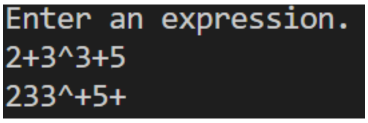
1)Infix to postfix

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


// Function to return precedence of operators

int prec(char c) {

        if (c == '^')

                return 3;

        else if (c == '/' || c == '*')

                return 2;

        else if (c == '+' || c == '-')

                return 1;

        else

                return -1;

}


// Function to return associativity of operators

char associativity(char c) {

        if (c == '^')

                return 'R';

        return 'L'; // Default to left-associative

}


// The main function to convert infix expression to postfix expression

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 the scanned character is an operand, add it to the output string.
        if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') || (c >= '0' && c <= '9')) {
            result[resultIndex++] = c;
        }
        // If the scanned character is an ‘(‘, push it to the stack.
        else if (c == '(') {
            stack[++stackIndex] = c;
        }
        // If the scanned character is an ‘)’, pop and add to the output string from the stack
        // until an ‘(‘ is encountered.
        else if (c == ')') {
            while (stackIndex >= 0 && stack[stackIndex] != '(') {
                result[resultIndex++] = stack[stackIndex--];
            }
            stackIndex--; // Pop '('
        }
        // If an operator is scanned
        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;
        }
    }
```

```
        // Pop all the remaining elements from the stack

        while (stackIndex >= 0) {

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

        }


        result[resultIndex] = '\0';

        printf("%s\n", result);

}


// Driver code

int main() {

        char exp[] = "a+b*(c^d-e)^(f+g*h)-i";


        // Function call

        infixToPostfix(exp);


        return 0;

}
```

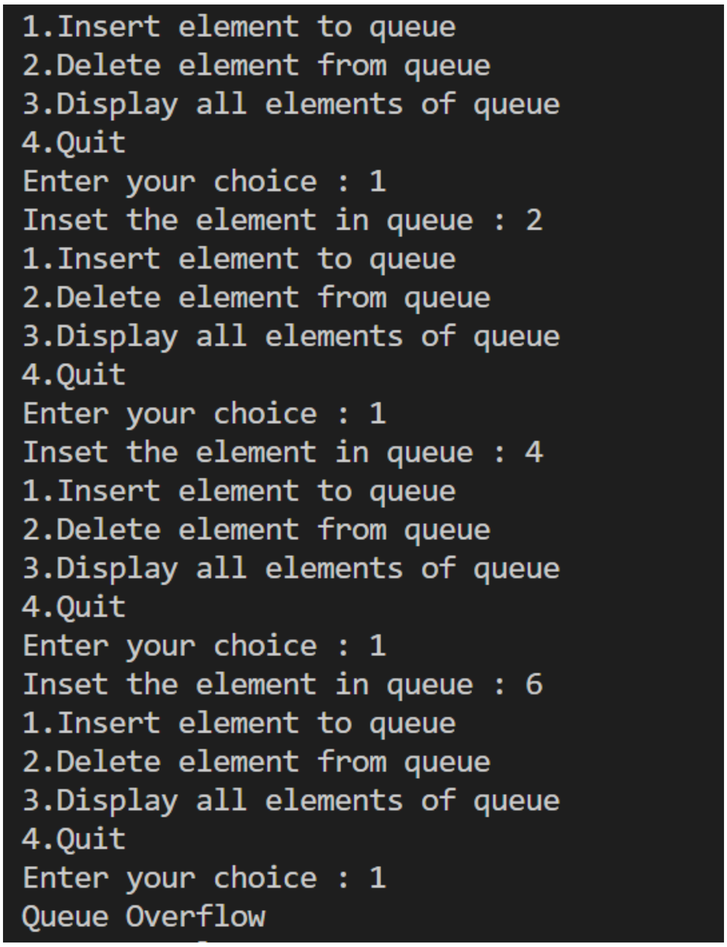


```
Enter an expression.
2+3^3+5
233^+5+
```

2) Queue implementation

```
/*

 * C Program to Implement a Queue using an Array

 */

#include <stdio.h>


#define MAX 3
```

```
void insert();
void delete();
void display();
int queue_array[MAX];
int rear = - 1;
int front = - 1;
main()
{
    int choice;
    while (1)
    {
        printf("1.Insert element to queue \n");
        printf("2.Delete element from queue \n");
        printf("3.Display all elements of queue \n");
        printf("4.Quit \n");
        printf("Enter your choice : ");
        scanf("%d", &choice);
        switch (choice)
        {
            case 1:
            insert();
            break;
            case 2:
            delete();
            break;
            case 3:
            display();
            break;
            case 4:
            exit(1);
            default:
            printf("Wrong choice \n");
        } /* End of switch */
    } /* End of while */
} /* End of main() */

void insert()
{
    int add_item;
    if (rear == MAX - 1)
    printf("Queue Overflow \n");
    else
    {
        if (front == - 1){
        /*If queue is initially empty */
        front = 0;}
        printf("Inset the element in queue : ");
        scanf("%d", &add_item);
        rear = rear + 1;
        queue_array[rear] = add_item;
    }
} /* End of insert() */

void delete()
{
    if (front == - 1 || front > rear)
    {
        printf("Queue Underflow \n");
        return ;
    }
    else
    {
        printf("Element deleted from queue is : %d\n", queue_array[front]);
        front = front + 1;
    }
} /* End of delete() */

void display()
{
    int i;
    if (front == - 1)
        printf("Queue is empty \n");
    else
    {
        printf("Queue is : \n");
        for (i = front; i <= rear; i++)
            printf("%d ", queue_array[i]);
        printf("\n");
    }
```

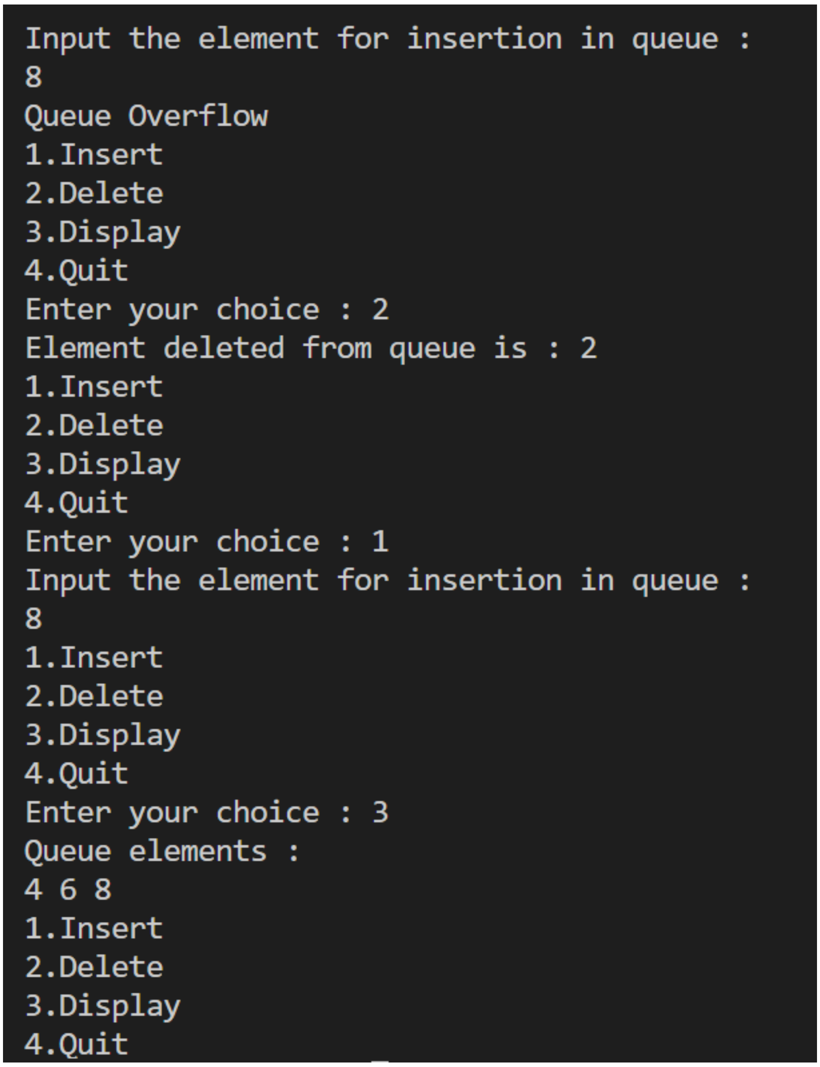


```
1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 1
Inset the element in queue : 2
1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 1
Inset the element in queue : 4
1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 1
Inset the element in queue : 6
1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 1
Queue Overflow
```

3) Circular queue implementation

```
#include<stdio.h>

# define MAX 5

int cqueue_arr[MAX];

int front = -1;

int rear = -1;

void insert(int item)

{

    if((front == 0 && rear == MAX-1) || (front == rear+1))

    {

        printf("Queue Overflow \n");

        return;

    }

    if(front == -1)
    {
        front = 0;
        rear = 0;
    }
    else
    {
        if(rear == MAX-1)
            rear = 0;
        else
            rear = rear+1;
    }
    cqueue_arr[rear] = item ;
}
void deletion()
{
    if(front == -1)
    {
        printf("Queue Underflow\n");
        return ;
    }
    printf("Element deleted from queue is : %d\n",cqueue_arr[front]);
    if(front == rear)
    {
        front = -1;
        rear=-1;
    }
    else
    {
        if(front == MAX-1)
            front = 0;
        else
            front = front+1;
    }
}
```

```c
void display()
{
    int front_pos = front,rear_pos = rear;
    if(front == -1)
    {
        printf("Queue is empty\n");
        return;
    }
    printf("Queue elements :\n");
    if( front_pos <= rear_pos )
        while(front_pos <= rear_pos)
        {
            printf("%d ",cqueue_arr[front_pos]);
            front_pos++;
        }
    else
    {
        while(front_pos <= MAX-1)
        {
            printf("%d ",cqueue_arr[front_pos]);
            front_pos++;
        }
        front_pos = 0;
        while(front_pos <= rear_pos)
        {
            printf("%d ",cqueue_arr[front_pos]);
            front_pos++;
        }
    }
    printf("\n");
}
int main()
{
    int choice,item;
    do
    {
        printf("1.Insert\n");
        printf("2.Delete\n");
        printf("3.Display\n");
        printf("4.Quit\n");
        printf("Enter your choice : ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1 :
            {
                printf("Input the element for insertion in queue : \n");
                scanf("%d", &item);
                insert(item);
                break;
            }
            case 2 :{
                deletion();
                break;}
            case 3:{
                display();
                break;}
            case 4:
                break;
            default:
                printf("Wrong choice\n");
        }
    }while(choice!=4);
    return 0;
}
```



```
Input the element for insertion in queue :
8
Queue Overflow
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice : 2
Element deleted from queue is : 2
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice : 1
Input the element for insertion in queue :
8
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice : 3
Queue elements :
4 6 8
1.Insert
2.Delete
3.Display
4.Quit
```