QUEUE ASSIGNMENTS

1. Refer the example code of queue using array in socodery. Modify the data structure as below and accordingly update the code to add support to keep MAX_QUEUE_SIZE user input strings read at runtime. Consider a configurable queue capacity

```
#define MAX_QUEUE_SIZE 5
typedef struct queue
{
int arr[MAX_QUEUE_SIZE];
char **data;
int front;
int rear;
}queue_t;
```

Sol:

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


#define MAX_QUEUE_SIZE 5

#define MAX_STRING_LENGTH 100

#define END -1

#define SUCCESS 0

#define FAILURE 1


typedef struct queue {

   int arr[MAX_QUEUE_SIZE];

   char **data;

   int front;

   int rear;

} queue_t;


void init_queue(queue_t *q) {

   q->front = END;
```

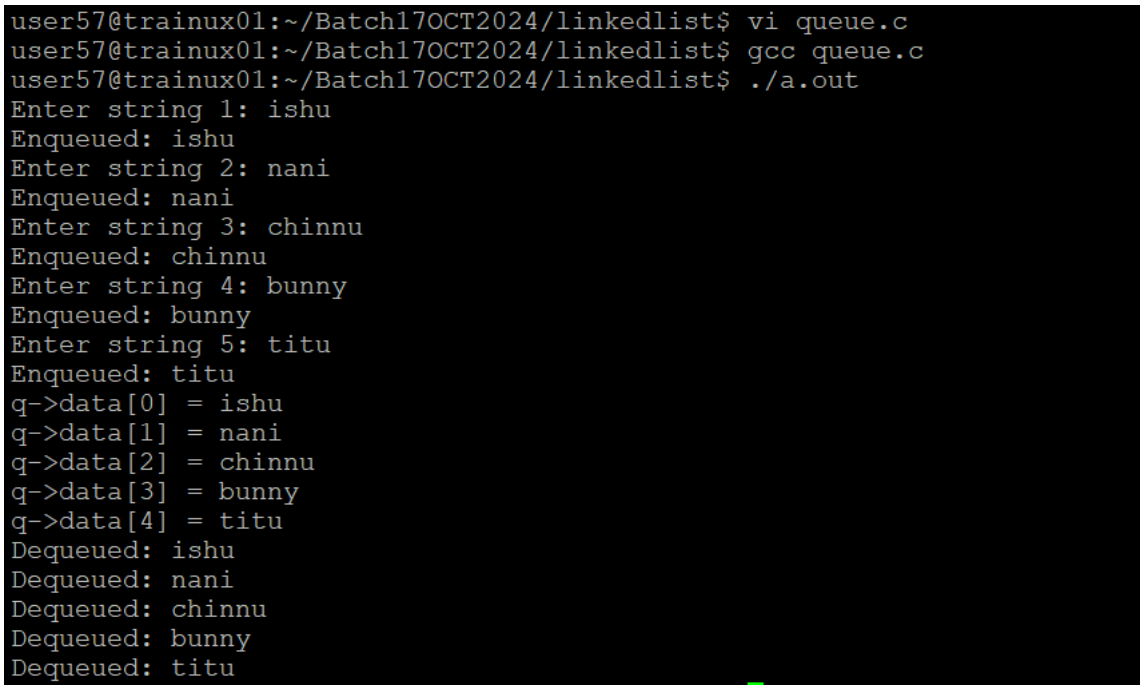
```
    q->rear = END;
    q->data = (char **)malloc(MAX_QUEUE_SIZE * sizeof(char *));
    for (int i = 0; i < MAX_QUEUE_SIZE; i++) {
        q->data[i] = (char *)malloc(MAX_STRING_LENGTH * sizeof(char));
    }
}

int check_empty(queue_t *q) {
    return (q->front == END);
}

int check_full(queue_t *q) {
    return (q->front == ((q->rear + 1) % MAX_QUEUE_SIZE));
}

int enqueue(queue_t *q, char *str) {
    if (check_full(q)) {
        return FAILURE;
    }

    if (check_empty(q)) {
        q->front = 0;
        q->rear = 0;
    } else {
        q->rear = (q->rear + 1) % MAX_QUEUE_SIZE;
    }

    strcpy(q->data[q->rear], str);
    return SUCCESS;
}
```

```
int dequeue(queue_t *q, char *p_str) {
    if (check_empty(q)) {
        return FAILURE;
    }

    strcpy(p_str, q->data[q->front]);

    if (q->front == q->rear) {
        q->front = END;
        q->rear = END;
    } else {
        q->front = (q->front + 1) % MAX_QUEUE_SIZE;
    }

    return SUCCESS;
}

void display(queue_t *q) {
    if (check_empty(q)) {
        printf("Queue is empty\n");
        return;
    }

    int index = q->front;
    do {
        printf("q->data[%d] = %s\n", index, q->data[index]);
        index = (index + 1) % MAX_QUEUE_SIZE;
    } while (index != (q->rear + 1) % MAX_QUEUE_SIZE);
}

void free_queue(queue_t *q) {
```

```
    for (int i = 0; i < MAX_QUEUE_SIZE; i++) {
        free(q->data[i]);
    }
    free(q->data);
}

int main() {
    queue_t q;
    init_queue(&q);

    char input[MAX_STRING_LENGTH];

    for (int i = 0; i < MAX_QUEUE_SIZE; i++) {
        printf("Enter string %d: ", i + 1);
        fgets(input, MAX_STRING_LENGTH, stdin);
        input[strcspn(input, "\n")] = 0;
        if (enqueue(&q, input) == SUCCESS) {
            printf("Enqueued: %s\n", input);
        } else {
            printf("Queue is full. Cannot enqueue.\n");
        }
    }

    display(&q);

    char dequeued_str[MAX_STRING_LENGTH];
    for (int i = 0; i < MAX_QUEUE_SIZE; i++) {
        if (dequeue(&q, dequeued_str) == SUCCESS) {
            printf("Dequeued: %s\n", dequeued_str);
        } else {
            printf("Queue is empty. Cannot dequeue.\n");
```

```
    }
  }

  free_queue(&q);

  return 0;
}
```

Output:



2. Refer the example code of queue using list in socodery. Modify the data structure as below and accordingly update the code to add support to keep string data read from user (of maximum length 80 characters) at runtime. Ensure 0 memory leaks.

```
typedef struct node_q
{
Int data;
char *data; //data read from user at runtime
struct node_q *next;
}node;
```

Sol:

```
#include <stdio.h>
```

```
#include <stdlib.h>
#include <string.h>

#define MAX_STRING_LENGTH 80

typedef struct node_q {
   char *data;
   struct node_q *next;
} node;

int isempty(node *front) {
   return front == NULL ? 1 : 0;
}

void enqueue(node **front, node **rear, char *value) {
   node *newnode;
   newnode = (node*) malloc(sizeof(node));
   if (newnode == NULL) {
      printf("Memory not available\n");
      exit(1);
   }

   newnode->data = (char *) malloc(strlen(value) + 1);
   if (newnode->data == NULL) {
      printf("Memory not available for string\n");
      exit(1);
   }

   strcpy(newnode->data, value);
   newnode->next = NULL;
```

```
    if (isempty(*front)) {
        *front = newnode;
        *rear = newnode;
    } else {
        (*rear)->next = newnode;
        *rear = newnode;
    }
}

int dequeue(node **front, node **rear) {
    node *temp_node;

    if (isempty(*front)) {
        printf("Queue Underflow\n");
        return -1;
    }

    temp_node = *front;
    *front = (*front)->next;

    printf("Element deleted: %s\n", temp_node->data);
    free(temp_node->data);
    free(temp_node);

    if (*front == NULL) {
        *rear = NULL;
    }
    return 0;
}

void display(node *front) {
```

```
    node *current = front;
    if (isempty(front)) {
        printf("The Queue is empty\n");
        return;
    }

    printf("Queue contents:\n");
    while (current != NULL) {
        printf("%s\n", current->data);
        current = current->next;
    }
}

void free_queue(node *front) {
    node *current = front;
    while (current != NULL) {
        node *temp = current;
        current = current->next;
        free(temp->data);
        free(temp);
    }
}

int main() {
    node *front = NULL, *rear = NULL;
    char input[MAX_STRING_LENGTH];

    printf("Enter strings to enqueue into the queue:\n");

    for (int i = 1; i <= 3; i++) {
        printf("Enter string %d: ", i);
```

```
    fgets(input, sizeof(input), stdin);
    input[strcspn(input, "\n")] = '\0';

    enqueue(&front, &rear, input);
  }

  display(front);

  while (!isempty(front)) {
    dequeue(&front, &rear);
  }

  return 0;
}
```

Output: