Module: R1: C Programming
Section: C Arrays, Pointers & Strings **Task:** 2.1

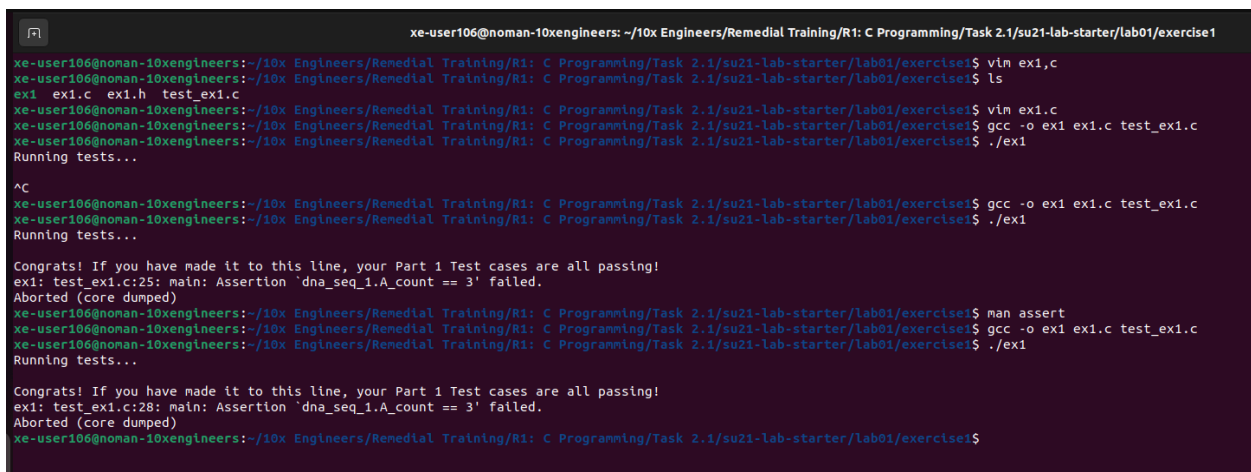## LAB 1: https://github.com/ImNomanCR7/fa21-lab-starter.git

Exercise 1:

1. **Number of Occurrences**
   Code Snippet:

```c
int num_occurrences(char *str, char letter) {
/* TODO: implement num_occurances */
int count = 0;
while (*str != '\0'){
if (letter == *str){
Count++;
}
Str++;
}
return count;
}
```

   Test Case:

```c
//Noman's Test Case
int num_o = num_occurrences(str, 'o');
assert(num_o == 2);
```

2. Nucleotide Occurrences
Code Snippet:

```c
void compute_nucleotide_occurrences(DNA_sequence *dna_seq) {
    /* TODO: implement compute_nucleotide_occurances */
    int count[4] = {0, 0, 0, 0};

    for (int i=0; i < strlen(dna_seq->sequence); i++){
      switch (dna_seq -> sequence[i]){

      case 'A':
      count[0]++;
      break;

      case 'C':
      count[1]++;
      break;

      case 'G':
      count[2]++;
      break;

      case 'T':
      count[3]++;
      break;

      default:
      break;
      }
    }

      dna_seq -> A_count = count[0];
      dna_seq -> C_count = count[1];
      dna_seq -> G_count = count[2];
      dna_seq -> T_count = count[3];

    }
```

Test Case:

```c
//Noman's Test case 2
    DNA_sequence dna_seq_3;
    strcpy(dna_seq_3.sequence, "AACGTACAGTTT");
```

```
        compute_nucleotide_occurrences(&dna_seq_3);
        assert(dna_seq_3.A_count == 4);
        assert(dna_seq_3.C_count == 2);
        assert(dna_seq_3.G_count == 2);
        assert(dna_seq_3.T_count == 4);
```

```
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Task 2.1/su21-lab-starter/lab01/exercise1$ gcc -o ex1 ex1.c test_ex1.c
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Task 2.1/su21-lab-starter/lab01/exercise1$ ./ex1
Running tests...

Congrats! If you have made it to this line, your Part 1 Test cases are all passing!
ex1: test_ex1.c:29: main: Assertion `dna_seq_1.A_count == 3' failed.
Aborted (core dumped)
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Task 2.1/su21-lab-starter/lab01/exercise1$ gcc -o ex1 ex1.c test_ex1.c
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Task 2.1/su21-lab-starter/lab01/exercise1$ ./ex1
Running tests...

Congrats! If you have made it to this line, your Part 1 Test cases are all passing!
Congrats! If you have made it to this line, your Part 2 Test cases are all passing!
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Task 2.1/su21-lab-starter/lab01/exercise1$ gcc -o ex1 ex1.c test_ex1.c
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Task 2.1/su21-lab-starter/lab01/exercise1$ ./ex1
Running tests...

Congrats! If you have made it to this line, your Part 1 Test cases are all passing!
Congrats! If you have made it to this line, your Part 2 Test cases are all passing!
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Task 2.1/su21-lab-starter/lab01/exercise1$
```

Exercise 2:

1. **Part 1 & Part 2**

Removed compiler warnings at first and added assert() statements to debug which part of the code is causing problems, the updated code snippet after assertions is:

```
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);
}
```

Output:

2. **Part 3**

   After resolving the problem with assert length,another issue with assertion name comes up. I followed the same steps to identify and address the issue using cgdb.

   After inspection, it appeared that the return value for the function **check_name** was not giving the required logic. Here's the corrected logic:

```c
bool check_name(const char *first_name, const char *last_name, const char *password) {
    /* Type "man strstr" in your terminal to learn what strstr does!
          To exit the man pages, press 'q' */
    /* Hint: a NULL pointer will evaluate to False in a logical
    statement while a non-NULL pointer
          will evaluate to True */
    const char *first = strstr(password, first_name);
    const char *last = strstr(password, last_name);
    return !(first || last);
}
```

3. **Part 4**

   After removing the assert statements, I encountered an error with
   check_number. Upon inspection, it was revealed that we passed numbers 0-9
   instead of the string '0' to '9'. Fixed that and recompiled the program.

```c
bool check_number(const char *password) {
while (*password != '\0') {
if (check_range(*password, '0', '9')) {
return true;
}
++password;
}
return false;
}
```

   Last bug which needs to be fixed is with **check_upper** function which is not
   behaving normally. Upon inspection, it was revealed that the bug originated from
   check_range function. Hence, it will take the following fix:

```c
bool check_range(char letter, char lower, char upper) {
    bool is_in_range = (letter >= lower && letter <= upper);
    return is_in_range;
}
```

Exercise 3:

1. **reverse_list SIGSEGV:**

   Code Snippet:

```c
void reverse_list(struct Node **head) {
    if (head == NULL || *head == NULL) {
        return;
    }
    struct Node *curr = *head;
    struct Node *next = (*head)->next;
    curr->next = NULL;
    while (next != NULL) {
        struct Node *temp = next->next;
        next->next = curr;
        curr = next;
        next = temp;
    }
    *head = curr;0
}
```


2. **add_to_back SIGSEGV:**

   Code Snippet:

```c
void add_to_back(Node **head, int data) {
    if (head == NULL) {
        return;
```

```
    }
    Node *new_node = create_node(data);
    if (*head == NULL){
       *head=new_node;
       return;
    }

    Node *prev;
    for (Node *curr = *head; curr != NULL; curr = curr->next) {
        prev = curr;
    }
    prev->next = new_node;
}
```

```
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Task 2.1/su21-lab-starter/lab01/exercise3$ gcc -g linked_list.c test_linked_list.c -o linked_list
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Task 2.1/su21-lab-starter/lab01/exercise3$ ./linked_list
Running tests...

Congrats! You have passed the reverse_list test!

Congrats! All of the test cases passed!
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Task 2.1/su21-lab-starter/lab01/exercise3$
```

Exercise 4:
1. Cyclic Linked-Lists:
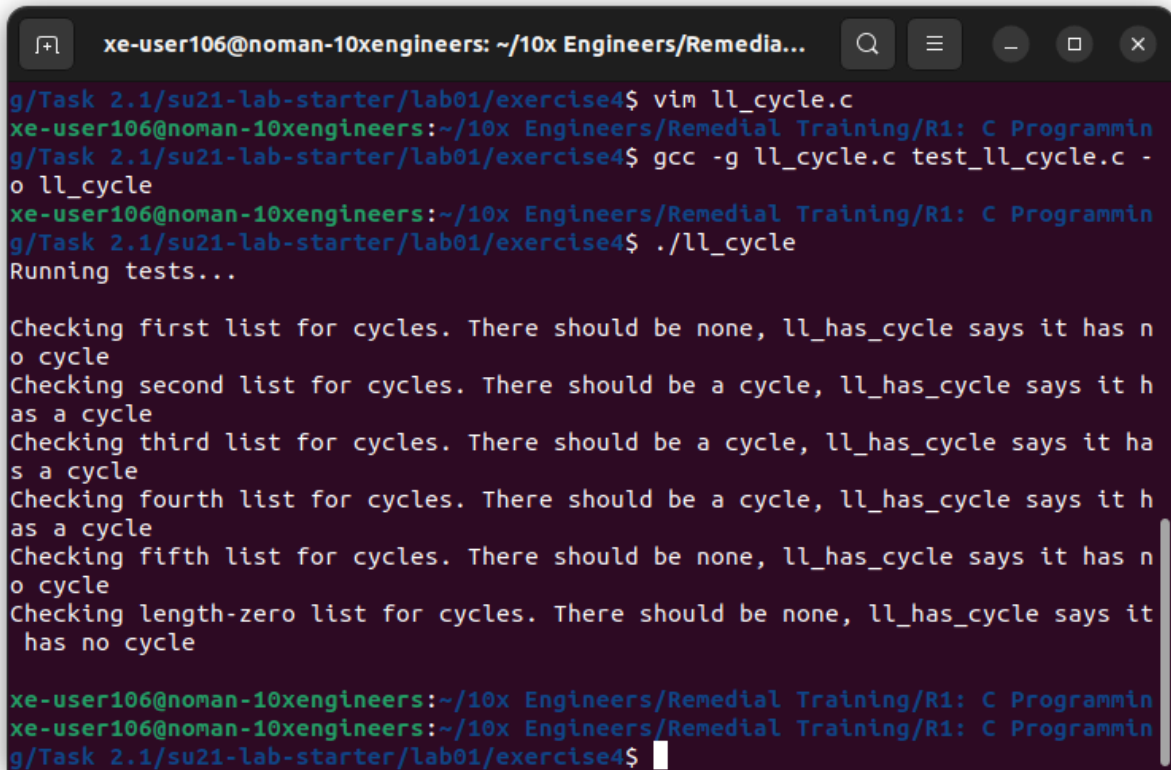### Code Snippet:

```c
#include <stddef.h>
#include "ll_cycle.h"

int ll_has_cycle(node *head) {
    /* TODO: Implement ll_has_cycle */
    node *fast_ptr = head;
    node *slow_ptr = head;

    while (fast_ptr != NULL && fast_ptr->next != NULL){
      fast_ptr = fast_ptr->next->next;
      slow_ptr = slow_ptr->next;

      if (fast_ptr == slow_ptr){
      //Has a Cycle
      return 1;
      }
    }
    //Acyclic
    return 0;
}
```

Repo URL: https://github.com/ImNomanCR7/fa21-lab-starter.git