1. Write a C program to remove duplicate element from sorted Linked List.

Input:

 2 -> 3 -> 3 -> 4

Output:

 2 -> 3 -> 4

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

// Node structure
struct Node {
    int data;
    struct Node* next;
};

// Insert node at the end of linked list
void insert(struct Node** head_ref, int new_data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    struct Node* last = *head_ref;

    new_node->data = new_data;
    new_node->next = NULL;

    if (*head_ref == NULL) {
        *head_ref = new_node;
        return;
    }

    while (last->next != NULL)
        last = last->next;

    last->next = new_node;
    return;
}

// Remove duplicates from sorted linked list
```

```
void removeDuplicates(struct Node* head) {
    struct Node* current = head;
    struct Node* next_next;

    if (current == NULL)
        return;

    while (current->next != NULL) {
        if (current->data == current->next->data) {
            next_next = current->next->next;
            free(current->next);
            current->next = next_next;
        } else
            current = current->next;
    }
}

// Print linked list
void printList(struct Node* node) {
    while (node != NULL) {
        printf("%d  ", node->data);
        node = node->next;
    }
}

int main() {
    struct Node* head = NULL;

    // Insert elements
    insert(&head, 2);
    insert(&head, 3);
    insert(&head, 3);
    insert(&head, 4);

    printf("Input: ");
    printList(head);
    // Remove duplicates
    removeDuplicates(head);

    printf("\nOutput: ");
    printList(head);

    return 0;
}
```

===

2. Write a C program to rotate a doubly linked list by N nodes.
Input: (When N=2)
a b c d e
Output:
c d e a b
Input: (When N=4)
a b c d e f g h
Output:
e f g h a b c d

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

struct Node {
    char data;
    struct Node* prev;
    struct Node* next;
};

void insert(struct Node** head_ref, char new_data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = NULL;
    if (*head_ref == NULL) {
        new_node->prev = NULL;
        *head_ref = new_node;
        return;
    }

    struct Node* last = *head_ref;
    while (last->next != NULL)
        last = last->next;

    last->next = new_node;
    new_node->prev = last;
}

void rotate(struct Node** head_ref, int N) {
    if (N == 0)
        return;

    struct Node* current = *head_ref;
    int count = 1;

    while (count < N && current != NULL) {
        current = current->next;
        count++;
    }

    if (current == NULL)
        return;

    struct Node* Nth_node = current;

    while (current->next != NULL)
        current = current->next;

    current->next = *head_ref;
    (*head_ref)->prev = current;
    *head_ref = Nth_node->next;
    Nth_node->next->prev = NULL;
    Nth_node->next = NULL;
}

void printList(struct Node* node) {
    while (node != NULL) {
        printf("%c ", node->data);
        node = node->next;
    }
    printf("\n");
}

int main() {
    struct Node* head = NULL;

    insert(&head, 'a');
    insert(&head, 'b');
    insert(&head, 'c');
    insert(&head, 'd');
    insert(&head, 'e');

    printf("Input (N=2): ");
    printList(head);

    rotate(&head, 2);
    printf("Output (N=2): ");
    printList(head);

    insert(&head, 'f');
    insert(&head, 'g');
    insert(&head, 'h');

    printf("Input (N=4): ");
    printList(head);

    rotate(&head, 4);
    printf("Output (N=4): ");
    printList(head);

    return 0;
}
```

==

3. Write a C program to sort the elements of a queue in ascending order.

Input

4 2 7 5 1

Output

1 2 4 5 7

```
#include <stdio.h>

#define MAX_SIZE 100

int queue[MAX_SIZE];
int front = -1, back = -1;

void enqueue(int item) {
    if (back == MAX_SIZE - 1) {
        printf("Error: Queue is full\n");
        return;
    }
    if (front == -1) {
        front = 0;
    }
    back++;
    queue[back] = item;
}

int dequeue() {
    if (front == -1 || front > back) {
        printf("Error: Queue is empty\n");
        return -1;
    }
    int item = queue[front];
    front++;
    return item;
}

void display() {
    if (front == -1) {
        printf("Error: Queue is empty\n");
        return;
    }
    for (int i = front; i <= back; i++) {
        printf("%d ", queue[i]);
    }
    printf("\n");
}

void sort_queue_asc() {
    int i, j, temp;
    int n = back - front + 1;

    for (i = 0; i < n - 1; i++) {
        for (j = i + 1; j < n; j++) {
            if (queue[i] > queue[j]) {
                temp = queue[i];
                queue[i] = queue[j];
                queue[j] = temp;
            }
        }
    }
}

int main() {
    printf("Input some elements into the queue: 4 2 7 5 1\n");
    enqueue(4);
    enqueue(2);
    enqueue(7);
    enqueue(5);
    enqueue(1);

    printf("\nElements of the queue:\n");
    display();

    printf("\nSort the said queue:\n");
    sort_queue_asc();

    printf("\nElements of the sorted queue in ascending order:\n");
    display();

    return 0;
}
```

==

4. List all queue function operations available for manipulation of data elements in C

1. enqueue()
2. dequeue()
3. isEmpty()
4. isFull()
5. peek()
6. size()

==

5. Reverse the given string using stack
Input: (string)
"LetsLearn"
Output: (string)
"nraeLsteL

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

#define MAX_SIZE 100

struct Stack {
    int top;
    char elements[MAX_SIZE];
};

struct Stack* createStack() {
    struct Stack* stack = (struct Stack*)malloc(sizeof(struct Stack));
    stack->top = -1;
    return stack;
}

int isFull(struct Stack* stack) {
    return stack->top == MAX_SIZE - 1;
}

int isEmpty(struct Stack* stack) {
    return stack->top == -1;
}

void push(struct Stack* stack, char item) {
    if (isFull(stack)) {
        printf("Error: Stack is full\n");
        return;
    }
    stack->elements[++stack->top] = item;
}

char pop(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("Error: Stack is empty\n");
        return '\0';
    }
    return stack->elements[stack->top--];
}

void reverseString(char* str) {
    int length = strlen(str);
    struct Stack* stack = createStack();

    for (int i = 0; i < length; i++) {
        push(stack, str[i]);
    }

    for (int i = 0; i < length; i++) {
        str[i] = pop(stack);
    }
}

int main() {
    char input[] = "LetsLearn";

    printf("Input: %s\n", input);

    reverseString(input);

    printf("Output: %s\n", input);

    return 0;
}
```

==

6. Insert value in sorted way in a sorted doubly linked list. Given a sorted doubly linked list
and a value to insert, write a function to insert the value in sorted way.
Initial doubly linked list
3 5 8 10 12
Doubly Linked List after insertion of 9
3 5 8 9 10 12

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

struct Node {
    int data;
    struct Node* prev;
    struct Node* next;
};

struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}

void sortedInsert(struct Node** head_ref, int newData) {
    struct Node* newNode = createNode(newData);
    if (*head_ref == NULL || (*head_ref)->data >= newData) {
        newNode->next = *head_ref;
        if (*head_ref != NULL)
            (*head_ref)->prev = newNode;
        *head_ref = newNode;
    } else {
        struct Node* current = *head_ref;
        while (current->next != NULL && current->next->data < newData)
            current = current->next;

        newNode->next = current->next;
        if (current->next != NULL)
            current->next->prev = newNode;
        current->next = newNode;
        newNode->prev = current;
    }
}
```

```
void printList(struct Node* node) {
    while (node != NULL) {
        printf("%d ", node->data);
        node = node->next;
    }
    printf("\n");
}

int main() {
    struct Node* head = NULL;

    sortedInsert(&head, 3);
    sortedInsert(&head, 5);
    sortedInsert(&head, 8);
    sortedInsert(&head, 10);
    sortedInsert(&head, 12);

    printf("Initial doubly linked list: ");
    printList(head);

    int valueToInsert = 9;
    printf("Value to insert: %d\n", valueToInsert);

    sortedInsert(&head, valueToInsert);

    printf("Doubly linked list after insertion of %d: ", valueToInsert);
    printList(head);

    return 0;
}
```

==

7. Write a C program to insert/delete and count the number of elements in a queue.

Expected Output:

```
Initialize a queue!
Check the queue is empty or not? Yes
```

```
Number of elements in queue: 0
Insert some elements into the queue:
Queue elements are: 1 2 3
Number of elements in queue: 3
Delete two elements from the said queue:
Queue elements are: 3
Number of elements in queue: 1
Insert another element into the queue:
Queue elements are: 3 4
Number of elements in the queue: 2
```

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

// Queue structure
struct Queue {
    int front, rear, capacity;
    int *array;
};

// Function to create a queue of given capacity
struct Queue *createQueue(int capacity) {
    struct Queue *queue = (struct Queue*)malloc(sizeof(struct Queue));
    queue->capacity = capacity;
    queue->front = queue->rear = -1;
    queue->array = (int*)malloc(queue->capacity * sizeof(int));
    return queue;
}

// Function to check if the queue is full
int isFull(struct Queue *queue) {
    return (queue->rear == queue->capacity - 1);
}

// Function to check if the queue is empty
int isEmpty(struct Queue *queue) {
    return (queue->front == -1);
}

// Function to insert an element into the queue
void enqueue(struct Queue *queue, int item) {
    if (isFull(queue))
        return;
    if (isEmpty(queue))
        queue->front = 0;
    queue->rear++;
    queue->array[queue->rear] = item;
}

// Function to delete an element from the queue
void dequeue(struct Queue *queue) {
    if (isEmpty(queue))
        return;
    if (queue->front == queue->rear)
        queue->front = queue->rear = -1;
    else
        queue->front++;
}

// Function to count the number of elements in the queue
int count(struct Queue *queue) {
    if (isEmpty(queue))
        return 0;
    return (queue->rear - queue->front + 1);
}

// Function to display the elements of the queue
void display(struct Queue *queue) {
    int i;
    if (isEmpty(queue)) {
        printf("Queue is empty.\n");
        return;
    }
    printf("Queue elements are: ");
    for (i = queue->front; i <= queue->rear; i++)
        printf("%d ", queue->array[i]);
    printf("\n");
}

int main() {
    struct Queue *queue = createQueue(100); // Initialize queue with
capacity 100

    printf("Initialize a queue!\n");
    printf("Check the queue is empty or not?");
    if (isEmpty(queue))
         printf("Yes\n");
    else
        printf("No\n");
    int n,a[20];
    printf("Number of elements in queue: %d\n", count(queue));
    printf("Number of elements to Be inserted ");scanf("%d",&n);
    printf("Insert some elements into the queue:\n");
    int *pt=a;
    for(int i=0 ;i<n;i++)
        {
            scanf("%d",pt+i);
        }
    for(int i=0 ;i<n;i++)
        {
            enqueue(queue, *(pt+i));
        }

    display(queue);
    printf("Number of elements in queue: %d\n", count(queue));

    printf("Delete two elements from the said queue:\n");
    dequeue(queue);
    dequeue(queue);
    display(queue);
    printf("Number of elements in queue: %d\n", count(queue));

    printf("Insert another element into the queue:\n");
    enqueue(queue, 4);
    display(queue);
    printf("Number of elements in the queue: %d\n", count(queue));

    free(queue->array);
    free(queue);

    return 0;
}
```

===

8. Write a C program to Find whether an array is a subset of another array.

Input:

arr1[] = {11, 1, 13, 21, 3, 7}, arr2[] = {11, 3, 7, 1}

Output:

arr2[] is a subset of arr1[]

Input:

arr1[] = {10, 5, 2, 23, 19}, arr2[] = {19, 5, 3}

Output:

arr2[] is not a subset of arr1[]

```
#include <stdio.h>

// Function to check if arr2[] is a subset of arr1[]
int isSubset(int arr1[], int arr2[], int m, int n) {
    int i = 0, j = 0;
    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++) {
            if(arr2[i] == arr1[j])
                break;
        }
        if (j == m)
            return 0;
    }
    return 1;
}

int main() {
    int arr1[] = {11, 1, 13, 21, 3, 7};
    int arr2[] = {11, 3, 7, 1};
    int m = sizeof(arr1) / sizeof(arr1[0]);
    int n = sizeof(arr2) / sizeof(arr2[0]);
    if (isSubset(arr1, arr2, m, n))
        printf("arr2[] is a subset of arr1[]\n");
    else
        printf("arr2[] is not a subset of arr1[]\n");

    int arr3[] = {10, 5, 2, 23, 19};
    int arr4[] = {19, 5, 3};
    m = sizeof(arr3) / sizeof(arr3[0]);
    n = sizeof(arr4) / sizeof(arr4[0]);
    if (isSubset(arr3, arr4, m, n))
        printf("arr4[] is a subset of arr3[]\n");
    else
        printf("arr4[] is not a subset of arr3[]\n");

    return 0;
}
```

==