1.Convert from infix to postfix using c program

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#define MAX 100
char stack[MAX];
int top = -1;
void push(char ch)
 {
    if (top == MAX - 1)
 {
        printf("Stack overflow\n");
        return;
    }
    stack[++top] = ch;
}
char pop()
{
    if (top == -1)
{
        printf("Stack underflow\n");
        return '\0';
    }
    return stack[top--];
}
int precedence(char ch)
{
    switch (ch)
 {
        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
            return 2;
        case '^':
            return 3;
    }
    return 0;
}
int isOperator(char ch)
{
    return (ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch == '^');
}
void infixToPostfix(char *infix, char *postfix)
{
    int i, j = 0;
    char ch;

    for (i = 0; infix[i] != '\0'; i++)
 {
        ch = infix[i];

        if (isdigit(ch) || isalpha(ch))
{
            postfix[j++] = ch;
        } else if (ch == '(')
 {
            push(ch);
        } else if (ch == ')')
```

```
{
            while (top != -1 && stack[top] != '(')
{
                postfix[j++] = pop();
            }
            if (top == -1) {
                printf("Mismatched parentheses\n");
                return;
            }
            pop();  // Pop the '('
        } else if (isOperator(ch))
 {
            while (top != -1 && precedence(stack[top]) >= precedence(ch))

{
                postfix[j++] = pop();
            }
            push(ch);
        }
    }

    while (top != -1)
 {
        if (stack[top] == '(')
 {
            printf("Mismatched parentheses\n");
            return;
        }
        postfix[j++] = pop();
    }
    postfix[j] = '\0';
}

int main()
 {
    char infix[MAX], postfix[MAX];

    printf("Enter infix expression: ");
    if (fgets(infix, sizeof(infix), stdin) == NULL)
{
        printf("Error reading input\n");
        return 1;
    }
    size_t len = strlen(infix);
    if (len > 0 && infix[len - 1] == '\n')
{
        infix[len - 1] = '\0';
    }

    infixToPostfix(infix, postfix);

    printf("Postfix expression: %s\n", postfix);

    return 0;
}
```

Output:

```
Enter infix expression: 5
Postfix expression: 5
```

2.Write a c program for array using queue.

Program:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100

typedef struct
{
   int front;
   int rear;
   int items[MAX];
} Queue;
void initializeQueue(Queue* q)
 {
   q->front = -1;
   q->rear = -1;
}

int isEmpty(Queue* q)
{
   return q->front == -1;
}
int isFull(Queue* q)
{
   return q->rear == MAX - 1;
}
void enqueue(Queue* q, int value)
 {
   if (isFull(q))
{
       printf("Queue is full\n");
       return;
   }
   if (isEmpty(q))
{
       q->front = 0;
   }
   q->rear++;
   q->items[q->rear] = value;
   printf("%d enqueued to queue\n", value);
}
int dequeue(Queue* q)
 {
   if (isEmpty(q))
{
       printf("Queue is empty\n");
       return -1;
   }
   int value = q->items[q->front];
   q->front++;
   if (q->front > q->rear)
{
       q->front = q->rear = -1;
   }
   printf("%d dequeued from queue\n", value);
   return value;
}
void displayQueue(Queue* q)
{
   if (isEmpty(q))
{
       printf("Queue is empty\n");
       return;
   }
   printf("Queue elements: ");
   for (int i = q->front; i <= q->rear; i++)
{
       printf("%d ", q->items[i]);
   }
   printf("\n");
}

int main()
 {
   Queue q;
   initializeQueue(&q);

   enqueue(&q, 10);
   enqueue(&q, 20);
   enqueue(&q, 30);
   displayQueue(&q);

   dequeue(&q);
   displayQueue(&q);

   dequeue(&q);
   displayQueue(&q);

   dequeue(&q);
   displayQueue(&q);

   dequeue(&q);
   return 0;
}
```

Output:

```
10 enqueued to queue
20 enqueued to queue
30 enqueued to queue
Queue elements: 10 20 30
10 dequeued from queue
Queue elements: 20 30
20 dequeued from queue
Queue elements: 30
30 dequeued from queue
Queue is empty
Queue is empty
```

3.Give a c program for Linked list using queue.

Program:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node
{
    int data;
    struct Node* next;
}
Node;
typedef struct
{
    Node* front;
    Node* rear;
} ;
Node* createNode(int data)
{
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (!newNode)
{
        printf("Memory allocation error\n");
        exit(1);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}
void initializeQueue(Queue* q)
{
    q->front = q->rear = NULL;
}
int isEmpty(Queue* q)
{
    return q->front == NULL;
}
void enqueue(Queue* q, int data)

    Node* newNode = createNode(data);
    if (q->rear == NULL) {
        q->front = q->rear = newNode;
        printf("%d enqueued to queue\n", data);
        return;
    }
    q->rear->next = newNode;
    q->rear = newNode;
    printf("%d enqueued to queue\n", data);
}
int dequeue(Queue* q)
 {
    if (isEmpty(q))
{
        printf("Queue is empty\n");
        return -1;
    }
    Node* temp = q->front;
    int data = temp->data;
    q->front = q->front->next;
    if (q->front == NULL) {
        q->rear = NULL;
    }
    free(temp);
    printf("%d dequeued from queue\n", data);
    return data;
}
void displayQueue(Queue* q)
{
    if (isEmpty(q))
{
        printf("Queue is empty\n");
        return;
    }
    Node* temp = q->front;
    printf("Queue elements: ");
    while (temp)
{
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main()
{
    Queue q;
    initializeQueue(&q);

    enqueue(&q, 10);
    enqueue(&q, 20);
    enqueue(&q, 30);
    displayQueue(&q);

    dequeue(&q);
    displayQueue(&q);

    dequeue(&q);
    displayQueue(&q);

    dequeue(&q);
    displayQueue(&q);

    dequeue(&q);

    return 0;
}
```

Output:

```
10 enqueued to queue
20 enqueued to queue
30 enqueued to queue
Queue elements: 10 20 30
10 dequeued from queue
Queue elements: 20 30
20 dequeued from queue
Queue elements: 30
30 dequeued from queue
Queue is empty
Queue is empty
```