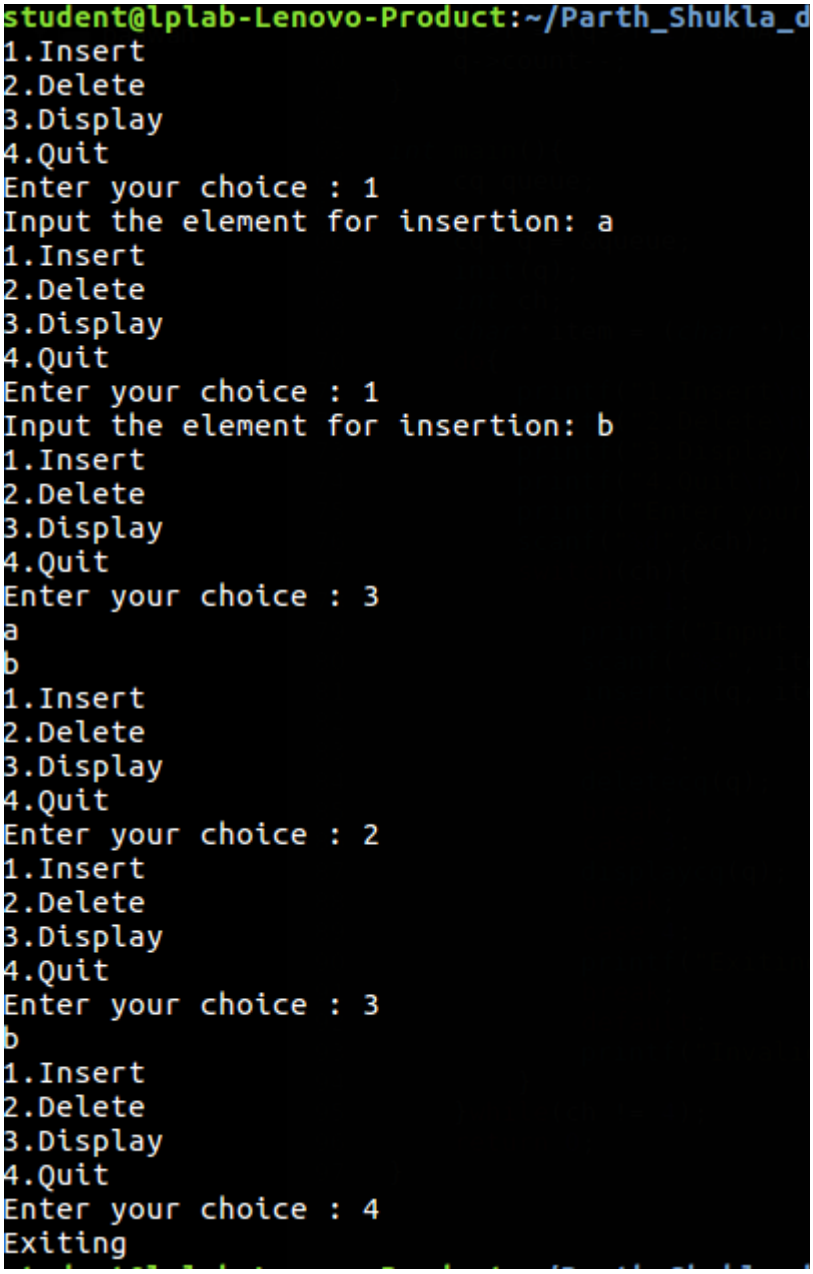
Week 5

1)

// Implement a circular queue of Strings using structures. Include functions insertcq, deletecq and displaycq.

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#define MAX 10

typedef struct{
	char a[MAX][MAX];
	int f;
	int r;
	int count;
}cq;

void init(cq *q){
	q->f = 0;
	q->r = -1;
	q->count = 0;
}

int isFull(cq *q){
	if(q->count == MAX)
		return 1;
	return 0;
}

int isEmpty(cq *q){
	if(q->count == 0)
		return 1;
	return 0;
}

void displaycq(cq *q){
	if (isEmpty(q)) {
		printf("Queue is empty\n");
		return;
	}
	for (int i = q->f; i != q->r; i = (i + 1) % MAX){
		printf("%s\n", q->a[i]);
	}
	printf("%s\n", q->a[q->r]);
}

void insertcq(cq *q, char* s){
	if(isFull(q)){
		printf("Queue is full\n"); return;
	}
```

```
	q->r = (q->r + 1) % MAX;
	strcpy(q->a[q->r], s);
	q->count++;
}

void deletecq(cq *q){
	if (isEmpty(q)) {
	printf("Queue is empty\n"); return;
	}
	int temp = q->f;
	q->f = (q->f+1) % MAX;
	q->count--;
}

int main(){
	cq queue;

	cq* q = &queue;
	init(q);
	int ch;
	char* item = (char *)calloc(MAX, sizeof(char));
	do{
		printf("1.Insert\n");
		printf("2.Delete\n");
		printf("3.Display\n");
		printf("4.Quit\n");
		printf("Enter your choice: ");
		scanf("%d",&ch);
		switch(ch){
			case 1:
			printf("Input the element for insertion: ");
			scanf("%s", item);
			insertcq(q, item);
			break;
			case 2:
			deletecq(q);
			break;
			case 3:
			displaycq(q);
			break;
			case 4:
			printf("Exiting\n");
			break;
			default:
			printf("Invalid choice\n");
		}
	}while(ch != 4);
	return 0;
}
```



2)

// Implement two circular queues of integers in a single array where first queue will run from 0 to N/2
// and second queue will run from N/2+1 to N-1 where N is the size of the array.

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#define MAX 10

typedef struct{
```

```
	int* q;
	int size;
	int f1, f2;
	int r1, r2;
	int count1, count2;
}cqd;

void cqd_init(cqd* c, int n) {
	c->q = (int*)calloc(n, sizeof(int));
	c->size = n;
	c->r1 = -1;
	c->f1 = -1;
	c->r2 = (n / 2);
	c->f2 = (n / 2);
	c->count1 = 0;
	c->count2 = 0;
}

int cqd_empty(cqd* c, int qno) {
	if (qno == 1) {
		return !(c->count1);
	}
	else if (qno == 2) {
		return !(c->count2);
	}
}

int cqd_full(cqd* c, int qno) {
	if (qno == 1){
		return c->count1 == (c->size / 2) ? 1 : 0;
	}
	else if (qno == 2){
		return c->count2 == (c->size / 2) ? 1 : 0;
	}
}

void insertcqd(cqd* c, int qno, int e){
	if (qno == 1) {
		if (cqd_full(c, qno)) {
			printf("Queue 1 is full");
			return;
		}
		c->r1 = (c->r1 + 1) % (c->size / 2);
		c->q[c->r1] = e;
		if(cqd_empty(c, qno)){
			c->f1=c->r1;
		}
		c->count1++;
		return;
	}
	else if (qno == 2) {
		if (cqd_full(c, qno))
```

```
		{
			printf("Queue 2 is full");
			return;
		}
		c->r2 = (c->r2 + 1) % (c->size / 2) + c->size / 2;
		c->q[c->r2] = e;
		if(cqd_empty(c, qno)){
			c->f2=c->r2;
		}
		c->count2++;
		return;
	}
}

int deletecqd(cqd* c, int qno) {
	if (qno == 1) {
		if (cqd_empty(c, qno)) {
		printf("Queue 1 is empty\n");
		return 0;
		}
		int element = c->q[c->f1];
		c->f1 = (c->f1 + 1) % (c->size / 2);
		c->count1--;
		return element;
	}
	else if (qno == 2) {
		if (cqd_empty(c, qno)){
			printf("Queue 2 is empty\n");
			return 0;
		}
		int element = c->q[c->f2];
		c->f2 = (c->f2 + 1) % (c->size / 2) + (c->size / 2);
		c->count2--;
		return element;
	}
}

void display(cqd* c, int qno) {
	printf("Queue:\n");
	if (qno == 1) {
		if (cqd_empty(c, qno)) {
			printf("Queue 1 is empty");
			return;
		}
		for (int i = c->f1; i != c->r1; i = (i + 1) % (c->size / 2))
		{
			printf("%d, ", c->q[i]);
		}
		printf("%d\n", c->q[c->r1]);
	}
	else if (qno == 2) {
		if (cqd_empty(c, qno))
```

```
		{
			printf("Queue 2 is empty \n");
			return;
		}
		for (int i = c->f2; i != c->r2; i = (i + 1) % (c->size / 2) + (c->size / 2))
		{
			printf("%d, ", c->q[i]);
		}
		printf("%d\n", c->q[c->r2]);
	}
}

int main() {
	cqd cqdouble;
	cqd* c = &cqdouble;
	cqd_init(c, MAX);
	int n, ch, qno;
	do{
		printf("1 - Insert");
		printf("\n2 - Delete");
		printf("\n3 - Display");
		printf("\n4 - Exit");
		printf("\nEnter queue number and choice: ");
		scanf("%d %d", &qno, &ch);
		switch (ch)
		{
			case 1:
			printf("Enter value to be inserted: ");
			scanf("%d",&n);
			insertcqd(c, qno, n);
			break;
			case 2:
			deletecqd(c, qno);
			break;
			case 3:
			display(c, qno);
			break;
			case 4:
			printf("Exiting\n");
			break;
			default:
			printf("Invalid option");
		}
	}while(ch != 4);
	return 0;
}
```

```
student@lplab-Lenovo-Product:~/Parth_Shukla_dsa/lab5$ ./cqd
1 - Insert
2 - Delete
3 - Display
4 - Exit
Enter queue number and choice: 1 1
Enter value to be inserted: 1
1 - Insert
2 - Delete
3 - Display
4 - Exit
Enter queue number and choice: 1 1
Enter value to be inserted: 2
1 - Insert
2 - Delete
3 - Display
4 - Exit
Enter queue number and choice: 1 2
1 - Insert
2 - Delete
3 - Display
4 - Exit
Enter queue number and choice: 1 3
Queue:
2
1 - Insert
2 - Delete
3 - Display
4 - Exit
Enter queue number and choice: 2 1
Enter value to be inserted: 3
1 - Insert
2 - Delete
3 - Display
4 - Exit
Enter queue number and choice: 2 3
Queue:
3
```

3)

// Implement a queue with two stacks without transferring the elements of the second stack back to stack one.
// (use stack1 as an input stack and stack2 as an output stack).

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
        int data;
        struct node *next;
};
void push(struct node** top, int data);
int pop(struct node** top);
```

```
struct queue
{
	struct node *stack1;
	struct node *stack2;
};

void enqueue(struct queue *q, int x)
{
	push(&q->stack1, x);
}
void dequeue(struct queue *q)
{
	int x;
	if (q->stack1 == NULL && q->stack2 == NULL) {
	printf("queue is empty");
	return;
	}
	if (q->stack2 == NULL) {
		while (q->stack1 != NULL) {
			x = pop(&q->stack1);
			push(&q->stack2, x);
		}
	}
	x = pop(&q->stack2);
	printf("Deleted: %d\n", x);
}

void push(struct node** top, int data) {
	struct node* newnode = (struct node*) malloc(sizeof(struct node));
	if (newnode == NULL) {
		printf("Stack overflow \n");
		return;
	}
	newnode->data = data;
	newnode->next = (*top);
	(*top) = newnode;
}

int pop(struct node** top){
	int buff;struct node *t;
	if (*top == NULL) {
		printf("Stack underflow \n");
	}
	else {
		t = *top;
		buff = t->data;
		*top = t->next;
		free(t);
		return buff;
	}
}
```

```
void display(struct node *top1,struct node *top2)
{
	while (top1 != NULL) {
	printf("%d ", top1->data);
	top1 = top1->next;
	}
	while (top2 != NULL) {
	printf("%d ", top2->data);
	top2 = top2->next;
	}
	printf("\n");
}
int main(){
	struct queue *q = (struct queue*)malloc(sizeof(struct queue));
	int f = 0, a;
	// char ch = 'y';
	q->stack1 = NULL;
	q->stack2 = NULL;
	do {
		printf("1.Insert\n");
		printf("2.Delete\n");
		printf("3.Display\n");
		printf("4.Quit\n");
		printf("Enter your choice: ");
		scanf("%d", &f);
		switch(f) {
		case 1 : printf("Enter element: ");
		scanf("%d", &a);
		enqueue(q, a);
		break;
		case 2 : dequeue(q);
		break;
		case 3 : display(q->stack1, q->stack2);
		break;
		case 4 : printf("Exiting\n");
		break;
		default : printf("invalid\n");
		break;
		}
	}while(f != 4);
	return 0;
}
```

```
student@lplab-Lenovo-Product:~/Parth_Shukla_dsa/lab5$ ./stack2
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice: 1
Enter element: 1
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice: 1
Enter element: 2
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice: 1
Enter element: 3
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice: 3
3 2 1
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice: 2
Deleted: 1
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice: 3
2 3
1.Insert
2.Delete
```