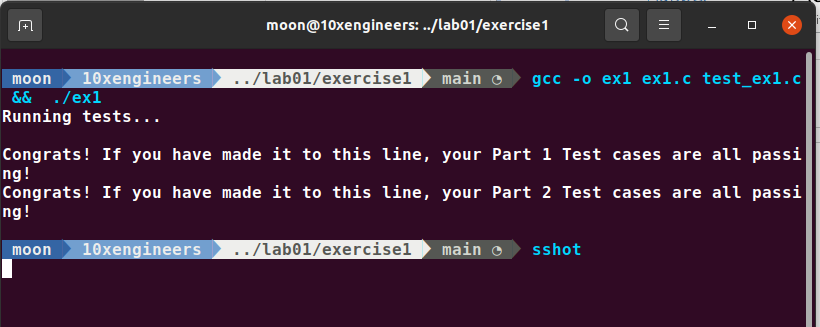
assert(dna\_seq\_3.C\_count == 5);

assert(dna\_seq\_3.G\_count == 5);

assert(dna\_seq\_3.T\_count == 5);

**Output of Part 1 and Part 2:**

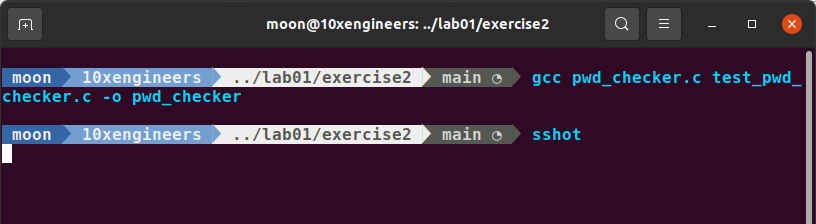
You can see there is no error generated by any of assert statement.

****

**Exercise 2):**

**Part 1): Compiler Warnings**

I have resolve all warnings. You can see it is successfully compiling and there is no warning.



**Part 2): Assert Statements**

I have added the following assert statements in check\_password function

bool check\_password(const char \*first\_name, const char \*last\_name, const char \*password) {

bool length, upper, lower, number, name;

lower = check\_lower(password);

assert(lower);

length = check\_length(password);

assert(length);

name = check\_name(first\_name, last\_name, password);

assert(name);

number = check\_number(password);

assert(number);

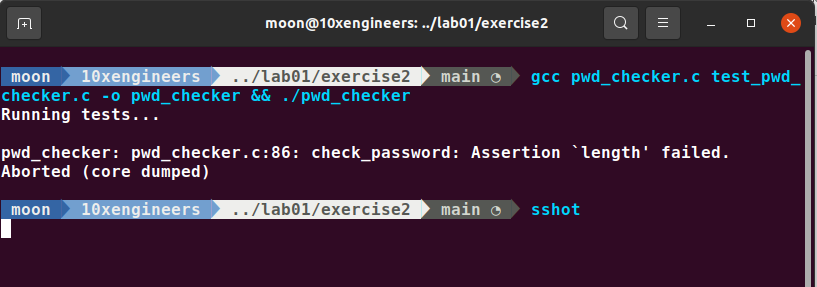
upper = check\_upper(password);

assert(upper);

return (lower && length && name && upper && number);

}

When I run this program, I came to know that my length function is not working properly because its assertion is failing.



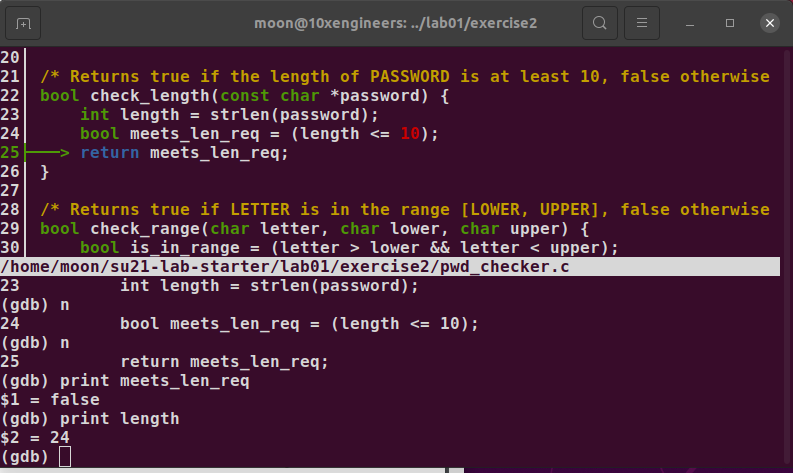
**Part 3): Intro to GDB: start, step, next, finish, print, quit**

**Step 1):**

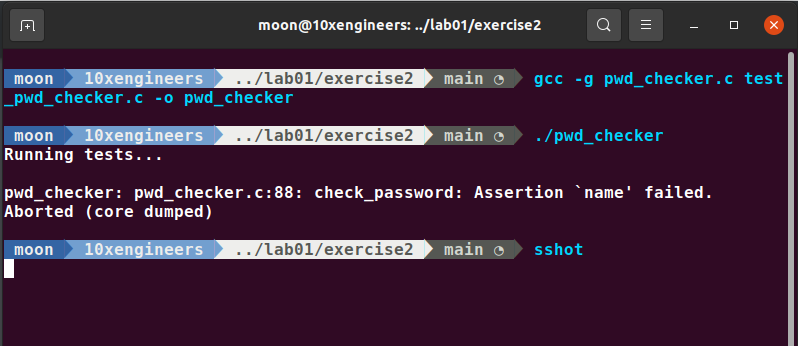
Now I use cgdb to find why error function is failing so executing the below command leads us to the cgdb environment.

$ cgdb pwd\_checker

Now I used next and step command and finally step into length function and found that meets\_len\_req is returning false and length is correct so there must be some logical error. And finally I found that in line 24 the check should be length>=10. Then I go to my source code and correct that check.

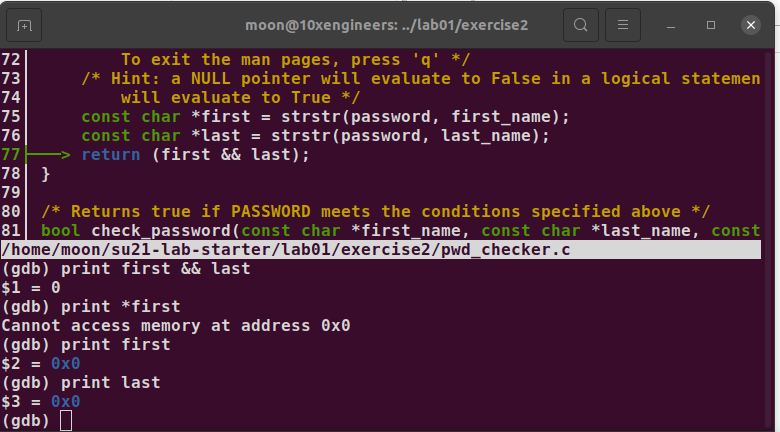


After this length assertion passed. But this time name assertion is failing.

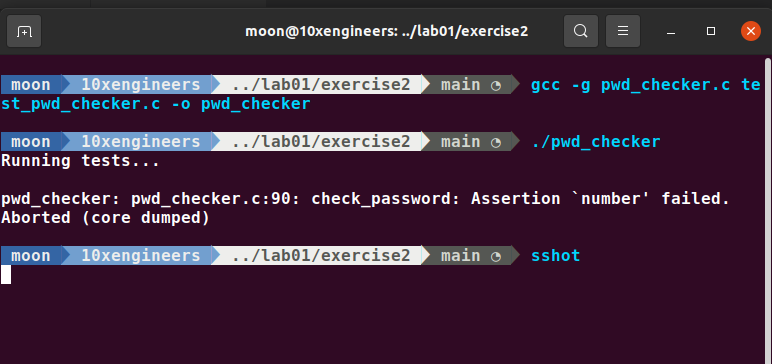


**Step 2):**

Again using cgdb and its command (start, next and step), I step into name function and print first && last I get 0. Printing first and last gives 0x0 means they are pointing to null. Then I google about the usage of strstr function and come to know that this function returns false and points to null if pattern is not found. So in this case first\_name and last\_name pattern are not in the password so it returns null (Working perfectly fine). Null usage in boolean expression will evaluate to false. But I need true to be return by this function. So finally I found the error to be in line 77 of the ss below. There should be return (!first && !last)



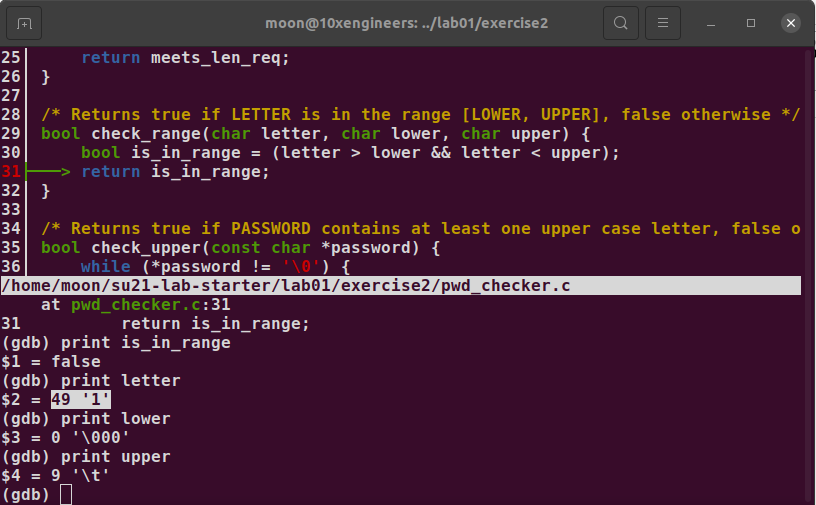
Correcting this error and running again, I found that my name assertion is passed. But this time number assertion is failing. I will resolve this assertion in next part.



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

**Step 1):**

Using next, step, break, conditional break, run and continue I step into check\_range function. Then I print is\_in\_range, letter, lower and upper. On printing lower it shows me 0 '\000' and printing upper shows 9 '\t' while printing letter gives 49 '1'. That means 1 is a character that is I was comparing with the integers.



So I found error to be in check\_number function and then fix it as:

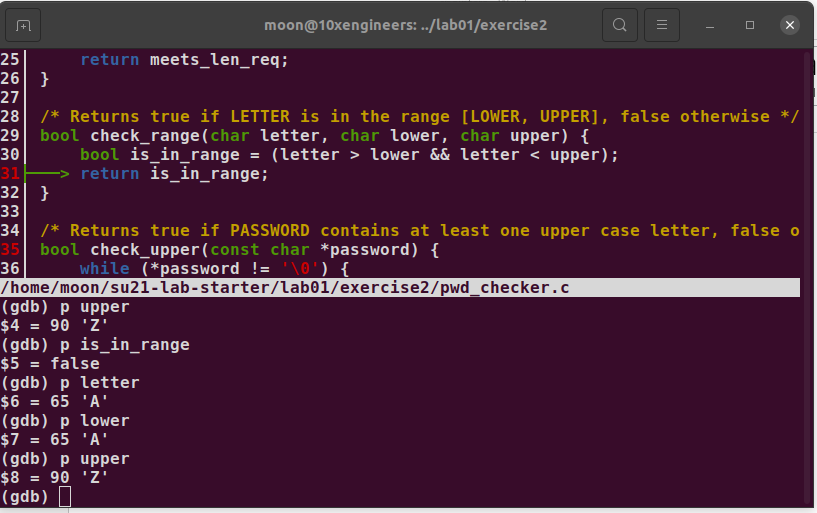
if (check\_range(\*password, ‘0’, ‘9’))

Then compiling and running, we have solved the assertion error of check\_number function as it is shown below. But assertion is failed in upper function. Now we will solved this in next part.



**Step 2):**

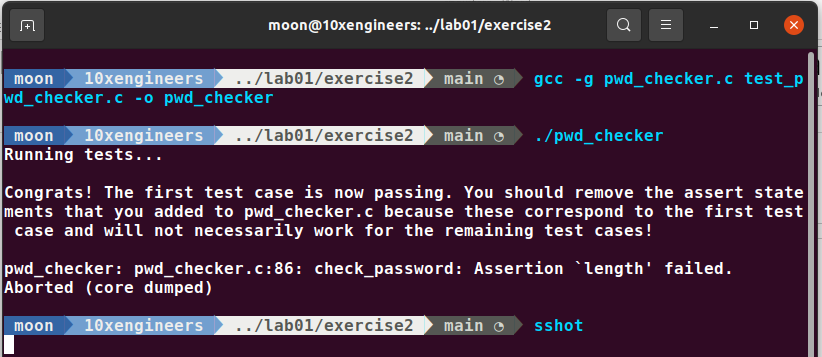
Using next, step, break, conditional break, run and continue I step into check\_upper function. Then step into check\_range function and set up a break point when letter==’A’. Then I print is\_in\_range and it gives me false. Infact ‘A’ is in range. Then I print letter, lower and upper.



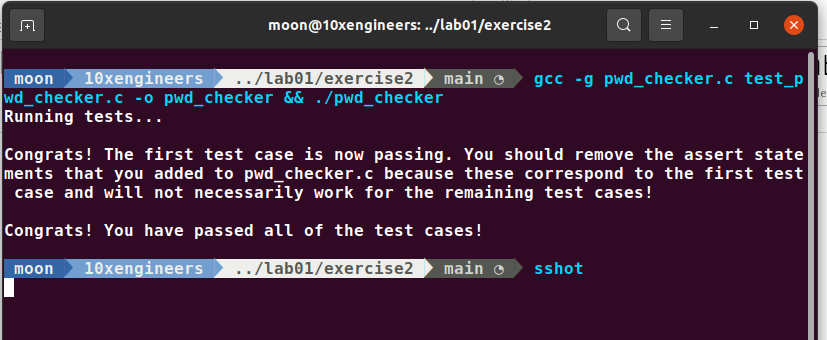
Looking at the ASCII values, I finally come to know that my condition is that letter > lower so it is comparing 65>65 and resulting false. So I need to fix > with >= as:

bool is\_in\_range = (letter >= lower && letter < upper);

By compiling and running, my assertion check\_upper also passed this time.



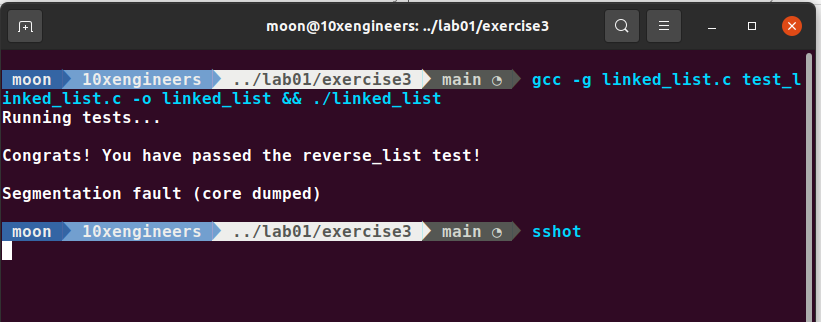
Then removing the assert statements, finally all the tests will be passed as shown in the picture below.



**Exercise 3):**

**How to find Segmentation Faults:**

By following all the instructions given in the manual, I have passed my reverse\_list test.



Now the task given in the manual is to debug add\_to\_back using the same approach.

Using cgdb and the debugging commands, I found that there is segmentation fault when

curr\_2->data is going to compare with expexted\_values\_2[i]. Debugging add\_to\_back shows that when the list was empty there was no \*head created and prev->next was going to be the new\_node instead of \*head. Adding the following lines solved the problem

Node \*prev=NULL;

if (prev == NULL) { // If the list was empty

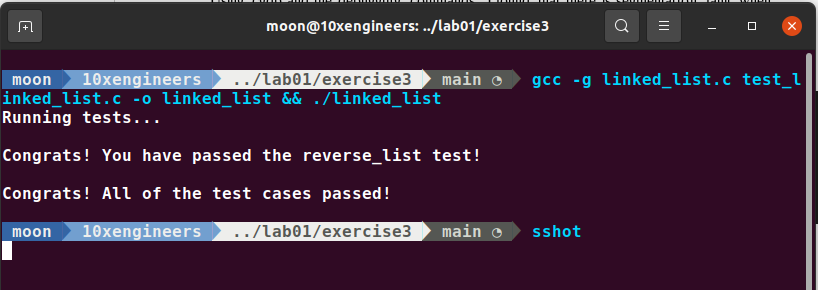
\*head = new\_node;

} else {

prev->next = new\_node;

}

Now compiling and running results in no segmentation fault.



**Exercise 4):**

According to the provided information, here is the function ll\_has\_cycle which I had implemented.

int ll\_has\_cycle(node \*head) {

/\* TODO: Implement ll\_has\_cycle \*/

// Initialize the fast and slow pointers

node \*fast\_ptr = head;

node \*slow\_ptr = head;

// Traverse the list with fast and slow pointers

while (fast\_ptr && fast\_ptr->next) {

// Advance the fast pointer by two nodes

fast\_ptr = fast\_ptr->next->next;

// Advance the slow pointer by one node

slow\_ptr = slow\_ptr->next;

// If the fast and slow pointers ever point to the same node, the list is cyclic

if (fast\_ptr == slow\_ptr) {

return 1;

}

}

// If we reach the end of the list, it is acyclic

return 0;

}

Here, I started with two pointers at the head of the list. One is fast\_ptr and the other is slow\_ptr. Then I advance fast\_ptr by two nodes according to the given instructions. The while loop is the important condition which is checking if fast-ptr and the next of fast-ptr i.e fast\_ptr->next should not be null. If they are null means that linked list is not cyclic and we have to return 0 and while loop is not executed. But if it is not null, the program will enter into while loop. Then inside while loop, I advance fast\_ptr by two node and slow\_ptr by one node as mentioned in the manual. After that if the fast\_ptr and slow\_ptr ever point to the same node, the list is cyclic and I write this logic in if statement.

The output obtained by running this code is:

