@Write a function in C to count the number of nodes in Singly Linked list

```
#include <stdio.h>
#include <stdlib.h>
struct SinglyNode {
    int data;
    struct SinglyNode* next;
int countSinglyNodes(struct SinglyNode* head) {
    int count = 0;
    struct SinglyNode* current = head;
    while (current != NULL) {
        count++;
        current = current->next;
    }  return count;
}int main() {
 struct SinglyNode* head = (struct SinglyNode*)malloc(sizeof(struct
SinglyNode));
    struct SinglyNode* second = (struct
SinglyNode*)malloc(sizeof(struct SinglyNode));
    struct SinglyNode* third = (struct SinglyNode*)malloc(sizeof(struct
SinglyNode));
    head->data = 1; head->next = second;
    second->data = 2; second->next = third;
    third->data = 3; third->next = NULL; printf("Number of nodes in
Singly Linked List: %d\n", countSinglyNodes(head));
  free(head); free(second); free(third);   return 0;}
```

@Write a function in C to count the number of nodes in doublely Linked list

```
#include <stdio.h>
#include <stdlib.h>
struct DoublyNode {
    int data;
    struct DoublyNode* next;
    struct DoublyNode* prev;
};int countDoublyNodes(struct DoublyNode* head) {
    int count = 0;
    struct DoublyNode* current = head;
    while (current != NULL) {
        count++;
        current = current->next;
    }  return count;
}int main() {
    struct DoublyNode* head = (struct
DoublyNode*)malloc(sizeof(struct DoublyNode));
    struct DoublyNode* second = (struct
DoublyNode*)malloc(sizeof(struct DoublyNode));
    struct DoublyNode* third = (struct
DoublyNode*)malloc(sizeof(struct DoublyNode));
    head->data = 1; head->next = second; head->prev = NULL;
    second->data = 2; second->next = third; second->prev = head;
    third->data = 3; third->next = NULL; third->prev = second;
    printf("Number of nodes in Doubly Linked List: %d\n",
countDoublyNodes(head));
    free(head); free(second); free(third);
    return 0;}
```

@Write a C program to perform the operation on Singly linked list: i) Insert a new node at the end of the list. ii) Delete a node from the beginning of the list. iii) Search for a given node. iv)Display the list

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};void insertAtEnd(struct Node** head, int data);
void deleteFromBeginning(struct Node** head);
struct Node* searchNode(struct Node* head, int key);
void displayList(struct Node* head);
int main() {
    struct Node* head = NULL; // Initialize an empty list
    int choice, value, key;
    struct Node* result;
  while (1) {
        printf("\n--- Singly Linked List Operations ---\n");
        printf("1. Insert a node at the end\n");
        printf("2. Delete a node from the beginning\n");
        printf("3. Search for a node\n");
        printf("4. Display the list\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
  switch (choice) {
   case 1:  printf("Enter the value to insert: ");
                scanf("%d", &value);
          insertAtEnd(&head, value);  break;
   case 2:   deleteFromBeginning(&head);
                break;
 case 3: printf("Enter the value to search: ");
                scanf("%d", &key);
                result = searchNode(head, key);
                if (result != NULL) {
                    printf("Node with value %d found.\n", result->data);
                } else {
                    printf("Node with value %d not found.\n", key);
                }
                break;
            case 4:  displayList(head);
                break;
            case 5:   printf("Exiting...\n");
                return 0;
            default:
                printf("Invalid choice. Try again.\n"); } }}void
insertAtEnd(struct Node** head, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct
Node));
    newNode->data = data;
    newNode->next = NULL;
  if (*head == NULL) {
        *head = newNode;
        printf("Node inserted at the end.\n");
        return;
    }  struct Node* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }temp->next = newNode;
    printf("Node inserted at the end.\n");
}void deleteFromBeginning(struct Node** head) {
    if (*head == NULL) {
        printf("List is empty. Nothing to delete.\n");return; }
 struct Node* temp = *head;
    *head = (*head)->next;
    free(temp);
    printf("Node deleted from the beginning.\n");
}struct Node* searchNode(struct Node* head, int key) {
    struct Node* current = head;
    while (current != NULL) {
        if (current->data == key) {
            return current;
        } current = current->next;
    }  return NULL;
}void displayList(struct Node* head) {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }  struct Node* temp = head;
    printf("Linked List: ");
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    } printf("NULL\n");}
```

@Write a C program to implement the operation on (Circular Linked list):

i) Delete a node after given node.
ii) Find node with smallest data value
iii)Display the list
iv)Insert a node at the end of the list.

```
#include <stdio.h>
#include <stdlib.h>
struct Node { int data;
    struct Node* next;
};void insertAtEnd(struct Node** head, int data) {struct Node*
newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    if (*head == NULL) {
        newNode->next = newNode;
        *head = newNode;
    } else { struct Node* temp = *head;
        while (temp->next != *head) temp = temp->next;
        temp->next = newNode;
        newNode->next = *head;
    }}void deleteAfterNode(struct Node** head, int value) {if (*head ==
NULL) return;
    struct Node* current = *head;
    do { if (current->data == value) {
            struct Node* temp = current->next;
            if (temp == *head) *head = (*head == temp->next) ? NULL :
temp->next;
            current->next = temp->next;
            free(temp);
            return;
        }   current = current->next;
    } while (current != *head);
}struct Node* findSmallestNode(struct Node* head) {
    if (!head) return NULL;
    struct Node* smallest = head;
    struct Node* current = head->next;
    while (current != head) {
        if (current->data < smallest->data) smallest = current;
        current = current->next;
    }return smallest;
}void displayList(struct Node* head) {
    if (!head) { printf("List is empty.\n"); return; }
    struct Node* temp = head;
    do { printf("%d -> ", temp->data); temp = temp->next; } while
(temp != head);
    printf("(back to head)\n");
}int main() {
    struct Node* head = NULL;
    int choice, value;
 while (1) {
        printf("\n1. Insert at end\n2. Delete after node\n3. Find
smallest\n4. Display list\n5. Exit\nChoice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1: printf("Enter value: "); scanf("%d", &value);
insertAtEnd(&head, value); break;
            case 2: printf("Delete after value: "); scanf("%d", &value);
deleteAfterNode(&head, value); break;
            case 3: {struct Node* smallest = findSmallestNode(head);
                if (smallest) printf("Smallest value: %d\n", smallest->data);
                else printf("List is empty.\n");
                break;
            }case 4: displayList(head); break;
            case 5: return 0;
  default: printf("Invalid choice.\n");  }   }}
```

@Write a C program to stimulate linear queue as a linked list.

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

```
struct Node {
    int data;
    struct Node* next;
};struct Queue {
    struct Node* front;
    struct Node* rear;
};void initQueue(struct Queue* q) {
    q->front = q->rear = NULL;
}int isEmpty(struct Queue* q) {
    return (q->front == NULL);
}void enqueue(struct Queue* q, int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct
Node));
    newNode->data = value;
    newNode->next = NULL;
     if (q->rear == NULL) {
        q->front = q->rear = newNode;
        return;}
      q->rear->next = newNode;
    q->rear = newNode;}
int dequeue(struct Queue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty!\n");
        return -1;
    }  struct Node* temp = q->front;
    int value = temp->data;
    q->front = q->front->next;
    if (q->front == NULL) {
        q->rear = NULL;
    }  free(temp);
    return value;
}void display(struct Queue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty!\n");
        return;
    } struct Node* temp = q->front;
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }printf("NULL\n");
}int main() {
    struct Queue q;
    initQueue(&q);
enqueue(&q, 10);
    enqueue(&q, 20);
    enqueue(&q, 30);
 printf("Queue: ");
    display(&q);
  printf("Dequeued: %d\n", dequeue(&q));
    printf("Queue after dequeue: ");
    display(&q);
return 0;}
```

@Write a c program infix to postfix.

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

// Stack structure for storing operators
struct Stack {
    int top;
    char arr[MAX];
};void initStack(struct Stack* s) {
    s->top = -1;
}int isEmpty(struct Stack* s) {
    return s->top == -1;
}int precedence(char ch) {
    if (ch == '+' || ch == '-') return 1;
    if (ch == '*' || ch == '/') return 2;
    if (ch == '^') return 3;
    return 0;
}void infixToPostfix(char* infix, char* postfix) { struct Stack s;
    initStack(&s);
    int k = 0;
        for (int i = 0; infix[i] != '\0'; i++) {
        char ch = infix[i];
        if (isalnum(ch)) {
            postfix[k++] = ch;
        }   else if (ch == '(') {
            s.arr[++s.top] = ch;
        }   else if (ch == ')') {
            while (!isEmpty(&s) && s.arr[s.top] != '(') {postfix[k++] =
s.arr[s.top--];
            }s.top--;    }else {
            while (!isEmpty(&s) && precedence(s.arr[s.top]) >=
precedence(ch)) {  postfix[k++] = s.arr[s.top--];
            }s.arr[++s.top] = ch;
        }   }  while (!isEmpty(&s)) {
        postfix[k++] = s.arr[s.top--];
    }  postfix[k] = '\0';  // Null-terminate the postfix expression
}int main() {
    char infix[MAX], postfix[MAX];
      printf("Enter an infix expression: ");
    scanf("%s", infix);
     infixToPostfix(infix, postfix);
      printf("Postfix expression: %s\n", postfix);  return 0;}
```

@postfix to infix

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#define MAX 100
struct Stack {
    int top;
    char arr[MAX][MAX];
};void initStack(struct Stack* s) {
    s->top = -1;}
int isEmpty(struct Stack* s) {
    return s->top == -1;
}void postfixToInfix(char* postfix, char* infix) {  struct Stack s;
    initStack(&s);
       for (int i = 0; postfix[i] != '\0'; i++) {
        char ch = postfix[i];
        if (isalnum(ch)) {
            char operand[2] = {ch, '\0'};
            strcpy(s.arr[++s.top], operand);
        }  else {
            char operand2[MAX], operand1[MAX];
            strcpy(operand2, s.arr[s.top--]);
            strcpy(operand1, s.arr[s.top--]);
            char expr[MAX];
            sprintf(expr, "(%s%c%s)", operand1, ch, operand2);
            strcpy(s.arr[++s.top], expr);  }    }
strcpy(infix, s.arr[s.top]);
}int main() {
    char postfix[MAX], infix[MAX];
       printf("Enter a postfix expression: ");
    scanf("%s", postfix);
       postfixToInfix(postfix, infix);
    printf("Infix expression: %s\n", infix);
     return 0;}
```

@Write a program in C to implement queue ADT using linked list.

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};struct Queue {
    struct Node* front;
    struct Node* rear;
};void initQueue(struct Queue* q) {
    q->front = q->rear = NULL;
}int isEmpty(struct Queue* q) {
    return (q->front == NULL);
}void enqueue(struct Queue* q, int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct
Node));
    newNode->data = value;
    newNode->next = NULL;
      if (q->rear == NULL) {
        q->front = q->rear = newNode;
        return;    }
      q->rear->next = newNode;
    q->rear = newNode;
}int dequeue(struct Queue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty!\n");
        return -1;
    }  struct Node* temp = q->front;
    int value = temp->data;
    q->front = q->front->next;
   if (q->front == NULL) {
        q->rear = NULL;
    }  free(temp);
    return value;
}void display(struct Queue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty!\n");
        return;
    }   struct Node* temp = q->front;
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }   printf("NULL\n");
}int main() {
    struct Queue q;
    initQueue(&q);
   enqueue(&q, 10);
    enqueue(&q, 20);
    enqueue(&q, 30);
  printf("Queue: ");
    display(&q);
  printf("Dequeued: %d\n", dequeue(&q));
    printf("Queue after dequeue: ");
    display(&q);
  return 0;}
```