1.//INFIX TO POSTFIX CONVERSION

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int prec(char c) {
    if (c == '^')
        return 3;
    else if (c == '/' || c == '*')
        return 2;
    else if (c == '+' || c == '-')
        return 1;
    else
        return -1;
}
char associativity(char c) {
    if (c == '^')
        return 'R';
    return 'L';
}
void infixToPostfix(char s[]) {
    char result[1000];
    int resultIndex = 0;
    int len = strlen(s);
    char stack[1000];
```

```
    int stackIndex = -1;
    for (int i = 0; i < len; i++) {
        char c = s[i];
        if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') || (c >= '0' && c <= '9')) {
            result[resultIndex++] = c;
        }
        else if (c == '(') {
            stack[++stackIndex] = c;
        }
        else if (c == ')') {
            while (stackIndex >= 0 && stack[stackIndex] != '(') {
                result[resultIndex++] = stack[stackIndex--];
            }
            stackIndex--;
        }
        else {
            while (stackIndex >= 0 && (prec(s[i]) < prec(stack[stackIndex]) ||prec(s[i])
== prec(stack[stackIndex]) &&associativity(s[i]) == 'L')) {
                result[resultIndex++] = stack[stackIndex--];
            }
            stack[++stackIndex] = c;
        }
    }
    while (stackIndex >= 0) {
```

```
        result[resultIndex++] = stack[stackIndex--];
    }

    result[resultIndex] = '\0';
    printf("%s\n", result);
}
int main() {
    char exp[] = "a+b*c+d";
    infixToPostfix(exp);
    return 0;
}
```

OUTPUT:

abc*+d+

2.ARRAY IMPLIMENTATION IN QUEUE:

```
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
struct Queue {
        int front, rear, size;
        unsigned capacity;
        int* array;
};
struct Queue* createQueue(unsigned capacity)
{
```

```
        struct Queue* queue = (struct Queue*)malloc(
                sizeof(struct Queue));
        queue->capacity = capacity;
        queue->front = queue->size = 0;
        queue->rear = capacity - 1;
        queue->array = (int*)malloc(
                queue->capacity * sizeof(int));
        return queue;
}
int isFull(struct Queue* queue)
{
        return (queue->size == queue->capacity);
}
int isEmpty(struct Queue* queue)
{
        return (queue->size == 0);
}
void enqueue(struct Queue* queue, int item)
{
        if (isFull(queue))
                return;
        queue->rear = (queue->rear + 1)
                                % queue->capacity;
        queue->array[queue->rear] = item;
```

```
        queue->size = queue->size + 1;
        printf("%d enqueued to queue\n", item);
}
int dequeue(struct Queue* queue)
{
        if (isEmpty(queue))
                return INT_MIN;
        int item = queue->array[queue->front];
        queue->front = (queue->front + 1)
                                % queue->capacity;
        queue->size = queue->size - 1;
        return item;
}
int front(struct Queue* queue)
{
        if (isEmpty(queue))
                return INT_MIN;
        return queue->array[queue->front];
}
int rear(struct Queue* queue)
{
        if (isEmpty(queue))
                return INT_MIN;
        return queue->array[queue->rear];
```

```
}
int main()
{
        struct Queue* queue = createQueue(1000);
        enqueue(queue, 10);
        enqueue(queue, 20);
        enqueue(queue, 30);
        enqueue(queue, 40);
        printf("%d dequeued from queue\n\n",
                dequeue(queue));
        printf("Front item is %d\n", front(queue));
        printf("Rear item is %d\n", rear(queue));
        return 0;
}
```

OUTPUT:

```
10 enqueued to queue
20 enqueued to queue
30 enqueued to queue
40 enqueued to queue
10 dequeued from queue
Front item is 20
Rear item is 40
```

3.LINKED LIST IMLIMENTATION IN QUEUE:

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

```
#include <stdlib.h>
struct QNode {
	int key;
	struct QNode* next;
};
struct Queue {
	struct QNode *front, *rear;
};
struct QNode* newNode(int k)
{
	struct QNode* temp
		= (struct QNode*)malloc(sizeof(struct QNode));
	temp->key = k;
	temp->next = NULL;
	return temp;
}
struct Queue* createQueue()
{
	struct Queue* q
		= (struct Queue*)malloc(sizeof(struct Queue));
	q->front = q->rear = NULL;
	return q;
}
void enQueue(struct Queue* q, int k)
```

```
{
        struct QNode* temp = newNode(k);
        if (q->rear == NULL) {
                q->front = q->rear = temp;
                return;
        }
        q->rear->next = temp;
        q->rear = temp;
}
void deQueue(struct Queue* q)
{
                if (q->front == NULL)
                return;

        struct QNode* temp = q->front;

        q->front = q->front->next;
        if (q->front == NULL)
                q->rear = NULL;

        free(temp);
}
int main()
{
```

```
    struct Queue* q = createQueue();
    enQueue(q, 10);
    enQueue(q, 20);
    deQueue(q);
    deQueue(q);
    enQueue(q, 30);
    enQueue(q, 40);
    enQueue(q, 50);
    deQueue(q);
    printf("Queue Front : %d \n", ((q->front != NULL) ? (q->front)->key : -1));
    printf("Queue Rear : %d", ((q->rear != NULL) ? (q->rear)->key : -1));
    return 0;
}
```

OUTPUT:

```
Queue Front : 40
Queue Rear : 50
```